42nd International Symposium on Mathematical Foundations of Computer Science

MFCS 2017, August 21–25, 2017, Aalborg, Denmark

Edited by

Kim G. Larsen
Hans L. Bodlaender
Jean-François Raskin

Editors

Kim G. Larsen
Aalborg University
Aalborg, Denmark
kgl@cs.aau.dk

Hans L. Bodlaender
Eindhoven University of Technology
Eindhoven, Netherlands
h.l.bodlaender@tue.nl

Jean-François Raskin
Université libre de Bruxelles
Brussels, Belgium
jraskin@ulb.ac.be

ACM Classification 1998
F. Theory of Computation

ISBN 978-3-95977-046-0

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/978-3-95977-046-0.

Publication date
November, 2017

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.2017.0

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

42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017).
Editors: Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Invited Talks

Foreword

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 Jabłonna (near Warsaw, Poland). Since then, the conference traditionally moved between the Czech Republic, Slovakia, and Poland. A few years ago, the conference started traveling around Europe: in 2013 it was held in Austria, in 2014 in Hungary, in 2015 in Italy, and most recently, in 2016, the conference returned to Poland. We are happy that this year MFCS is organized in Denmark, the most northern place yet to host MFCS.

Over 200 abstracts were submitted, of which 192 materialized as papers, of which 80 were finally accepted. The authors of the submitted papers represent nearly 40 countries. The authors first registered their papers' abstracts (by the 24th of April, 2017) and only then their content (by the 28th of April, 2017). 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 Glynn Winskel (University of Cambridge, UK), Michał Pilipczuk (University of Warsaw, Poland), Rasmus Pagh (IT University of Copenhagen, Denmark), Nicolas Markey (CNRS, Rennes, France), and Philippe Schnoebelen (LSV – CNRS & ENS Cachan, Université Paris-Saclay, France). We would like to thank them deeply for their contributions and their time.

This is the second 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.

Kim G. Larsen
Hans L. Bodlaender
Jean-François Raskin

42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017).
Editors: Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Conference Organization

Program Commitee

Lars Birkedal	Aarhus University, Denmark
Manuel Bodirsky	TU Dresden, Germany
Hans L. Bodlaender	Eindhoven University of Technology, Netherlands
Udi Boker	Interdisciplinary Center (IDC) Herzliya, Israel
Patricia Bouyer	LSV, CNRS & ENS Cachan, Université Paris Saclay, France
Franck Cassez	Macquarie University, Australia
Rocco De Nicola	IMT - School for Advanced Studies Lucca, Italy
Rod Downey	Victoria University of Wellington, New Zealand
Manfred Droste	Leipzig University, Germany
Vojtech Forejt	Oxford University, UK
Paweł Gawrychowski	University of Haifa, Israel
Raffaella Gentilini	University of Perugia, Italy
Kasper Green Larsen	MADALGO, Aarhus University, Denmark
Kim Guldstrand Larsen	Aalborg University, Denmark
Petteri Kaski	Helsinki Institute for Information Technology, Aalto University, Finland
Bartek Klin	University of Warsaw, Poland
Dexter Kozen	Cornell University, USA
Stephan Kreutzer	Technical University Berlin, Germany
Alexander Kurz	University of Leicester, UK
Martin Lange	University of Kassel, Germany
Sławomir Lasota	University of Warsaw, Poland
Axel Legay	IRISA/INRIA, Rennes, France
Christof Löding	RWTH Aachen, Germany
Radu Mardare	Aalborg University, Denmark
Roland Meyer	TU Braunschweig, Germany
Matteo Mio	CNRS–ENS-Lyon, France
Luca Moscardelli	University of Chieti-Pescara, Italy
Aniello Murano	Università degli Studi di Napoli Federico II, Italy
Prakash Panangaden	McGill University, Canada
Dana Pardubska	Comenius University, Slovakia
Ramamohan Paturi	University of California, USA
Arno Pauly	Université Libre de Bruxelles, Belgium
Doron Peled	Bar Ilan University, Israel
Damien Pous	CNRS–ENS Lyon, France
Jean-François Raskin	Université Libre de Bruxelles, Belgium
Jörg Rothe	Universität Düsseldorf, Germany
Pierre-Yves Schobbens	University of Namur, Belgium
Bettina Speckmann	TU Eindhoven, Netherlands
Sam Staton	University of Oxford, UK
Hans Raj Tiwary	Charles University, Czech Republic
Tarmo Uustalu	Institute of Cybernetics, Tallinn University of Technology, Estonia
Peter Van Emde Boas	ILLC-FNWI-Universiteit van Amsterdam (emeritus), Netherlands
Jiri Wiedermann	Academy of Sciences, Czech Republic

42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017).
Editors: Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Aaronson, Scott
Abd Alrahman, Yehia
Aceto, Luca
Adams, Michael D.
Almagor, Shaull
Aman, Bogdan
Amato, Gianluca
Angelini, Patrizio
Anshu, Anurag
Arcucci, Rossella
Asada, Kazuyuki
Avni, Guy
Babai, Laszlo
Bacci, Giorgio
Bacci, Giovanni
Balaji, Nikhil
Banik, Aritra
Barbieri, Sebastián
Baumeister, Dorothea
Beckmann, Arnold
Bekos, Michael
Ben Said, Najah
Beretta, Stefano
Berkholz, Christoph
Bienkowski, Marcin
Bilò, Vittorio
Biondi, Fabrizio
Bodor, Bertalan
Boža, Vladimír
Brazdil, Tomas
Bredereck, Robert
Brejová, Broňa
Brunet, Paul
Bruse, Florian
Bulian, Jannis
Béal, Marie-Pierre
Cadilhac, Michaël
Capobianco, Silvio
Carayol, Arnaud
Caucal, Didier
Chalermsook, Parinya
Chang, Huilan
Chatain, Thomas
Chen, Hubie
Chen, Yijia
Chini, Peter
Chistikov, Dmitry
Choffrut, Christian
Clemente, Lorenzo
Clifford, Raphael
Comin, Carlo
Cosme Llópez, Enric
Czerwiński, Wojciech
D'Angelo, Gianlorenzo
D'Emidio, Mattia
Darais, David
Dartois, Luc
Das, Anupam
Della Monica, Dario
Dereniowski, Dariusz
Detinko, Alla
Di Stasio, Antonio
Divakaran, Srikrishnan
Doczkal, Christian
Doyen, Laurent
Dudek, Bartlomiej
Duerr, Christoph
Duong, Tan
Durier, Adrien
Duris, Pavol
Dvorak, Zdenek
Dvořák, Pavel
Dvořák, Wolfgang
Ehlers, Rüdiger
Faella, Marco
Fahrenberg, Uli
Fanelli, Angelo
Ferrara, Michael
Fijalkow, Nathanaël
Forisek, Michal
Frati, Fabrizio
Freeman, Rupert
Freydenberger, Dominik D.
Friedler, Ophir
Froese, Vincent
Fuegger, Matthias
Furbach, Florian
Furber, Robert
Förster, Klaus-Tycho
Gasieniec, Leszek
Giesen, Joachim
Given-Wilson, Thomas
Godin, Thibault
Goncharov, Sergey
Greenberg, Noam
Greiner, Johannes
Grellois, Charles
Grigoriev, Alexander
Grippo, Luciano
Guillon, Pierre
Guo, Jiong
Halfon, Simon
Hansen, Thomas Dueholm
Harrenstein, Paul
Heindel, Tobias
Heizmann, Matthias
Herbreteau, Frédéric
Hermelin, Danny
Heuser, Annelie
Hlineny, Petr
Holik, Lukas
Hon, Wing-Kai
Huang, Zengfeng
Hundeshagen, Norbert
Hutagalung, Milka
Hölzl, Rupert
Ibsen-Jensen, Rasmus
Iwata, Yoichi
Jansen, Thomas
Jeandel, Emmanuel
Jecker, Ismaël
Jeż, Artur
Johannsen, Jan
Jugé, Vincent
Jukna, Stasys
Katreniakova, Jana
Kazda, Alexandr
Kernberger, Daniel
Kieronski, Emanuel
Klauck, Hartmut
Knop, Dušan
Kocay, William
Kociumaka, Tomasz
Kopczynski, Eryk
Koroth, Sajin
Koslowski, Jürgen
Koster, Arie
Kostolányi, Peter

Kralovic, Rastislav
Krebs, Andreas
Kreiker, Joerg
Krichen, Moez
Kufleitner, Manfred
Kumar, Nirman
Kuperberg, Denis
Kuske, Dietrich
Lagerqvist, Victor
Lahiri, Abhiruk
Lanese, Ivan
Laurent, Fribourg
Lauri, Juho
Lazic, Ranko
Lehtonen, Erkko
Lombardy, Sylvain
Lorber, Florian
Loreti, Michele
Lukoťka, Robert
Löffler, Maarten
Makowsky, Johann
Malvone, Vadim
Mamino, Marcello
Mamouras, Konstantinos
Manthey, Bodo
Marino, Andrea
Marković, Petar
Martens, Wim
Maubert, Bastien
Maushagen, Cynthia
Mayhew, Dillon
Mazowiecki, Filip
Mccartin, Catherine
McKenzie, Pierre
Melideo, Giovanna
Mennicke, Stephan
Mercas, Robert
Merkle, Wolfgang
Merro, Massimo
Meulemans, Wouter
Michalewski, Henryk
Michaliszyn, Jakub
Michell, Joseph
Misra, Neeldhara
Monaco, Gianpiero
Mosca, Raffaele
Mottet, Antoine
Muskalla, Sebastian
Neider, Daniel
Neuen, Daniel
Neugebauer, Daniel
Neveling, Marc
Nguyen, Nhan-Tam
Ochremiak, Joanna
Okamoto, Yoshio
Okhotin, Alexander
Oliveira, Igor Carboni
Opršal, Jakub
Ostertág, Richard
Ostropolski-Nalewaja, Piotr
Ouaknine, Joel
Pascual, Fanny
Paulusma, Daniel
Pecatte, Timothée
Pedersen, Christian Storm
Pedersen, Mathias Ruggaard
Perarnau, Home
Perez, Guillermo
Peron, Adriano
Piazza, Carla
Pilipczuk, Marcin
Pilz, Alexander
Pinault, Laureline
Piperno, Adolfo
Plandowski, Wojciech
Portier, Natacha
Potapov, Igor
Poulsen, Danny Bøgsted
Prezza, Nicola
Pulina, Luca
Quilbeuf, Jean
Radhakrishna, Arjun
Raman, Venkatesh
Ramsay, Steven
Ramyaa, Ramyaa
Reimann, Jan
Rey, Lisa
Reynier, Pierre-Alain
Richerby, David
Riondato, Matteo
Roberson, David
Roberts, Matt
Rubin, Sasha
Rutter, Ignaz
S., Krishna
Saivasan, Prakash
Sammartino, Matteo
Sandu, Gabriel
Sattler, Christian
Sauro, Luigi
Schadrack, Hilmar
Scheder, Dominik
Schneider, Klaus
Sciavicco, Guido
Selker, Ann-Kathrin
Shirmohammadi, Mahsa
Siebertz, Sebastian
Silva, Alexandra
Simpson, Stephen
Sintos, Stavros
Skrzypczak, Michał
Sorrentino, Loredana
Stepanovs, Igors
Stephan, Frank
Straszak, Damian
Streib, Amanda Pascoe
Stützle, Thomas
Subramani, K.
Thilikos, Dimitrios
Tribastone, Mirco
Trubiani, Catia
Tschaikowski, Max
Turrini, Andrea
Uznański, Przemysław
van der Zanden, Tom
van Goethem, Arthur
van Iersel, Leo
Velaj, Yllka
Veltri, Niccolò
Vignudelli, Valeria
Vigny, Alexandre
Vikas, Narayan
Vinar, Tomas
Vinci, Cosimo
Viola, Caterina
Volk, Ben Lee
Wahlström, Magnus
Watanabe, Osamu
Wijs, Anton
Williams, Ryan
Witt, Carsten
Wolff, Sebastian
Wood, David R.
Worrell, James

Wrochna, Marcin	Wrona, Michał	Xue, Jie
Yamamoto, Masaki	Zahn, Philipp	Zamdzhiev, Vladimir
Zehavi, Meirav	Zendra, Olivier	Zetzsche, Georg
Zhang, Yao	Živný, Stanislav	

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

Does Looking Inside a Circuit Help?

Russell Impagliazzo[1], Valentine Kabanets[2], Antonina Kolokolova[3], Pierre McKenzie[4], and Shadab Romani[5]

1 **University of California, San Diego, La Jolla, CA, USA**
russell@cs.ucsd.edu
2 **Simon Fraser University, Burnaby, BC, Canada**
kabanets@cs.sfu.ca
3 **Memorial University of Newfoundland, St. John's, NL, Canada**
kol@cs.mun.ca
4 **Université de Montréal, Montréal, QC, Canada**
mckenzie@iro.umontreal.ca
5 **Simon Fraser University, Burnaby, BC, Canada**
sromani@sfu.ca

Abstract

The Black-Box Hypothesis, introduced by Barak et al. [5], states that any property of boolean functions decided efficiently (e.g., in BPP) with inputs represented by circuits can also be decided efficiently in the black-box setting, where an algorithm is given an oracle access to the input function and an upper bound on its circuit size. If this hypothesis is true, then $\mathsf{P} \neq \mathsf{NP}$. We focus on the consequences of the hypothesis being false, showing that (under general conditions on the structure of a counterexample) it implies a non-trivial algorithm for Circuit-SAT. More specifically, we show that if there is a property F of boolean functions such that F has high sensitivity on some input function f of subexponential circuit complexity (which is a sufficient condition for F being a counterexample to the Black-Box Hypothesis), then Circuit-SAT is solvable by a subexponential-size circuit family. Moreover, if such a counterexample F is symmetric, then Circuit-SAT $\in \mathsf{P/poly}$. These results provide some evidence towards the conjecture (made in this paper) that the Black-Box Hypothesis is false if and only if Circuit-SAT is easy.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Black-Box Hypothesis, Rice's theorem, circuit complexity, SAT, sensitivity of boolean functions, decision tree complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.1

1 Introduction

Given access to a boolean function $f\colon \{0,1\}^n \to \{0,1\}$, how fast can we decide if $f \not\equiv 0$? If we can only access f as an oracle (i.e., in the "black-box" fashion), then it is well-known that one needs time $\Omega(2^n)$ for any deterministic or randomized algorithm (and time $\Omega(2^{n/2})$ for any quantum algorithm). What if f is computable by some small boolean circuit C, and we are given this circuit C (i.e., we can access f in the "white-box" fashion)? Then the question of deciding if $f \not\equiv 0$ is exactly the famous Circuit-SAT problem, and no non-trivial complexity lower bounds are known.

One possible approach to proving that $\mathsf{P} \neq \mathsf{NP}$ is to argue that being given an actual small circuit C computing a given boolean function f does not help much, compared to being given just oracle access to f, and being told the size of C. This could be formalized

as the *Black-Box Hypothesis (BBH)* (introduced by Barak et al. [5] as "Scaled-down Rice's Theorem" conjecture), which can be informally stated as follows:

> If a property F of boolean functions can be decided efficiently on circuits computing input functions, then F can also be decided efficiently in the black-box setting (that is, given oracle access to the input function and its circuit size bound).

If this hypothesis is true, then, for $F = \{f\colon \{0,1\}^n \to \{0,1\} \mid f \not\equiv 0\}$, we conclude that Circuit-SAT cannot be solved efficiently, since there are exponential lower bounds for deciding F in the black-box setting.

So proving the BBH is hard, as it would imply that $\mathsf{P} \neq \mathsf{NP}$. The hypothesis may well be false. Barak et al. [5] already proved that a version of the BBH (for promise problems) is false, assuming that one-way functions exist. Can we just disprove it then?

In this paper, we give some evidence that disproving the BBH is also hard, as it would have non-trivial algorithmic applications for Circuit-SAT. Note that if Circuit-SAT is efficiently solvable, then, as observed above, the Black-Box Hypothesis must be false. We conjecture that the converse implication also holds. Thus we conjecture the following:

> The BBH is false iff Circuit-SAT has a (somewhat) efficient algorithm.

We make a step towards proving this conjecture by showing that if the BBH fails *in a particular way*, then Circuit-SAT can be decided by a nonuniform family of subexponential-size circuits, which would disprove the nonuniform analogue of the Exponential-Time Hypothesis (ETH) of [13].

1.1 Our results

Before stating our results formally, let us discuss what it means for the BBH to fail. Clearly, if the BBH fails, there is a property F that is easy in the white-box setting (say, is in BPP), but requires superpolynomial complexity in the black-box setting. Note that for n-variate boolean functions f of circuit complexity $2^{\Omega(n)}$, there can't be any superpolynomial gap between the white-box and black-box complexities of deciding a given property F. This is because a white-box algorithm has to look at the input circuit, which is of size at least $2^{\Omega(n)}$, and the black-box algorithm can read the entire truth-table of f, build a trivial circuit of size about 2^n, and then just simulate the white-box algorithm on it, running in overall time at most $\mathsf{poly}(2^n)$. Thus any "magic" speed-up that we get for a property F violating the BBH must necessarily manifest itself over "easy" inputs, boolean n-variate functions f of circuit complexity at most $2^{o(n)}$. In other words, any black-box algorithm for F must be "slow" even if we care only about inputs f of low circuit complexity.

Recall that the sensitivity of a function F is the maximum, over all its inputs $x \in \{0,1\}^N$, of the number of positions $i \in [N]$ such that $F(x) \neq F(x^i)$, where x^i is x with the ith bit flipped. It is well-known that every F with sensitivity s requires $\Omega(s)$ queries to decide by any (randomized) black-box algorithm [15]. Thus, a sufficient condition for any black-box algorithm deciding F to be "slow" (taking time at least T) is that F has "high" sensitivity (at least $\Omega(T)$). In fact, the same argument from [15] actually implies that if F has a sensitive input x^*, then F requires large query complexity even when restricted to the inputs $x^*, (x^*)^1, (x^*)^2, \ldots, (x^*)^N$. The latter can be used to show (see Theorem 3.6 below) that a *sufficient condition for any black-box algorithm deciding F to be "slow" on all inputs f of subexponential circuit complexity* is the following:

> there exists a function $f^*\colon \{0,1\}^n \to \{0,1\}$ of circuit complexity $2^{o(n)}$ such that F has "high" sensitivity at f^*.

An important feature of the OR function (which explains why it requires high black-box complexity) is the existence of a highly sensitive input, the all-zero string. Moreover, this sensitive input has a very low circuit complexity (as a boolean function). We show that if the BBH fails because of a property F with similar conditions (i.e., that F has an "easy" but "highly sensitive" input), then Circuit-SAT admits a non-trivial algorithm.

▶ **Theorem 1.1** (Main theorem: Informal version)**.** *Suppose there is a property F of n-variate boolean functions such that*

1. *F is decidable in* BPP *in the white-box setting, but,*
2. *for almost all n, F has an input $f^*\colon \{0,1\}^n \to \{0,1\}$ of sensitivity $2^{\Omega(n)}$ and of circuit complexity $2^{o(n)}$ (which implies that F requires exponential time $2^{\Omega(n)}$ to decide in the black-box setting, even on inputs f of circuit complexity $2^{o(n)}$).*

Then Circuit-SAT *for n-input circuits of size at most $2^{o(n)}$ can be decided by a nonuniform family of circuits of size $2^{o(n)}$.*

Intuitively, Theorem 1.1 says that if the BBH fails in a strong way for some property F, with an exponential gap between the white-box and the black-box complexities, so that the high black-box complexity of F can be explained through the existence of a highly sensitive input f^* (of relatively low circuit complexity), then Circuit-SAT is decidable by a subexponential-time nonuniform algorithm.

We also observe that the assumption of Theorem 1.1 holds for any property F violating the BBH whenever F is one of the following:

- F is a symmetric function, or
- F is a subset of easy functions (i.e., $F \subseteq \{f \mid size(f) \leq 2^{o(n)}\}$).

Hence, if a counterexample to the BBH is of this kind, then Circuit-SAT is easy for nonuniform algorithms.

Finally, for the special case of *monotone* properties F, we get a version of Theorem 1.1 where it suffices to assume that a sensitive input in item (2) of Theorem 1.1 has just superpolynomial sensitivity $s > n^{\omega(1)}$ and circuit complexity $s^{o(1)}$ (rather than requiring an exponential sensitivity $s \geq 2^{\Omega(n)}$). More precisely, we prove the following.

▶ **Theorem 1.2** (Monotone Properties)**.** *Let F be a monotone property such that*

1. *F is decidable in* BPP *in the white-box setting, but,*
2. *for almost all n, F has an input $f^*\colon \{0,1\}^n \to \{0,1\}$ of sensitivity $s \geq n^{\omega(1)}$ and of circuit complexity $s^{o(1)} \geq \mathsf{poly}(n)$ (which implies that F requires superpolynomial time to decide in the black-box complexity setting, even on inputs of circuit complexity $s^{o(1)}$).*

Then Circuit-SAT *for n-input circuits of size at most $2^{o(n)}$ can be decided by a nonuniform family of circuits of size $2^{o(n)}$.*

We also use a "win-win" argument to show the following: If a monotone property is a counterexample to the Block-box Hypothesis (with appropriate parameters), then either Circuit-SAT is nonuniformly easy infinitely often, or BPP $\subseteq$ NP (see Theorem 5.2).

1.2 Related work

The Black-Box Hypothesis has its roots in a classical result of computability theory, Rice's theorem, which says that any non-trivial property of languages accepted by Turing machines is undecidable. There are two ways of interpreting Rice's theorem: (1) Given a Turing machine M, the only thing one can do is to run it, or (2) the Halting problem is the easiest non-trivial property of languages of Turing machines, in the sense that if any non-trivial property is decidable, then so is the Halting problem.

The intuition that it may be hard to understand what an algorithm does by looking at the algorithm description naturally extends to the class of non-uniform algorithms (i.e., circuits). The focus of this paper is on the second interpretation of Rice's theorem, with Circuit-SAT as a complexity counterpart of the Halting problem. In other words, we would like to show any "non-trivial" counterexample to the Black-Box Hypothesis implies a somewhat efficient algorithm for SAT.

There have been several attempts to scale down Rice's theorem to the complexity-theoretic realm, with different notions of 'non-trivial' and 'hard'. In Rice's theorem, 'non-trivial' means neither F nor $\bar{F}$ is empty, and 'hard' = undecidable. Borchert and Stephan [6] pioneered a line of research that looked at counting properties of circuits and stated an analogue of Rice's theorem for such properties: if a counting property is non-empty, then it is UP-hard. There, a property F is a counting property if it only depends on the number of solutions (i.e., F is a symmetric function). Subsequently, Hemaspaandra and Rothe [10] and Hemaspaandra and Thakur [11] improved the hardness result, obtaining a version of Rice's theorem with NP-hardness.

Barak et al. [5] also look at the properties of boolean functions computed by circuits, but consider a property trivial if it can be decided by checking the circuit value on relatively few points. That is, in their setting, the semantic property $f(00\ldots0) = f(11\ldots1)$ is trivial, but $\exists x\ f(x) = 1$ is not. Their 'Scaled-down Rice's theorem' conjecture states that every property of boolean functions f that can be computed in BPP given a circuit for f can be also computed in comparable probabilistic polynomial time given only oracle access to f and an upper bound on its circuit complexity. There is a clear relation to obfuscation: if it were possible to produce a circuit for any f so garbled that access to it is not much better than the black-box access, that would prove the conjecture. However, in the same paper they show impossibility of achieving such obfuscation. Nonetheless, [5] is able to disprove a certain "promise" version of the conjecture, under the assumption that one-way functions exist (using a special family of unobfuscatable circuits). The main statement, which we will call here 'the Black-Box Hypothesis', remains open.

1.3 Our techniques

Our starting point is the isolation lemma of Valiant and Vazirani [19], which can be interpreted to say that any white-box BPP algorithm deciding the property $F = XOR$ yields a BPP algorithm for Circuit-SAT. This can be extended to any property F computing a symmetric function, at the expense of introducing a small (polynomial) amount of nonuniformity. The main idea is to take advantage of the existence of a very sensitive input f for any symmetric property F. (For example, for the case of XOR, every input $f\colon \{0,1\}^n \to \{0,1\}$ has maximum sensitivity 2^n. In general, every symmetric F has a polysize input f of sensitivity at least $2^n/2$.)

Suppose that $f\colon \{0,1\}^n \to \{0,1\}$ is such a sensitive input for the property F, and moreover, suppose that f is computable by a small circuit C_f (say of $\mathsf{poly}(n)$ size). To decide if a given circuit C on n inputs is satisfiable, we first use the Valiant-Vazirani result to get from C a new circuit C' such that C' is uniquely satisfiable if C is satisfiable, and C' is unsatisfiable otherwise. By XORing the circuits C_f and C', we get a new (small) circuit that leaves f unchanged if C is unsatisfiable, and flips f in exactly one location if C is satisfiable. If the flipped location happens to land among the sensitive locations of f, we can detect this by running our assumed white-box algorithm on $C_f \oplus C'$ and noting that its output is different from that on C_f. To make sure that the flipped location is among the sensitive ones for f, we consider a random-shift version of C' so that its unique satisfying assignment

(if it exists) will be in a uniformly random location. As, by assumption, f has very many sensitive locations, this randomization will ensure that we detect if C is satisfiable with high probability. The runtime of the described algorithm is polynomial in the sizes of C_f and C. We think of a small circuit C_f as nonuniform advice, thereby getting a non-trivial nonuniform algorithm for Circuit-SAT.

The (nonuniform) algorithm for Circuit-SAT described above achieves high success probability in case a sensitive input $f\colon \{0,1\}^n \to \{0,1\}$ (provided as advice via a small circuit computing f) has very large sensitivity $s \geq \Omega(2^n)$. What if the sensitivity is only as large as $2^{\Omega(n)}$? (Such a lower bound is the best one can hope for if one assumes the Sensitivity Conjecture and that the given property F has exponential decision tree complexity.) In this case, our described algorithm would have success probability only about $2^{-\delta n}$, for some constant $0 < \delta < 1$, for solving Circuit-SAT on n-input circuits. However, if the algorithm runs in (non-uniform) time at most $2^{o(n)}$ (which will happen if the advice circuit C_f is of size at most $2^{o(n)}$), then we can use the amplification technique of Paturi and Pudlák [17] to get a new algorithm in non-uniform time $2^{o(n)}$ that succeeds with probability 1.

For the special case of monotone properties F, we show how to make do with even smaller sensitivity assumption on the advice function f, getting a subexponential-size Circuit-SAT algorithm for any superpolynomial sensitivity $s > n^{\omega(1)}$. The idea is to use hashing (which is also the main ingredient in the aforementioned result of [17]).

If we don't assume that a sensitive input f for a given property F would have a small circuit, we can still say something interesting by applying a "win-win" argument. Informally, we get that if F has sensitive inputs and an efficient white-box algorithm, then either Circuit-SAT is nonuniformly easy (in subexponential size, infinitely often), or we get an efficient "hardness tester": a polytime algorithm that accepts only truth tables of boolean functions of exponential circuit complexity, and accepts at least one such truth table. Getting such a hardness tester is a highly non-trivial task, and is not known unconditionally. Once you have this tester, you can, for example, conclude that BPP $\subseteq$ NP, using standard "hardness-randomness" trade-offs [16, 4, 14].

Remainder of the paper. We give some basic definitions and facts in Section 2. We state and discuss the Black-Box Hypothesis in Section 3. We prove Theorem 1.1 in Section 4. In Section 5, we consider the special case of monotone properties as counterexamples to the Black-Box Hypothesis, getting a proof of Theorem 1.2. In Section 6, we consider the case of properties defined using succinct versions of the Minimal Circuit Size Problem (MCSP). We consider some variants of the BBH for restricted circuit classes in Section 7. We conclude with some open problems in Section 8. This is a conference version of the paper, with some proofs omitted due to space limitations. The full version can be found online as [12].

2 Preliminaries

The truth table of a boolean function $f\colon \{0,1\}^n \to \{0,1\}$ is denoted by $tt(f)$. With a boolean circuit C on n inputs, we associate the boolean function $f_n = [C]$ computed by C. Slightly abusing the notation, we use $tt(C)$ to denote the truth table of a boolean function computed by the circuit C. A standard encoding of C as a binary string is denoted $desc(C)$.

A *property of boolean functions* is a function $F\colon \{0,1\}^{2^n} \to \{0,1\}$, where strings over $\{0,1\}^{2^n}$ are interpreted as truth tables of boolean functions on n variables, for every n. A *meta-language* over circuits corresponding to a property F is $L_F = \{desc(C) \mid C \text{ is a boolean circuit and } tt(C) \in F\}$. In particular, if L_F is a meta-language over circuits, then for any circuits C_1 and C_2, if $[C_1] = [C_2]$ then $C_1 \in L_F \Leftrightarrow C_2 \in L_F$.

The size of a boolean circuit C is the number of gates plus the number of wires. Let $size(f) = \min_{C,[C]=f} |C|$. We say that $f \in \mathsf{SIZE}(t(n))$ if $size(f) \leq t(n)$.

We denote by $\mathsf{Circuit\text{-}SAT}_{n,m}$ the problem of deciding the satisfiability of a given n-input circuit of size at most m. For a time bound $t = t(n)$, we denote by $\mathsf{RTIME}(t)$ the class of languages decidable by randomized algorithms, with one-sided error at most $1/2$, in time t; as usual, $\mathsf{RP} = \mathsf{RTIME}(\mathsf{poly})$. For an advice size function $a = a(n)$, we denote by $\mathsf{RTIME}(t)/a$ the class of languages decidable by an $\mathsf{RTIME}(t)$ algorithm, given the correct advice of size at most a.[1]

For a function $F\colon \{0,1\}^N \to \{0,1\}$, with $N = 2^n$, we can think of inputs to F as truth tables of n-variate boolean functions $f\colon \{0,1\}^n \to \{0,1\}$. For a circuit size bound $t = t(n)$, we define the *randomized decision tree complexity of F on inputs of complexity at most t*, denoted $Rt_t(F)$, as the minimal depth of a randomized decision tree deciding F, with error probability at most $1/3$, on all inputs $f\colon \{0,1\}^n \to \{0,1\}$ of $size(f) \leq t(n)$.

A boolean function $f\colon \{0,1\}^n \to \{0,1\}$ is *sensitive* on the ith bit of input x if flipping that bit changes the value of $f(x)$. Sensitivity of f on input $x \in \{0,1\}^n$, denoted by $\mathsf{sens}(f,x)$, is the number of bits in x to which f is sensitive. The sensitivity of f, denoted $\mathsf{sens}(f)$, is $\max_{x\in\{0,1\}^n} \mathsf{sens}(f,x)$.

Simon's lemma [18] gives a weak lower bound on $\mathsf{sens}(f)$. We will use the following corollary of this lemma from [3]:

▶ **Lemma 2.1** ([18]). *For every non-constant n-variate boolean function f, there exists an input $x \in f^{-1}(1)$ with $\mathsf{sens}(f,x) \geq n - \log|f^{-1}(1)|$.*

Although decision tree complexity of a boolean function is polynomially related to many other measures that we do not define here (see, for example, [7, 9]), its relationship with the sensitivity remains elusive. The question of whether there is a polynomial relation between $\mathsf{sens}(f)$ and the decision tree complexity $Dt(f)$, known as the Sensitivity Conjecture, has been formulated already in [15]. However, despite much work, it is still unresolved.

▶ **Conjecture 2.2** (Sensitivity conjecture). There exists an integer k such that, for any function f, $Rt(f) \leq \mathsf{sens}(f)^k$.

3 Black-Box Hypothesis

3.1 Defining BBH

To investigate whether having a circuit C_f for an input function f helps decide a property F of boolean functions, we compare the complexity of deciding F on f given a circuit C_f versus given an oracle access to f. In the latter case, following [5], an algorithm deciding $F(f)$ is also given as its input the size m of some C_f (or, rather, an upper bound on C_f), in unary (that is, the algorithm can "see how large the box is", but cannot peek inside). This makes the comparison of the running time in both frameworks more meaningful. With this intuition, we define "white-box" and "black-box" algorithms as follows.

▶ **Definition 3.1** (White-box vs. black-box algorithms). An algorithm A decides a property F in *white-box* if A decides the corresponding meta-language L_F. That is, given as input a string $desc(C)$ A accepts iff $[C] \in F$.

[1] For semantic complexity classes such as RTIME, it is customary to use the weaker notion of a class with advice, where the algorithm is required to behave as a true RTIME-type algorithm only when given a correct advice string, and can behave arbitrarily otherwise.

An algorithm A decides F in *black-box* if $A^f(1^n, 1^m)$ accepts iff $f \in F$, where $f\colon \{0,1\}^n \to \{0,1\}$, m is an upper bound on the circuit size of f and A^f denotes that the algorithm A has oracle access to the boolean function f; as usual, 1^n and 1^m represent n and m in unary.

▶ **Definition 3.2.** A property F is in *white-box* BPP, denoted $F \in$ wbBPP, if there is a BPP algorithm deciding L_F. We say F is in *black-box* BPP, denoted $F \in$ bbBPP, if there is a black-box randomized algorithm $A^f(1^n, 1^m)$ deciding F in time polynomial in $n + m$, with the probability of error at most $1/3$ over the choice of randomness, for every f, n, m.

With the above definitions, the Black-Box Hypothesis can be stated concisely as follows.

▶ **Hypothesis 3.3** (Black-Box Hypothesis (BBH))**.** *For any property F of boolean functions,*

$$F \in \text{wbBPP} \iff F \in \text{bbBPP}.$$

If the BBH holds, then P $\neq$ NP, as the well-known exponential black-box lower bounds for SAT would rule out even a subexponential-time probabilistic algorithm for SAT. On the other hand, if NP $\subseteq$ BPP, then the BBH is false, with SAT as a counterexample. Suppose the BBH is false. Would that imply that SAT is easy? We make the following conjecture.

▶ **Conjecture 3.4.** (Informal) BBH is false iff Circuit-SAT is easy.

As a step towards proving the conjecture, we show that if the BBH fails in a particular way (see the next subsection for the definition), then there is a family of circuits of subexponential size that decides Circuit-SAT.

3.2 Defining a Strong Counter-Example to BBH

As noted before, a property $F \in$ wbBPP can only be a counterexample to BBH when any black-box algorithm requires superpolynomial time on some input of subexponential size (otherwise white-box complexity and black-box complexity are polynomially related).

Thus, if F is not in black-box BPP, then any black-box algorithm deciding F requires superpolynomial time on some input of subexponential circuit size, which we call an easy input.

Ideally, we would like to prove that if the BBH fails, then Circuit-SATis easy. We do not know how to show such an implication yet. Instead, we consider the following *sufficient* condition for the BBH to fail.

▶ **Definition 3.5** (Strong counterexample to the BBH)**.** A property F is an *s-strong counterexample to the BBH* if

1. F is in wbBPP, but
2. for almost all n, F has an input $f^* : \{0,1\}^n \to \{0,1\}$ of *size*$(f^*) \leq 2^{o(n)}$ such that sens$(F, f^*) \geq s$.

We call a property a *strong counterexample* if it is $2^{\Omega(n)}$-strong.

Next we argue that a strong counterexample to the BBH as defined above would indeed violate the BBH. First, we recall the following result.

▶ **Lemma 3.6** (implicit in [15])**.** *Let F be a property of n-variate boolean functions. If* sens$(F, f) \geq s$ *for some boolean function $f \in$* SIZE(t)*, then $Rt_{(t+cn)} \geq (2/3)s$ (for some constant $c > 0$).*

Proof. Let f^i be the function that disagrees with f on the ith bit of the output, which is a sensitive bit of f. Thus, (the truth tables of) f and f^i are Hamming neighbours and circuit complexity of f^i is greater than f by at most a linear factor, i.e., $size(f^i) \leq size(f) + O(n)$. Now to distinguish f from each Hamming neighbour f^i with probability at least $2/3$, any randomized decision tree needs to query the ith bit with probability at least $2/3$. As there are s many sensitive bits for f, the expected number of queries is $(2/3)s$. Thus, there is one branch on which the randomized decision tree has to query $(2/3)s$ of the bits. ◀

Applying Theorem 3.6 immediately yields the required implication.

▶ **Corollary 3.7.** *If F is a $n^{\omega(1)}$-strong counterexample to the BBH, then $F \notin$ bbBPP (and hence, the BBH is false).*

3.3 Examples of properties with easy sensitive inputs

We give a few examples of properties with easy sensitive inputs. For each of these properties, violating the BBH is actually *equivalent* to being a strong counterexample to the BBH.

Symmetric properties. A property F is symmetric if the membership of $tt(f) \in F$ depends only on the number of 1s in $tt(f)$. Such properties were the focus of one of the previous formulations of a possible complexity analogue of Rice's theorem, due to Borchert and Stephan [6] (though their notion of hardness was somewhat different). A basic symmetric property of N-bit strings such as OR or XOR has an easy input (the all-0 string) of sensitivity N. We note that every symmetric property has an easy input of sensitivity at least $N/2$.

▶ **Lemma 3.8.** *If F is a non-trivial symmetric property of n-variate boolean functions, then there is a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ with $\mathsf{sens}(F, f) \geq 2^n/2$ such that f is computable by an AC^0 circuit of polynomial size.*

Proof. As F is a non-trivial property, there is a number $1 \leq k \leq 2^n$ such that a $tt(f)$ with $k-1$ ones is accepted by F (wlog), but any $tt(f)$ with k ones is rejected by F. If $k \geq 2^n/2$, then any string with k ones has sensitivity k. Otherwise, any string with $k-1$ ones has sensitivity $2^n - (k-1) \geq 2^n/2$.

Let k be the number of 1s in an input with sensitivity at least $2^n/2$. Define a required boolean function f with exactly k ones in its truth table by $f(x) = 1$ iff $x < k$, where x is interpreted as an integer in binary. It is easy to see that f has a polynomial-size circuit, even of AC^0 type (as the comparison of two n-bit integers can be implemented in AC^0 [8]). ◀

Subsets of easy functions. Consider a property F that only contains a subset of easy functions, that is, only functions of circuit complexity at most $t = 2^{o(n)}$. Easy functions form a very sparse set (the number of n-bit functions of circuit size at most t is at most 2^{t^2}). So by Simon's lemma (Theorem 2.1), F contains an (easy) instance of sensitivity at least $2^n - t^2 = 2^n - 2^{o(n)} = \Omega(2^n)$.

4 Circuit-SAT algorithm from strong counterexamples

The main theorem of this section shows that a strong counterexample to the BBH (as in Theorem 3.5) implies that Circuit-SAT on n-input circuits of subexponential size can be decided by subexponential-size circuits. Formally, we have the following.

▶ **Theorem 4.1.** *If there is a strong counterexample to the BBH, then*

$\mathsf{Circuit\text{-}SAT}_{n,2^{o(n)}} \in \mathsf{SIZE}(2^{o(n)})$.

We prove this theorem in two steps. First we show (in Section 4.1) how sensitivity can be exploited for deriving a randomized algorithm for satisfiability, whose success probability depends on the assumed sensitivity of a given counterexample to the BBH. Then (in Section 4.2) we amplify the success probability of our algorithm.

4.1 From high sensitivity to Circuit-SAT

Here we prove the following.

▶ **Lemma 4.2.** *Let F be an s-strong counterexample to the BBH, with an s-sensitive function family $f \in \mathsf{SIZE}(t)$. Then $\mathsf{Circuit\text{-}SAT}_{n,m}$ is decidable in randomized time $\mathsf{poly}(t,m)$, with success probability $\Omega(s/2^n)$, given the advice of size $\mathsf{poly}(t)$. In particular, we have that*

$\mathsf{Circuit\text{-}SAT}_{n,m} \in \mathsf{SIZE}(\mathsf{poly}(n \cdot (t(n)+m) \cdot 2^n/s(n)))$.

Proof. Let A_F be a BPP algorithm for L_F. By Adleman's argument [1], we can assume that A_F is a deterministic algorithm, using at most $\mathsf{poly}(m)$ bits of advice on inputs of length m.

As a warm-up, suppose that F has maximal sensitivity 2^n, and, moreover, for each n there is a maximally sensitive input $tt(f)$ where f has a circuit C_f of size t. Now, if C has at most 1 satisfying assignment, it is enough to check whether $A_F(C \oplus C_f) = A_F(C_f)$: if there is a satisfying assignment for C, it flips a sensitive bit of $tt(C_f)$, otherwise $tt(C \oplus C_f) = tt(C_f)$.

To use the idea described above we need to guarantee that the circuit C for which we want to decide satisfiability has at most one satisfiable assignment. This can be done by applying the Valiant-Vazirani reduction [19] to get new circuit C'. Assuming that f is a highly sensitive input, we have a non-trivial chance of hitting one of its sensitive bits if we randomly shift a unique satisfying assignment of C'. That is, we check $A_F(C'(x \oplus r) \oplus C_f)$, where r is a random binary string of length $|x|$. More formally, our algorithm for Circuit-SAT is as follows.

Algorithm for Circuit-SAT
Input: A circuit C on n inputs.
Advice: A circuit C_f of size at most t such that $tt(C_f)$ is an s-sensitive string for F.

1. Apply the Valiant-Vazirani reduction to C to obtain a list $C_1, \ldots, C_n$ satisfying the following: if C is unsatisfiable then so is every C_i on the list, and if C is satisfiable, then, with probability at least $1/2$, at least one C_i on the list has a unique satisfying assignment.
2. Pick a random $r \in \{0,1\}^n$. For each C_i on the list, check if

   $$A_F(C_f) \neq A_F(C_i(x \oplus r) \oplus C_f).$$

   If the check passes for at least one $1 \leq i \leq n$, then accept; otherwise, reject.

The running time of the described algorithm is $\mathsf{poly}(n, t+m)$. The advice size is $\mathsf{poly}(t)$, as we need C_f, plus the advice of size $\mathsf{poly}(|C|+|C_f|)$ used in Adleman's averaging argument. If C is unsatisfiable, then the algorithm rejects C with probability 1. If C is satisfiable, then the algorithm accepts with probability at least $(1/2) \cdot s/2^n$ (the success probability of the Valiant-Vazirani reduction in Step (1), times the probability of hitting a sensitive bit of the advice $tt(C_f)$ by a random shift r in Step (2)).

Finally, applying Adleman's argument to the randomized algorithm above, we get a nonuniform circuit family solving Circuit-SAT with the stated parameters. ◀

▶ **Corollary 4.3.** *Let F be a non-trivial symmetric property such that $L_F \in$ BPP. Then* Circuit-SAT $\in$ RP/poly $\subseteq$ P/poly.

Proof. The proof follows from Theorem 3.8 and Theorem 4.2. ◀

4.2 Amplifying the success probability

Theorem 4.2 is a weaker version of Theorem 4.1 which needs the sensitivity bound $s \geq 2^{n-o(n)}$. To handle a smaller sensitivity $2^{\delta n}$, for any $\delta > 0$, we need a better way of amplifying the success probability of our randomized Circuit-SAT algorithm above, without increasing the circuit size by too much. We will use the following Exponential Amplification lemma by Paturi and Pudlák [17].

▶ **Lemma 4.4** (Exponential amplification lemma[17])**.** *Let $\mathcal{G}$ be a family of probabilistic circuits of size bounded by $g(m,n)$ such that $\mathcal{G}$ decides* Circuit-SAT *with one-sided error, achieving the success probability $2^{-\delta n}$ on satisfiable instances. Then there exist a circuit family $\mathcal{G}'$ deciding* Circuit-SAT *with success probability $2^{-\delta^2 n}$ on satisfiable instances, for all large enough n, where the circuit size of $\mathcal{G}'$ is bounded by $g'(n,m) = O(g(\lceil \delta n \rceil) + 5, \tilde{O}(g(n,m))))$.*

Now we can prove Theorem 4.1.

Proof of Theorem 4.1. Let $G^0_{m,n}$ be the circuit family encoding the randomized algorithm from Theorem 4.2. For concreteness, let $desc(C_f) = 2^{n^\gamma}$ denote a bound on the size of $|C_f|$. The size of the complete circuit $G^0_{m,n}$ is $O(2^{kn^\gamma} \cdot n^{k\gamma+1} \cdot m^k)$, where k is the exponent of the running time of A_F. Assuming that $m \leq |C_f|$ to bound smaller factors, $|desc(G^0_{m,n})| = O(2^{kn^\gamma} \cdot n^{(k+1)\gamma+1} \cdot m^k)$.

Apply the Exponential amplification lemma for t iteration to $G^0_{m,n}$, where $t \in \omega(1)$ is a very slow growing function. If $2^{o(n)} = 2^{\alpha(n)}$ is the bound on the advice circuit $|C_f|$, then we need $k^t \cdot \alpha(n) < \beta(n)$, where $\beta(n) \in o(n)$. As t is non-constant, success probability becomes $2^{\delta^t n} \in 2^{o(n)}$. Now, using the standard techniques to amplify the success probability (with $2^{\delta^t n} + O(n)$ trials and fixing randomness by the averaging argument), we obtain a deterministic circuit of subexponential size solving Circuit-SAT for circuits of description size m on n variables. ◀

5 Monotone properties

Here we consider a special case of monotone properties F. First, we argue that it suffices to have a monotone counterexample to the BBH with just superpolynomial sensitivity in order to obtain a non-trivial Circuit-SAT algorithm (Section 5.1). Then we show that having a monotone property F in white box P such that F requires high decision tree complexity implies either a non-trivial Circuit-SAT algorithm or non-trivial derandomization of BPP (Section 5.2).

5.1 Handling a lower sensitivity bound

So far, to get a non-trivial Circuit-SAT algorithm from a counterexample F to the BBH, we assumed that we have an easy sensitive input $f^*\colon \{0,1\}^n \to \{0,1\}$ with $\mathsf{sens}(F, f^*) \geq 2^{\Omega(n)}$. Here we show that for a special case of *monotone* properties F, any superpolynomial sensitivity $s \in n^{\omega(1)}$ would suffice to get the same kind of Circuit-SAT algorithms.

▶ **Theorem 5.1.** *Let F be a monotone property such that*

1. *F is decidable in* BPP *in the white-box setting, but,*
2. *for almost all n, F has an input $f^*\colon \{0,1\}^n \to \{0,1\}$ of sensitivity $s \geq n^{\omega(1)}$ and of circuit complexity $s^{o(1)} \geq \mathsf{poly}(n)$ (which implies that F requires superpolynomial time to decide in the black-box complexity setting, even on functions of circuit complexity $s^{o(1)}$).*

Then $\mathsf{Circuit\text{-}SAT}_{n,2^{o(n)}} \in \mathsf{SIZE}(2^{o(n)})$.

Proof. Without loss of generality, assume that $F(f^*) = 1$. Given f^* as advice, we describe a Circuit-SAT algorithm for circuits on $k = \log_2 s$ inputs. We will use random hash functions. Recall that a universal hash family $\mathcal{H}_{n,k} = \{h\colon \{0,1\}^n \to \{0,1\}^k\}$ has the properties: (1) for every fixed $x \in \{0,1\}^n$, the value $h(x)$, for a random $h \in \mathcal{H}_{n,k}$, is uniform over $\{0,1\}^k$, and (2) for every $x \neq y \in \{0,1\}^n$, the values $h(x)$ and $h(y)$, for a random $h \in \mathcal{H}_{n,k}$, are independent and uniform over $\{0,1\}^k$. Our Circuit-SAT algorithm is as follows:

Given a Circuit-SAT instance C on k inputs of size $2^{o(k)}$,

1. pick a random hash function $h\colon \{0,1\}^n \to \{0,1\}^k$ from the universal hash family $\mathcal{H}_{n,k}$, and build a circuit for the following function f': for every $x \in \{0,1\}^n$, set

$$f'(x) = \begin{cases} f^*(x) & \text{if } f^*(x) = 0 \\ f^*(x) \oplus C(h(x)) & \text{otherwise} \end{cases}$$

2. Run the white-box BPP algorithm to decide $F(f')$. If $F(f') = 0$, output "C is satisfiable"; otherwise, output "C is unsatisfiable".

For the time analysis, note that the circuit size for f' defined above is $O(s^{o(1)}) + \mathsf{poly}(n) \leq O(s^{o(1)})$, as $f'(x) = f^*(x) \wedge \neg C(h(x))$, and h has a circuit of size $\mathsf{poly}(n)$.

Thus, the described algorithm runs in time $\mathsf{poly}(s^{o(1)}) \leq s^{o(1)}$, which is $2^{o(k)}$ for k-input Circuit-SAT instances C.

For correctness, note that if C is unsatisfiable, then $f' = f^*$, and so $F(f') = 1$. If C is satisfiable, say by an assignment $y \in \{0,1\}^k$, then, with probability at least $1/2$ over the choice of h, the set $h^{-1}(y)$ will contain at least one sensitive location $x \in \{0,1\}^n$ such that $f^*(x) = 1$, but flipping f^* at x results in the new function g such that $F(g) = 0$.

By monotonicity of F, flipping f^* at x *and at any other locations x' where $f(x') = 1$* results in a new function f' such that $F(f') = 0$. ◀

5.2 Win-win analysis

As the Sensitivity Conjecture is true for monotone properties, assuming that a monotone property F requires high decision tree complexity (i.e., non-uniform black-box complexity) implies that F has a (not necessarily easy) sensitive input. We use a "win-win" argument to prove the following.

▶ **Theorem 5.2.** *Let F be any monotone property such that*

1. *F is in* P *in the white-box setting, but,*
2. *for almost all input lengths n, F requires decision tree complexity at least $s > n^{\omega(1)}$ on inputs $f\colon \{0,1\}^n \to \{0,1\}$.*

Then either $\mathsf{Circuit\text{-}SAT}_{n,2^{o(n)}} \in \mathsf{SIZE}(2^{o(n)})$ *infinitely often, or* $\mathsf{BPP} \subseteq \mathsf{NP}$.

6 Circuit-SAT algorithm from variants of MCSP

So far we have considered a Circuit-SAT algorithm that relies on the sensitivity of a given counterexample F to the BBH. In this section we will show a different approach to designing Circuit-SAT algorithm from properties that are subsets of easy functions, the one that does not explicitly use the notion of sensitivity.[2]

We consider the following *succinct* version of MCSP, denoted SuccinctMCSP, where one is given a circuit as input, and is asked to determine if there is a smaller circuit computing the same boolean function; see, e.g., [2] for a recent use and some basic results about SuccinctMCSP. More formally, for $t = t(n)$, $\mathsf{SuccinctMCSP}_t(C)$ asks to decide if $f = [C]$ is in $\mathsf{SIZE}(t)$.

▶ **Theorem 6.1.** *For any efficiently computable $t(n) \in \omega(n)$, if $\mathsf{SuccinctMCSP}_t \in \mathsf{BPP}$, then*

$$\mathsf{Circuit\text{-}SAT}_{n,m} \in \mathsf{RTIME}(\mathsf{poly}(t(n), m)).$$

▶ **Theorem 6.2.** *Let F be a non-empty (for all n) property that contains only a subset of functions $f \in \mathsf{SIZE}(t(n))$, for some efficiently computable $t(n) \in \omega(n)$. If $F \in \mathsf{wbBPP}$, then*

$$\mathsf{Circuit\text{-}SAT}_{n,m} \in \mathsf{RTIME}(\mathsf{poly}(t(n), m))/t(n).$$

7 BBH for restricted circuit classes

We formulated the BBH with general circuits as inputs to the white-box algorithm. It is natural to consider its variants with other types of circuits. We observe that for a very weak type of circuits, e.g., read-once branching programs, the corresponding version of the BBH is unconditionally false. For AC^0 circuits, we show that a strong counterexample to this version of the BBH implies a non-trivial Circuit-SAT algorithm for AC^0 circuits. The case of CNF formulas remains an interesting open question.

8 Conclusions

We conjecture that the falsehood of the BBH is equivalent to the easiness of Circuit-SAT. In the present paper, we make a step in that direction, but many interesting questions remain open. Below we list a few of them.

1. Is it possible to prove our conjecture, assuming the Sensitivity Conjecture is true?
2. Is it possible to get a *uniform* algorithm for Circuit-SAT for a general class of counterexamples to BBH, thereby (conditionally) violating the ETH?
3. Are there any algorithmic SAT consequences from the assumption that there is a strong counterexample to the BBH for *CNF formulas* (rather than AC^0 or general circuits)?
4. The initial formulation of BBH by Barak et al. [5] was mainly inspired by the idea of virtual black-box obfuscation. Is it possible to use indistinguishability obfuscators for proving or disproving BBH?

Acknowledgements. We are very grateful to Marco Carmosino, Shachar Lovett and Avi Wigderson for many insightful discussions. We also thank Rahul Santhanam for his comments and suggestions on the earlier version of this work.

[2] Of course, as noted earlier, Simon's lemma implies that any such property does have an easy sensitive input, and so one can use the sensitivity-based Circuit-SAT algorithm described above. The point here, however, is to have a different type of a Circuit-SAT algorithm.

References

1 Leonard Adleman. Two theorems on random polynomial time. In *Proceedings of the Nineteenth Annual IEEE Symposium on Foundations of Computer Science*, pages 75–83, 1978.

2 Eric Allender, Dhiraj Holden, and Valentine Kabanets. The minimum oracle circuit size problem. *Computational Complexity*, 26(2):469–496, 2017. doi:10.1007/s00037-016-0124-0.

3 Andris Ambainis and Jevgēnijs Vihrovs. Size of sets with small sensitivity: A generalization of Simon's lemma. In *International Conference on Theory and Applications of Models of Computation*, pages 122–133. Springer International Publishing, 2015.

4 Laci Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

5 Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. *Journal of the ACM (JACM)*, 59(2):6, 2012.

6 Bernd Borchert and Frank Stephan. Looking for an analogue of Rice's theorem in circuit complexity theory. *Math. Log. Q.*, 46(4):489–504, 2000. doi:10.1002/1521-3870(200010)46:4<489::AID-MALQ489>3.0.CO;2-F.

7 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, Oct 2002.

8 Ashok K Chandra, Larry Stockmeyer, and Uzi Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, 1984.

9 Pooya Hatami, Raghav Kulkarni, and Denis Pankratov. Variations on the sensitivity conjecture. *Theory of Computing, Graduate Surveys*, 2:1–27, 2011.

10 Lane A. Hemaspaandra and Jörg Rothe. A second step towards complexity-theoretic analogs of Rice's theorem. *Theor. Comput. Sci.*, 244(1-2):205–217, 2000.

11 Lane A. Hemaspaandra and Mayur Thakur. Lower bounds and the hardness of counting properties. *Theor. Comput. Sci.*, 326(1-3):1–28, 2004.

12 Russell Impagliazzo, Valentine Kabanets, Antonina Kolokolova, Pierre McKenzie, and Shadab Romani. Does looking inside a circuit help? *Electronic Colloquium on Computational Complexity*, 17(109), 2017.

13 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

14 Russell Impagliazzo and Avi Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, 1997.

15 Noam Nisan. CREW PRAMs and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991.

16 Noam Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

17 Ramamohan Paturi and Pavel Pudlák. On the complexity of circuit satisfiability. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 241–250, 2010.

18 Hans-Ulrich Simon. A tight ω (loglog n)-bound on the time for parallel RAM's to compute nondegenerated boolean functions. In *Foundations of Computation Theory*, pages 439–444. Springer, 1983.

19 Leslie Valiant and Vijay Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

The Power of Programs over Monoids in DA

Nathan Grosshans[*1], **Pierre McKenzie**[2], **and Luc Segoufin**[3]

1 **LSV, CNRS, ENS Paris-Saclay, Cachan, France, and DIRO, Université de Montréal, Montréal, Canada**
nathan.grosshans@polytechnique.edu
2 **DIRO, Université de Montréal, Montréal, Canada**
mckenzie@iro.umontreal.ca
3 **INRIA and LSV, ENS Paris-Saclay, Cachan, France**
luc.segoufin@inria.fr

Abstract

The program-over-monoid model of computation originates with Barrington's proof that it captures the complexity class NC^1. Here we make progress in understanding the subtleties of the model. First, we identify a new tameness condition on a class of monoids that entails a natural characterization of the regular languages recognizable by programs over monoids from the class. Second, we prove that the class known as **DA** satisfies tameness and hence that the regular languages recognized by programs over monoids in **DA** are precisely those recognizable in the classical sense by morphisms from **QDA**. Third, we show by contrast that the well studied class of monoids called **J** is not tame and we exhibit a regular language, recognized by a program over a monoid from **J**, yet not recognizable classically by morphisms from the class **QJ**. Finally, we exhibit a program-length-based hierarchy within the class of languages recognized by programs over monoids from **DA**.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Programs over monoids, DA, lower-bounds

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.2

1 Introduction

A program of range n on alphabet Σ over a finite monoid M is a sequence of pairs (i, f) where i is a number between 1 and n and f is a function assigning an element of the monoid M to any letter of Σ. This program assigns to each word $w_1 w_2 \cdots w_n$ the monoid element obtained by multiplying out in M the elements $f(w_i)$, one per pair (i, f), in the order of the sequence. When associated with a subset F of M as an acceptance set, a program naturally defines the language L_n of words of length n to which it assigns a monoid element in F. A program sequence $(P_n)_{n\in\mathbb{N}}$ then defines the language formed by the union of the L_n.

Such sequences became the focus of attention when Barrington [3] made the striking discovery, in fact partly observed earlier [17], that polynomial length program sequences over the group S_5 and sequences of Boolean circuits of polynomial size, logarithmic depth and constant fan-in (defining the complexity class NC^1) recognize precisely the same languages.

A flurry of work followed. After all, a program over M is a mere generalization of a morphism from Σ^* to M and recognition by a morphism equates with acceptance by a finite automaton. Given the extensive algebraic automata theory available at the time [15, 11, 22],

* This work is supported by grants from Digiteo France.

it was to be a matter of a few years before the structure of NC^1 got elucidated by algebraic means.

The "optimism period" produced many significant results. The classes $\mathsf{AC}^0 \subset \mathsf{ACC}^0 \subseteq \mathsf{NC}^1$ were characterized by polynomial length programs over the aperiodic, the solvable, and all monoids respectively [3, 6]. More generally for any variety $\mathbf{V}$ of monoids (a variety being the undisputed best fit with the informal notion of a natural class of monoids) one can define the class $\mathcal{P}(\mathbf{V})$ of languages recognized by polynomial length programs over a monoid drawn from $\mathbf{V}$. In particular, if $\mathbf{A}$ is the variety of aperiodic monoids, then $\mathcal{P}(\mathbf{A})$ characterizes the complexity class AC^0 [6].

But sadly, the optimism period ended: although partial results in restricted settings were obtained, the holy grail of reproving significant circuit complexity results and forging ahead by recycling the deep theorems afforded by algebraic automata theory never materialized. The test case for the approach was to try to prove, independently from the known combinatorial arguments [1, 12, 14] and those based on approximating circuits by polynomials over some finite field [24, 27], that $\mathcal{P}(\mathbf{A})$ does not contain the parity language MOD_2, i.e., that $\mathrm{MOD}_2 \notin \mathsf{AC}^0$. But why has this failed?

The answer of course is that programs are much more complicated than morphisms: programs can read an input position more than once, in non-left-to-right order, possibly assigning a different monoid element each time. Linear-length programs can indeed trivially recognize non-regular languages. In the classical theory, any two varieties provably recognize distinct classes of languages [11, 22]. In the theory of recognition by polynomial length programs (we will speak then of p-recognition), distinct varieties can yield the same class, as do, for instance, any two varieties of monoids $\mathbf{V}$ and $\mathbf{W}$ that each contain a simple non-Abelian group, for which $\mathcal{P}(\mathbf{V}) = \mathcal{P}(\mathbf{W}) = \mathsf{NC}^1$ [18, Theorem 4.1].

To further illustrate the subtle behavior of programs, consider the variety of monoids known as $\mathbf{J}$. $\mathbf{J}$ is the variety of monoids generated by the syntactic monoids of the languages such that membership can be decided by looking for the presence or absence of certain *subwords*, where u is a subword of v if u can be obtained from v by removing some letters of v [26]. It follows that $\mathbf{J}$ is unable to recognize the language defined by the regular expression $(a+b)^*ac^+$. Yet $(a+b)^*ac^+$ is p-recognizable over $\mathbf{J}$. To see this consider the language L of all words having ca as a subword but not the subwords cca, caa and cb. L is therefore recognized by a morphism φ to some monoid M of $\mathbf{J}$, i.e. $L = \varphi^{-1}(F)$ for some $F \subseteq M$. A program of range n over M can recognize the words of length n of $(a+b)^*ac^+$ by using the following trick: reading the input letters in the order $2, 1, 3, 2, 4, 3, 5, 4, \ldots, n, n-1$, assigning to each letter read its image in M by φ and using the same acceptance set F as for L. For instance, on input $abacc$ the program outputs $\varphi(baabcacc)$ which is in F, while on inputs $abbcc$ and $abacca$ the program outputs respectively $\varphi(babbcbcc)$ and $\varphi(baabcaccac)$ which are not in F.

Our paper is motivated by the need to better understand such subtle behaviors of polynomial length programs over monoids. Quite a bit of knowledge on such programs has accumulated over nearly thirty years (consider [5, 20, 21, 16, 28] beyond the references already mentioned). Yet, even within the realm of questions that do not hold pretense to major complexity class separations, gaps remain.

One beaming such gap concerns the variety of monoids $\mathbf{DA}$. The importance of $\mathbf{DA}$ in algebraic automata theory and its connections with other fields are well established (see [30] for an eloquent testimony). $\mathbf{DA}$ is a relatively "small" variety, well within the variety of aperiodic monoids. One could argue that "small" varieties will be sensitive to duplications and rearrangements in the order in which input letters are read by a program. Presumably in part for that reason, programs over $\mathbf{DA}$ have seemingly not been successfully analyzed prior to our work.

Our main result is a characterization of the regular languages recognized by polynomial length programs over **DA**. We show that $\mathcal{P}(\mathbf{DA}) \cap \mathcal{R}\mathsf{eg}$ is precisely the class $\mathcal{L}(\mathbf{QDA})$ of languages recognized classically by morphisms in quasi-**DA** (**QDA**). A surjective morphism φ from Σ^* to a finite monoid M is quasi-**DA** if, though M might not be in **DA**, there is a number k such that the image by φ of all words over Σ whose length is a multiple of k forms a submonoid of M which is in **DA**. In this particular case, this statement is equivalent to the fact that the only power added by p-recognition relative to classical recognition through monoids in **DA** is the ability to count input positions modulo a constant. Our proof shows, independently from [1, 12, 14, 24, 27], that, for regular languages, p-recognition over **DA** does not distort the algebraic properties of the underlying morphisms beyond adding the ability for fixed modulo counting on lengths. (This is precisely the statement, once extended to the variety of all aperiodic monoids, that would yield the elusive semigroup-theoretic proof that $\mathrm{MOD}_2 \notin \mathsf{AC}^0$.)

Our main result builds upon a statement of independent interest, namely, that any variety of monoids **V** that fulfills an appropriate tameness condition satisfies $\mathcal{P}(\mathbf{V}) \cap \mathcal{R}\mathsf{eg} \subseteq \mathcal{L}(\mathbf{QV})$. The new tameness condition (see Definition 2) differs subtly but fundamentally from a similar notion developed for semigroups by Péladeau, Straubing and Thérien [21] and also studied in the case of monoids by Péladeau [20] and later Tesson [28] in their respective Ph.D. theses, and thus requires a separate treatment here. Proving that **DA** indeed satisfies our tameness condition is the main technical difficulty behind our characterization of $\mathcal{P}(\mathbf{DA}) \cap \mathcal{R}\mathsf{eg}$.

We further consider $\mathcal{P}(\mathbf{DA})$. With $\mathcal{C}_k$ the class of languages recognized by programs of length $\mathrm{O}(n^k)$ over **DA**, we prove that $\mathcal{C}_1 \subset \mathcal{C}_2 \subset \cdots \subset \mathcal{C}_k \subset \cdots \subset \mathcal{P}(\mathbf{DA})$ forms a strict hierarchy. We also relate this hierarchy to another algebraic characterization of **DA** and exhibit conditions on $M \in \mathbf{DA}$ under which any program over M can be rewritten as an equivalent subprogram (made of a subsequence of the original sequence of instructions) of length $\mathrm{O}(n^k)$, refining a result by Tesson and Thérien [29].

Our final result concerns the variety **J**. Observing that the regular language $(a+b)^*ac^+$ mentioned above is not recognizable by a morphism in **QJ**, we conclude that **J** is not a tame variety. Be it the chicken or the egg, this lack of tameness "explains" the unexpected power of $\mathcal{P}(\mathbf{J})$ witnessed in our example above. Furthermore, since **J** is a p-variety of monoids in the sense of [21, 20, 28], **J** explicitly shows that our notion of tameness and that of [21, 20, 28] differ.

Organization of the paper. In Section 2 we define programs over varieties of monoids, p-recognition by such programs and the necessary algebraic background. The definition of tameness for a variety **V** is given in Section 3 with our first result showing that regular languages recognized by $\mathcal{P}(\mathbf{V})$ are included in $\mathcal{L}(\mathbf{QV})$ when **V** is tame; we also quickly discuss the case of **J**, which isn't tame. We show that **DA** is tame in Section 4. Finally, Section 5 contains the hierarchy results about $\mathcal{P}(\mathbf{DA})$.

2 Preliminaries

This section is dedicated to introducing the mathematical material used throughout this paper. Concerning algebraic automata theory, we only quickly review the basics and refer the reader to the two classical references of the domain by Eilenberg [10, 11] and Pin [22].

General notations. Let $i, j \in \mathbb{N}$ be two natural numbers. We shall denote by $[\![i, j]\!]$ the set of all natural numbers $n \in \mathbb{N}$ verifying $i \leq n \leq j$. We shall also denote by $[i]$ the set $[\![1, i]\!]$.

Words and languages. Let Σ be a finite alphabet. We denote by Σ^* the set of all finite words over Σ. We also denote by Σ^+ the set of all finite non empty words over Σ, the empty word being denoted by ε. A *language over* Σ is a subset of Σ^*. A language is *regular* if it can be defined using a regular expression. Given a language L, its *syntactic congruence* $\sim_L$ is the relation on Σ^* relating two words u and v whenever for all $x, y \in \Sigma^*$, $xuy \in L$ if and only if $xvy \in L$. It is easy to check that $\sim_L$ is an equivalence relation and a congruence for concatenation. The *syntactic morphism of* L is the mapping sending any word u to its equivalence class in the syntactic congruence.

The *quotient of a language* L *over* Σ *relative to the words* u *and* v is the language, denoted by $u^{-1}Lv^{-1}$, of the words w such that $uwv \in L$.

Monoids, semigroups and varieties. A *semigroup* is a set equipped with an associative law that we will write multiplicatively. A *monoid* is a semigroup with an identity. An example of a semigroup is Σ^+, the free semigroup over Σ. Similarly Σ^* is the free monoid over Σ. A *morphism* φ *from a semigroup* S *to a semigroup* T is a function from S to T such that $\varphi(xy) = \varphi(x)\varphi(y)$ for all $x, y \in S$. A morphism of monoids additionally requires that the identity is preserved. A semigroup T is a *subsemigroup* of a semigroup S if T is a subset of S and is equipped with the restricted law of S. Additionally the notion of submonoids requires the presence of the identity. The *Cartesian (or direct) product* of two semigroups is simply the semigroup given by the Cartesian product of the two underlying sets equipped with the Cartesian product of their laws.

A language L over Σ is *recognized by a monoid* M if there is a morphism h from Σ^* to M and a subset F of M such that $L = h^{-1}(F)$. We also say that *the morphism* h *recognizes* L. It is well known that a language is regular if and only if it is recognized by a finite monoid. Actually, as $\sim_L$ is a congruence, the quotient $\Sigma^*/\sim_L$ is a monoid, called *the syntactic monoid of* L, that recognizes L via the syntactic morphism of L. The syntactic monoid of L is finite if and only if L is regular. The quotient $\Sigma^+/\sim_L$ is analogously called *the syntactic semigroup of* L.

A *variety of monoids* is a class of finite monoids closed under submonoids, Cartesian product and morphic images. A *variety of semigroups* is defined similarly. When dealing with varieties, we consider only finite monoids and semigroups.

An element s of a semigroup is *idempotent* if $ss = s$. For any finite semigroup S there is a positive number (the minimum such number), *the idempotent number of* S, often denoted ω, such that for any element $s \in S$, s^ω is idempotent.

A variety can be defined by means of identities [25]. The variety is then the class of monoids such that each of them has all its elements satisfy the identities. For instance, the variety of aperiodic monoids **A** can be defined as the class of monoids satisfying the identity $x^\omega = x^{\omega+1}$, where x ranges over the elements of the monoid while ω is the idempotent power of the monoid. The variety of monoids **DA** is defined by the identity $(xy)^\omega = (xy)^\omega x (xy)^\omega$. The variety of monoids **J** is defined by the identity $(xy)^\omega = (xy)^\omega x = y(xy)^\omega$.

Quasi and locally V languages, modular counting and predecessor. If S is a semigroup we denote by S^1 the monoid S if S is already a monoid and $S \cup \{1\}$ otherwise.

The following definitions are taken from [23]. Let φ be a surjective morphism from Σ^* to a finite monoid M. For all k consider the subset $\varphi(\Sigma^k)$ of M. As M is finite there is a k such that $\varphi(\Sigma^{2k}) = \varphi(\Sigma^k)$. This implies that $\varphi(\Sigma^k)$ is a semigroup. The semigroup given by the smallest such k is called the *stable semigroup of* φ. If S is the stable semigroup of φ, S^1 is called *the stable monoid of* φ. If **V** is a variety of monoids, then we shall denote by

$\mathbf{QV}$ the class of such surjective morphisms whose stable monoid is in $\mathbf{V}$ and by $\mathcal{L}(\mathbf{QV})$ the class of languages whose syntactic morphism is in $\mathbf{QV}$.

For $\mathbf{V}$ a variety of monoids, we say that a finite semigroup S is *locally* $\mathbf{V}$ if, for every idempotent e of S, the monoid eSe belongs to $\mathbf{V}$; we denote by $\mathbf{LV}$ the class of locally-$\mathbf{V}$ finite semigroups, which happens to be a variety of semigroups. Finally, $\mathcal{L}(\mathbf{LV})$ denotes the class of languages whose syntactic semigroup is in $\mathbf{LV}$.

We now define languages recognized by $\mathbf{V} * \mathbf{Mod}$ and $\mathbf{V} * \mathbf{D}$. We do not use the standard algebraic definition using the wreath product as we won't need it, but directly a characterization of the languages recognized by such algebraic objects [7, 31].

Let $\mathbf{V}$ be a variety of monoids. We say that a language L over Σ is in $\mathcal{L}(\mathbf{V} * \mathbf{Mod})$ if there is a number $k \in \mathbb{N}_{>0}$ and a language L' over $\Sigma \times \{0, \cdots, k-1\}$ whose syntactic monoid is in $\mathbf{V}$, such that L is the set of words w that belong to L' after adding to each letter of w its position modulo k.

Similarly we say that a language L over Σ is in $\mathcal{L}(\mathbf{V} * \mathbf{D})$ if there is a number $k \in \mathbb{N}$ and a language L' over $\Sigma \times \Sigma^{\leq k}$ (where $\Sigma^{\leq k}$ denotes all words over Σ of length at most k) whose syntactic monoid is in $\mathbf{V}$, such that L is the set of words w that belong to L' after adding to each letter of w the word composed of the k (or less when near the beginning of w) letters preceding that letter. The variety of semigroups $\mathbf{V} * \mathbf{D}$ can then be defined as the one generated by the syntactic semigroups of the languages in $\mathcal{L}(\mathbf{V} * \mathbf{D})$ as defined above.

A variety $\mathbf{V}$ is said to be *local* if $\mathcal{L}(\mathbf{V} * \mathbf{D}) = \mathcal{L}(\mathbf{LV})$. This is not the usual definition of locality, defined using categories, but it is equivalent to it [31, Theorem 17.3]. One of the consequences of locality that we will use is that $\mathcal{L}(\mathbf{V} * \mathbf{Mod}) = \mathcal{L}(\mathbf{QV})$ [9, Corollary 18] (see also [8, 19]).

Programs over varieties of monoids. Programs over monoids form a non-uniform model of computation, first defined by Barrington and Thérien [6], extending Barrington's permutation branching program model [3]. Let M be a finite monoid and Σ a finite alphabet. A program P over M is a finite sequence of instructions of the form (i, f) where i is a positive integer and f a function from Σ to M. The *length* of P is the number of its instructions. A program has *range* n if all its instructions use a number less than n. A program P of range n defines a function from Σ^n, the words of length n, to M as follows. On input $w \in \Sigma^n$, each instruction (i, f) outputs the monoid element $f(w_i)$. A sequence of instructions then yields a sequence of elements of M and their product is the output $P(w)$ of the program.

A language L over Σ is p-*recognized* by a sequence of programs $(P_n)_{n\in\mathbb{N}}$ if for each n, P_n has range n and length polynomial in n and there exists a subset F_n of M such that $L \cap \Sigma^n$ is precisely the set of words w of length n such that $P_n(w) \in F_n$.

We denote by $\mathcal{P}(M)$ the class of languages p-recognized by a sequence of programs $(P_n)_{n\in\mathbb{N}}$ over M. If $\mathbf{V}$ is a variety of monoids we denote by $\mathcal{P}(\mathbf{V})$ the union of all $\mathcal{P}(M)$ for $M \in \mathbf{V}$.

The following is a simple fact about $\mathcal{P}(\mathbf{V})$. Let Σ and Γ be two finite alphabets and $\mu\colon \Sigma^* \to \Gamma^*$ be a morphism. We say that μ is length multiplying, or that μ *is an* lm-*morphism*, if there is a constant k such that for all $a \in \Sigma$, the length of $\mu(a)$ is k.

▶ **Lemma 1** ([18], Corollary 3.5). *Let* $\mathbf{V}$ *be a variety of monoids, then* $\mathcal{P}(\mathbf{V})$ *is closed under Boolean operations, quotients and inverse images of* lm-*morphisms.*

3 General results about regular languages and programs

Let $\mathbf{V}$ be a variety of monoids. By definition any regular language recognized by a monoid in $\mathbf{V}$ is *p*-recognized by a sequence of programs over a monoid in $\mathbf{V}$. Actually, since in a program over some monoid in $\mathbf{V}$, the monoid element output for each instruction can depend on the position of the letter read, hence in particular on its position modulo some fixed number, it is easy to see that any regular language in $\mathcal{L}(\mathbf{V} * \mathbf{Mod})$ is *p*-recognized by a sequence of programs over some monoid in $\mathbf{V}$. In this section we show that for some "well behaved" varieties $\mathbf{V}$ the converse inclusion holds.

For this we introduce the notion of an *sp*-variety of monoids. This notion is inspired by the notion of *p*-varieties (program-varieties) of monoids, that seems to have been originally defined by Péladeau in his Ph.D. thesis [20] and later used by Tesson in his own Ph.D. thesis [28]. The notion of a *p*-variety has also been defined for semigroups by Péladeau, Straubing and Thérien in [21] in order to obtain results similar to ours for varieties of semigroups of the form $\mathbf{V} * \mathbf{D}$.

Let μ be a morphism from Σ^* to a finite monoid M. We denote by $\mathcal{W}(\mu)$ the set of languages L over Σ such that $L = \mu^{-1}(F)$ for some subset F of M. Given a semigroup S there is a unique morphism $\eta_S \colon S^* \to S^1$ extending the identity on S, called the *evaluation morphism of* S. We write $\mathcal{W}(S)$ for $\mathcal{W}(\eta_S)$. We define $\mathcal{W}(M)$ similarly for any monoid M. It is easy to see that if $M \in \mathbf{V}$ then $\mathcal{W}(M) \subseteq \mathcal{P}(\mathbf{V})$. The tameness condition requires a converse of this observation.

▶ **Definition 2.** An sp-*variety of monoids*, which we will call a *tame* variety, is a variety $\mathbf{V}$ of monoids such that for any finite semigroup S, if $\mathcal{W}(S) \subseteq \mathcal{P}(\mathbf{V})$ then $S^1 \in \mathbf{V}$.

The *p*-variety of monoids in [20, 28] is similar to our *sp*-variety but the former only requires that any finite monoid M verifying $\mathcal{W}(M) \subseteq \mathcal{P}(\mathbf{V})$ must in fact belong to $\mathbf{V}$. This implies that any *sp*-variety of monoids is also a *p*-variety of monoids, but the converse is not always true. For instance, $\mathbf{J}$ is a *p*-variety of monoids [28], but Proposition 5 below states that $\mathbf{J}$ is not an *sp*-variety.

An example of an *sp*-variety of monoids is the class of aperiodic monoids $\mathbf{A}$. This is a consequence of the result that for any number $k > 1$, checking if the number of a modulo k in a word is congruent to 0 is not in $\mathsf{AC}^0 = \mathcal{P}(\mathbf{A})$ [12, 1] (a language we shall denote by $\mathrm{MOD_k}$ over the alphabet $\{0, 1\}$). Towards a contradiction, assume there would exist a semigroup S such that S^1 is not aperiodic but still $\mathcal{W}(S) \subseteq \mathcal{P}(\mathbf{A})$. Then there is an x in S such that $x^\omega \neq x^{\omega+1}$. Consider the morphism $\mu \colon \{a, b\}^* \to S^1$ sending a to $x^{\omega+1}$ and b to x^ω, and the language $L = \mu^{-1}(x^\omega)$. It is easy to see that L is the language of all words with a number of a congruent to 0 modulo k, where k is the smallest number such that $x^{\omega+k} = x^\omega$. As $x^\omega \neq x^{\omega+1}$, $k > 1$. From $\mathcal{W}(S) \subseteq \mathcal{P}(\mathbf{A})$ it follows that $\eta_S^{-1}(x^\omega)$ is *p*-recognized by a sequence of programs $(P_n)_{n\in\mathbb{N}}$ over an aperiodic monoid. For each $n \in \mathbb{N}$, we can transform P_n into P'_n where each instruction (i, f) in P_n simply becomes the instruction (i, f') in P'_n with $f'(\sigma) = f(\mu(\sigma))$ for all $\sigma \in \{a, b\}$, so that $P'_n(w) = P_n(\mu(w_1)\mu(w_2)\cdots\mu(w_n))$ for all $w \in \{a, b\}^n$. This entails that the sequence of programs $(P'_n)_{n\in\mathbb{N}}$ *p*-recognizes L, hence L is in $\mathcal{P}(\mathbf{A})$, a contradiction.

The following is the desired consequence of tameness.

▶ **Proposition 3.** *Let* $\mathbf{V}$ *be an* sp-*variety of monoids. Then* $\mathcal{P}(\mathbf{V}) \cap \mathcal{R}\mathsf{eg} \subseteq \mathcal{L}(\mathbf{QV})$.

A similar result was proven for varieties of semigroups of the form $\mathbf{V} * \mathbf{D}$: if $\mathbf{V} * \mathbf{D}$ is a *p*-variety then the regular languages in $\mathcal{P}(\mathbf{V} * \mathbf{D})$ are exactly those in $\mathcal{L}(\mathbf{V} * \mathbf{D} * \mathbf{Mod})$ [21] (programs over semigroups being defined in the obvious way). Our proof follows the same lines.

Proof. Let L be a regular language in $\mathcal{P}(M)$ for some $M \in \mathbf{V}$. Let M_L be the syntactic monoid of L and η_L its syntactic morphism. Let S be the stable semigroup of η_L, in particular $S = \eta_L(\Sigma^k)$ for some k. We wish to show that S^1 is in $\mathbf{V}$.

We show that $\mathcal{W}(S) \subseteq \mathcal{P}(\mathbf{V})$ and conclude from the fact that $\mathbf{V}$ is an *sp*-variety that $S^1 \in \mathbf{V}$ as desired. Let $\eta_S \colon S^* \to S^1$ be the evaluation morphism of S. Consider $m \in S$ and consider $L' = \eta_S^{-1}(m)$. We wish to show that $L' \in \mathcal{P}(\mathbf{V})$. This implies that $\mathcal{W}(S) \subseteq \mathcal{P}(\mathbf{V})$ by closure under union, Lemma 1.

Let $L'' = \eta_L^{-1}(m)$. Since m belongs to the syntactic monoid of L and η_L is the syntactic morphism of L, a classical algebraic argument [22, Chapter 2, proof of Lemma 2.6] shows that L'' is a Boolean combination of quotients of L or their complements. By Lemma 1, we conclude that $L'' \in \mathcal{P}(\mathbf{V})$.

By definition of S, for any element s of S there is a word u_s of length k such that $\eta_L(u_s) = s$. Notice that this is precisely where we need to work with S and not S^1.

Let $f \colon S^* \to \Sigma^*$ be the *lm*-morphism sending s to u_s and notice that $L' = f^{-1}(L'')$. The result follows by closure of $\mathcal{P}(\mathbf{V})$ under inverse images of *lm*-morphisms, Lemma 1. ◀

We don't know whether it is always true that for *sp*-varieties of monoids $\mathbf{V}$, $\mathcal{L}(\mathbf{QV})$ is included into $\mathcal{P}(\mathbf{V})$. We can only prove it for local varieties.

▶ **Proposition 4.** *Let* $\mathbf{V}$ *be a local* sp*-variety of monoids. Then* $\mathcal{P}(\mathbf{V}) \cap \mathcal{R}\mathsf{eg} = \mathcal{L}(\mathbf{QV})$.

Proof. This follows from the fact that for local varieties $\mathcal{L}(\mathbf{QV}) = \mathcal{L}(\mathbf{V} * \mathbf{Mod})$ [9]. The result follows from Proposition 3, as we always have $\mathcal{L}(\mathbf{V} * \mathbf{Mod}) \subseteq \mathcal{P}(\mathbf{V})$. ◀

As $\mathbf{A}$ is local [31, Example 15.5] and an *sp*-variety, it follows from Proposition 4 that the regular languages in $\mathcal{P}(\mathbf{A})$, hence in AC^0, are precisely those in $\mathcal{L}(\mathbf{QA})$, which is the characterization of the regular languages in AC^0 obtained by Barrington, Compton, Straubing and Thérien [4].

We will see in the next section that $\mathbf{DA}$ is an *sp*-variety. As it is also local [2], we get from Proposition 4 that the regular languages of $\mathcal{P}(\mathbf{DA})$ are precisely those in $\mathcal{L}(\mathbf{QDA})$.

As explained in the introduction, the language $(a+b)^*ac^+$ can be *p*-recognized by a program over $\mathbf{J}$. A simple algebraic argument shows that it is not in $\mathcal{L}(\mathbf{QJ})$. Hence, by Proposition 3, we have the following result:

▶ **Proposition 5.** $\mathbf{J}$ *is not an* sp*-variety of monoids.*

4 The case of DA

In this section, we prove that $\mathbf{DA}$ is an *sp*-variety of monoids. Combined with the fact that $\mathbf{DA}$ is local [2], we obtain the following result by Proposition 4.

▶ **Theorem 6.** $\mathcal{P}(\mathbf{DA}) \cap \mathcal{R}\mathsf{eg} = \mathcal{L}(\mathbf{QDA})$.

The result follows from the following main technical contribution:

▶ **Proposition 7.** $(c+ab)^*$, $(b+ab)^*$ *and* $((b^*ab^*)^k)^*$ *for any integer* $k \geq 2$ *are regular languages not in* $\mathcal{P}(\mathbf{DA})$.

Before proving the proposition we first show that it implies that $\mathbf{DA}$ is an *sp*-variety of monoids. Assume S is a finite semigroup such that $\mathcal{W}(S) \subseteq \mathcal{P}(\mathbf{DA})$. Let $\eta_S \colon S^* \to S^1$ be the evaluation morphism of S. We need to show that S^1 is in $\mathbf{DA}$.

Assume first that S^1 is aperiodic. Towards a contradiction, assume S^1 is not in $\mathbf{DA}$. Then, there exist x, y in S such that $(xy)^\omega \neq (xy)^\omega x (xy)^\omega$.

Set $e = (xy)^\omega$, $f = (yx)^\omega$, $s = ex$ and $t = ye$. Our hypothesis says that $exe \neq e$. We now have two cases, depending on whether $fyf = f$ or not.

So, suppose $fyf \neq f$. In that case, let $\mu\colon \{a,b,c\}^* \to S^1$ be the morphism sending a to s, b to t and c to e and consider the language $L = \mu^{-1}(\{1,e\})$. Assume that L contains a word w with two consecutive a. Then $w = w_1aaw_2$ and as $w \in L$, either $e = \mu(w_1)exex\mu(w_2)$ or $1 = \mu(w_1)exex\mu(w_2)$. In both cases $e = u_1exeu_2$ for some suitable values of u_1 and u_2. This implies that $e = u_1^\omega e(xeu_2)^\omega = (u_1xe)^\omega eu_2^\omega$. Because S^1 is aperiodic, this implies: $e = exeu_2 = eu_2$. Hence $exe = e$, contradicting the fact that $exe \neq e$. Similar arguments show that L cannot contain a word with two consecutive b, a factor ac or a factor cb.

Any word in L is of the form $u_0v_1u_1\cdots u_{k-1}v_ku_k$ where $k \in \mathbb{N}$, $v_1,\ldots,v_k \in c^+$ and $u_0,\ldots,u_k \in (a+b)^*$. As w does not contain aa nor bb as a factor, we have that $u_0,\ldots,u_k \in (b+\varepsilon)(ab)^*(a+\varepsilon)$. When $k \geq 1$, as moreover w does not contain ac nor cb as a factor, it follows that $u_1,\ldots,u_{k-1} \in (ab)^*$, $u_0 \in (b+\varepsilon)(ab)^*$ and $u_k \in (ab)^*(a+\varepsilon)$; now since $\mu(ab) = exye = e$ by aperiodicity and $\mu(b)e = yee = ye \notin \{1,e\}$ (otherwise $fyf = f$), $e\mu(a) = eex = ex \notin \{1,e\}$ (otherwise $exe = e$), $\mu(b)e\mu(a) = yeeex = yex = f \notin \{1,e\}$ (by aperiodicity and as otherwise $exe = e$), w cannot start with b or end with a, hence $u_0, u_k \in (ab)^*$. And similarly, $u_0 \in (ab)^*$ when $k = 0$. Therefore, $L = (c+ab)^*$.

The other case, when $fyf = f$, is treated similarly using the morphism $\mu\colon \{a,b\}^* \to S^1$ sending a to s and b to t and considering the language $L = \mu^{-1}(\{1,e,t\})$. Using arguments as for the previous case, one can conclude that $L = (b+ab)^*$.

Assume now that S^1 is not aperiodic. As in the two previous cases, we can then prove that there exist a morphism $\mu\colon \{a,b\}^* \to S^1$ and a subset $F \subseteq S^1$ such that $L = \mu^{-1}(F)$ is the regular language $((b^*ab^*)^k)^*$ for some $k \in \mathbb{N}, k \geq 2$ of all words over $\{a,b\}$ with a number of a congruent to 0 modulo k.

In all cases, we have a language L defined as $\mu^{-1}(Q)$ for some subset Q of S^1 and some morphism μ to S^1 sending letters to elements of S. As $\mathcal{W}(S) \subseteq \mathcal{P}(\mathbf{DA})$, it follows that $\eta_S^{-1}(Q)$ is p-recognized by a sequence of programs $(P_n)_{n\in\mathbb{N}}$ over a monoid in $\mathbf{DA}$. As in the example prior to Proposition 3 in the previous section, for each $n \in \mathbb{N}$, we can transform P_n into P'_n over the same monoid so that the sequence of programs $(P'_n)_{n\in\mathbb{N}}$ p-recognizes L. In all cases, we get a contradiction with Proposition 7.

In the remaining part of this section we prove Proposition 7.

Proof of Proposition 7. The idea of the proof is the following. We work by contradiction and assume that we have a sequence of programs over some monoid M of $\mathbf{DA}$ deciding one of the targeted language. Let n be much larger than the size of M, and let P_n be the program running on words of length n. Consider a language of the form Δ^* for some finite set Δ of words (for instance assume $\Delta = \{c, ab\}$, $\Delta = \{b, ab\}$, ...). We will show that we can fix a constant (depending on M and Δ but not on n) number of entries to P_n such that P_n always accepts or always rejects and there is a completion of the fixed entries in Δ^*; hence, if Δ was chosen so that there is actually a completion of the fixed entries in the targeted language and one outside of it, P_n cannot recognize it. We cannot prove this for all Δ, in particular it will not work for $\Delta = \{ab\}$ and indeed $(ab)^*$ is in $\mathcal{P}(\mathbf{DA})$. The key property of our Δ is that after fixing any letter at any position, except maybe for a constant number of positions, one can still complete the word into one within Δ^*. This is not true for $\Delta = \{ab\}$ because after fixing a b in an odd position all completions fall outside of $(ab)^*$.

We now add some technical details.

Let Δ be a finite set of words. Let Σ be the corresponding finite alphabet and let $\perp$ be a letter not in Σ. A *mask* is a word over $\Sigma \cup \{\perp\}$. The positions of a mask carrying a $\perp$ are called *free* while the positions carrying a letter in Σ are called *fixed*. A mask λ' is a *submask* of a mask λ if it is formed from λ by replacing some occurrences of $\perp$ by a letter in Σ.

A *completion* of a mask λ is a word w over Σ that is built from λ by replacing all occurrences of $\perp$ by a letter in Σ. Notice that all completions of a mask have the same length as the mask itself. A mask λ is Δ-*compatible* if it has a completion in Δ^*.

The *dangerous* positions of a mask λ are the positions within distance $2l-2$ of the fixed positions or within distance $l-1$ of the beginning or the end of the mask, where l is the maximal length of a word in Δ. A position that is not dangerous is said to be *safe*.

We say that Δ is *safe* if the following holds. Let λ be a Δ-compatible mask. Let i be any free position of λ that is not dangerous. Let a be any letter in Σ. Then the submask of λ constructed by fixing a at position i is Δ-compatible. We have already seen that $\Delta = \{ab\}$ is not safe. However our targeted Δ, $\Delta = \{c, ab\}$, $\Delta = \{b, ab\}$, $\Delta = \{a, b\}$, are safe. We always consider Δ to be safe in the following.

Finally, we say that a completion w of a mask λ is *safe* if w is a completion of λ belonging to Δ^* or is constructed from a completion of λ in Δ^* by modifying only letters at safe positions of λ, the dangerous positions remaining unchanged.

The following lemma is the key to the proof. It shows that modulo fixing a few entries, one can fix the output.

▶ **Lemma 8.** *Let M be a monoid in* **DA**. *Let λ be a Δ-compatible mask of length n, let P be a program over M of range n, let u and v be elements of M. Then there is an element t of M and a Δ-compatible submask λ' of λ obtained by fixing a number of free positions independent from n such that any safe completion w of λ' verifies $uP(w)v = t$.*

The proof of Lemma 8 is technical. As often in this setting, it relies on Green's relations for decomposing monoids. Then, at each stage of the decomposition, a small number of safe positions are fixed accordingly. Details can be found in Appendix A.

Setting $\Delta = \{c, ab\}$ or $\Delta = \{b, ab\}$, when applying Lemma 8 for some monoid $M \in \mathbf{DA}$ with the trivial Δ-compatible mask λ of length n containing only free positions, with P some program over M of range n and with u and v the identity of M, the resulting mask λ' has the property that we have an element t of M such that $P(w) = t$ for any safe completion w of λ'. Since the mask λ' is Δ-compatible and as long as n is big enough, we have a safe completion $w_0 \in \Delta^*$ and a safe completion $w_1 \notin \Delta^*$. Hence P cannot recognize Δ^*. This implies that $(c+ab)^* \notin \mathcal{P}(M)$ and $(b+ab)^* \notin \mathcal{P}(M)$. Finally, for any $k \in \mathbb{N}, k \geq 2$, we can prove that $((b^*ab^*)^k)^* \notin \mathcal{P}(M)$ by setting $\Delta = \{a, b\}$ and completing the mask given by the lemma by setting the letters in such a way that we have the right number of a modulo k in one case and not in the other case.

This concludes the proof of Proposition 7 because the argument above holds for any monoid in **DA**.

5 A fine hierarchy in $\mathcal{P}(\mathbf{DA})$

The definition of p-recognition by a sequence of programs over a monoid given in Section 2 requires that for each n, the program reading the entries of length n has a length polynomial in n. In the case of $\mathcal{P}(\mathbf{DA})$, the polynomial length restriction is without loss of generality: any program over a monoid in **DA** is equivalent to one of polynomial length over the same monoid [29] (in the sense that they recognize the same languages). In this section, we show that this does not collapse further: in the case of **DA**, programs of length $O(n^{k+1})$ express strictly more than those of length $O(n^k)$.

Following [13], we use an alternative definition of the languages recognized by a monoid in **DA**. We define by induction a hierarchy of classes of languages $\mathcal{SUM}_k$, where $\mathcal{SUM}$ stands for *strongly unambiguous monomial*. A language L is in $\mathcal{SUM}_0$ if it is of the form

A^* for some finite alphabet A. A language L is in $\mathcal{SUM}_k$ if it is in $\mathcal{SUM}_{k-1}$ or $L = L_1 a L_2$ for some languages $L_1 \in \mathcal{SUM}_i$ and $L_2 \in \mathcal{SUM}_j$ and some letter a with $i + j = k - 1$ such that no word of L_1 contains the letter a or no word of L_2 contains the letter a.

Gavaldà and Thérien showed that a language L is recognized by a monoid in **DA** iff there is a k such that L is a Boolean combination of languages in $\mathcal{SUM}_k$ [13]. For each $k \in \mathbb{N}$, we denote by $\mathbf{DA_k}$ the variety of monoids generated by the syntactic monoids of the Boolean combinations of languages in $\mathcal{SUM}_k$. It can be checked that, for each k, $\mathbf{DA_k}$ forms a variety of monoids recognizing precisely Boolean combinations of languages in $\mathcal{SUM}_k$ (see Appendix B).

5.1 Strict hierarchy

For each k we exhibit a language $L_k \subseteq \{0,1\}^*$ that can be recognized by a sequence of programs of length $\mathrm{O}(n^k)$ over a monoid M_k in **DA** but cannot be recognized by any sequence of programs of length $\mathrm{O}(n^{k-1})$ over any monoid in **DA**.

The language L_k expresses a property of the first k occurrences of 1 in the input word. To define L_k we say that S is a *k-set over n* if S is a set where each element is an ordered tuple of k distinct elements of $[n]$. For any sequence $\Delta = (S_n)_{n\in\mathbb{N}}$ of k-sets over n, we set $L_\Delta = \bigcup_{n\in\mathbb{N}} K_{n,S_n}$, where K_{n,S_n} is the set of words over $\{0,1\}$ of length n such that for each of them, it contains at least k occurrences of 1 and the ordered k-tuple of the positions of the first k occurrences of 1 belongs to S_n.

On the one hand, we show that for all k there is a monoid M_k in **DA** such that for all Δ the language L_Δ is recognized by a sequence of programs over M_k of length $\mathrm{O}(n^k)$. The proof is done by an inductive argument on k.

On the other hand, we show that there is a Δ such that for any finite monoid M and any sequence of programs $(P_n)_{n\in\mathbb{N}}$ over M of length $\mathrm{O}(n^{k-1})$, L_Δ is not recognized by $(P_n)_{n\in\mathbb{N}}$. This is done using a counting argument: for some monoid size i, for n big enough, the number of languages in $\{0,1\}^n$ recognized by a program over some monoid of size i of length at most $\alpha \cdot n^{k-1}$ is upper-bounded by a number that turns out to be asymptotically smaller than the number of different possible K_{n,S_n}.

Upper bound. We start with the upper bound. Notice that for $\Delta = (S_n)_{n\in\mathbb{N}}$, the language of words of length n of L_Δ is exactly K_{n,S_n}. Hence the fact that L_Δ can be recognized by a sequence of programs over a monoid in **DA** of length $\mathrm{O}(n^k)$ is a consequence of the following proposition.

▶ **Proposition 9.** *For all $k \in \mathbb{N}_{>0}$ there is a monoid $M_k \in \mathbf{DA_k}$ such that for all $n \in \mathbb{N}$ and all S_n k-sets over n, the language K_{n,S_n} is recognized by a program over M_k of length at most $4n^k$.*

Proof. We first define by induction on k a family of languages Z_k over the alphabet $Y_k = \{\bot_l, \top_l \mid 1 \le l \le k\}$. For $k = 0$, Z_0 is $\{\varepsilon\}$. For $k > 0$, Z_k is the set of words containing $\top_k$ and such that the first occurrence of $\top_k$ has no $\bot_k$ to its left, and the sequence between the first occurrence of $\top_k$ and the first occurrence of $\bot_k$ or $\top_k$ to its right, or the end of the word if there is no such letter, belongs to Z_{k-1}. A simple induction on k shows that Z_k is defined by the following expression

$$Y_{k-1}^* \top_k Y_{k-2}^* \top_{k-1} \cdots Y_1^* \top_2 \top_1 Y_k^*$$

and therefore it is in $\mathcal{SUM}_k$ and its syntactic monoid M_k is in $\mathbf{DA_k}$.

Fix n. If $n = 0$, the proposition follows trivially, otherwise, we define by induction on k a program $P_k(i, S)$ for every k-set S over n and every $1 \leq i \leq n+1$ that will for the moment output letters of Y_k instead of outputting elements of M_k.

For any $k > 0$ and S a k-set over n, let $f_{j,S}$ be the function with $f_{j,S}(0) = \varepsilon$ and $f_{j,S}(1) = \top_k$ if j is the first element of some ordered k-tuple of S, $f_{j,S}(1) = \bot_k$ otherwise. We also let g_k be the function with $g_k(0) = \varepsilon$ and $g_k(1) = \bot_k$. If S is a k-set over n and $j \leq n$ then $S|j$ denotes the $(k-1)$-set over n containing the ordered $(k-1)$-tuples $\bar{t}$ such that $(j, \bar{t}) \in S$.

For $k > 0$, $1 \leq i \leq n+1$ and S a k-set over n, the program $P_k(i, S)$ is the following sequence of instructions:

$$(i, f_{i,S})P_{k-1}(i+1, S|i)(i, g_k) \cdots (n, f_{n,S})P_{k-1}(n+1, S|n)(n, g_k).$$

In other words, the program guesses the first occurrence $j \geq i$ of 1, returns $\bot_k$ or $\top_k$ depending on whether it is the first element of an ordered k-tuple in S, and then proceeds for the next occurrences of 1 by induction.

For $k = 0$, $1 \leq i \leq n+1$ and S a 0-set over n (that is empty or contains ε, the only ordered 0-tuple of elements of $[n]$), the program $P_0(i, S)$ is the empty program ε.

A simple computation shows that for any $k \in \mathbb{N}_{>0}$, $1 \leq i \leq n+1$ and S a k-set over n, the number of instructions in $P_k(i, S)$ is at most $4n^k$.

A simple induction on k shows that when running on a word $w \in \{0,1\}^n$, for any $k \in \mathbb{N}_{>0}$, $1 \leq i \leq n+1$ and S a k-set over n, $P_k(i, S)$ returns a word in Z_k iff the ordered k-tuple of the positions of the first k occurrences of 1 starting at position i in w exists and is an element of S.

For any $k > 0$ and S_n a k-set over n, it remains to apply the syntactic morphism of Z_k to the output of the functions in the instructions of $P_k(1, S_n)$ to get a program over M_k of length at most $4n^k$ recognizing K_{n,S_n}. ◀

Lower bound. The following claim is a simple counting argument.

▶ **Claim 10.** *For all $i \in \mathbb{N}_{>0}$ and $n \in \mathbb{N}$, the number of languages in $\{0,1\}^n$ recognized by programs over a monoid of size i, reading inputs of length n over the alphabet $\{0,1\}$, with at most $l \in \mathbb{N}$ instructions, is bounded by $i^{i^2} 2^i \cdot (n \cdot i^2)^l$.*

Proof. Fix a monoid M of size i. Since a program over M of range n with less than l instructions can always be completed into such a program with exactly l instructions recognizing the same languages in $\{0,1\}^n$ (using the identity of M), we only consider programs with exactly l instructions. As $\Sigma = \{0,1\}$, there are $n \cdot i^2$ choices for each of the l instructions of a range n program over M reading inputs in $\{0,1\}^*$. Such a program can recognize at most 2^i different languages in $\{0,1\}^n$. Hence, the number of languages in $\{0,1\}^n$ recognized by programs over M of length at most l is at most $2^i \cdot (n \cdot i^2)^l$. The result follows from the facts that there are at most i^{i^2} isomorphism classes of monoids of size i and that two isomorphic monoids allow to recognize the same languages in $\{0,1\}^n$. ◀

If for some $k \in \mathbb{N}_{>0}$ and $1 \leq i \leq \alpha$, $\alpha \in \mathbb{N}_{>0}$, we apply Claim 10 for all $n \in \mathbb{N}$, $l = \alpha \cdot n^{k-1}$, we get a number of languages upper-bounded by $n^{\mathrm{O}(n^{k-1})}$, which is asymptotically strictly smaller than the number of distinct K_{n,S_n}, which is $2^{\binom{n}{k}}$.

Hence, for all $j \in \mathbb{N}_{>0}$, there exist an $n_j \in \mathbb{N}$ and T_j a k-set over n_j such that no program over a monoid of size $1 \leq i \leq j$ and of length at most $j \cdot n^{k-1}$ recognizes K_{n_j,T_j}. Moreover, we can assume without loss of generality that the sequence $(n_j)_{j \in \mathbb{N}_{>0}}$ is increasing. Let

$\Delta = (S_n)_{n\in\mathbb{N}}$ be such that $S_{n_j} = T_j$ for all $j \in \mathbb{N}_{>0}$ and $S_n = \emptyset$ for any $n \in \mathbb{N}$ verifying that it is not equal to any n_j for $j \in \mathbb{N}_{>0}$. We show that no sequence of programs over a finite monoid of length $\mathrm{O}(n^{k-1})$ can recognize L_Δ. If this were the case, then let i be the size of the monoid. Let $j \geq i$ be such that for any $n \in \mathbb{N}$, the n-th program has length at most $j \cdot n^{k-1}$. But, by construction, we know that there does not exist any such program of range n_j recognizing K_{n_j,T_j}, a contradiction.

This implies the following hierarchy (where $\mathcal{P}(\mathbf{V}, s(n))$ for some variety of monoids $\mathbf{V}$ and a function $s\colon \mathbb{N} \to \mathbb{N}$ denotes the class of languages recognizable by a sequence of programs of length $\mathrm{O}(s(n))$):

▶ **Proposition 11.** *For all $k \in \mathbb{N}$, $\mathcal{P}(\mathbf{DA}, n^k) \subsetneq \mathcal{P}(\mathbf{DA}, n^{k+1})$.*

5.2 Collapse

Tesson and Thérien showed that any program over a monoid M in **DA** is equivalent to one of polynomial length [29]. We now show that if we further assume that M is in $\mathbf{DA_k}$ then the length can be assumed to be $\mathrm{O}(n^k)$.

▶ **Proposition 12.** *Let $k > 0$. Let $M \in \mathbf{DA_k}$. Then any program over M is equivalent to a program over M of length $\mathrm{O}(n^k)$.*

The equivalent program of length $\mathrm{O}(n^k)$ is actually a subprogram of the initial one. For each possible acceptance set, an input word to the program is accepted iff the word over the alphabet M produced by the program belongs to some fixed Boolean combination of languages in $\mathcal{SUM}_k$. The idea is then just to keep enough instructions so that membership of the produced word over M in each of these languages does not change. For each of those languages, the set of instructions to keep is defined by induction on k using the inductive definition of $\mathcal{SUM}_k$ given at the beginning of this section. Roughly, at each step, for each input letter and each input position, the small program keeps the first or last instruction of the big program producing the "pivot element" when reading this input letter at that position. The number of instructions kept in the end is then in $\mathrm{O}(n^k)$. The details can be found in Appendix C.

6 Conclusion

For local and tame varieties $\mathbf{V}$ we have shown that the regular languages recognized by programs over $\mathbf{V}$ are exactly those in $\mathcal{L}(\mathbf{QV})$. It is not clear whether locality is necessary. We don't have any example of a tame variety $\mathbf{V}$ for which $\mathcal{L}(\mathbf{QV})$ is not included into $\mathcal{P}(\mathbf{V})$. We leave this question for future work.

We have shown that **DA** is tame but that **J** is not. Another example of a tame variety is **A**. However we needed the fact that $\mathrm{MOD_m}$ is not in $\mathsf{AC^0}$ for all $m \geq 2$ in order to prove tameness. It would be interesting to give a direct algebraic proof of this result. As this would in particular imply that $\mathrm{MOD_2}$ is not in $\mathsf{AC^0}$ by Proposition 4, it is certainly a challenging task.

Finding the regular languages recognized by programs over **J** is left for future work.

To conclude we should add, in fairness, that the progress reported here does not obviously bring us closer to major $\mathsf{NC^1}$ complexity subclasses separations, but it does uncover new ways in which a program can or cannot circumvent the limitations imposed by the underlying monoid algebraic structure available to it.

References

1 Miklós Ajtai. Σ^1_1-formulae on finite structures. In *Ann. Pure and Appl. Logic*, volume 24, pages 1–48, 1983.

2 Jorge Almeida. A syntactical proof of locality of DA. *Int. J. of Algebra and Computation (IJAC)*, 6(2):165–178, 1996.

3 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

4 David A. Mix Barrington, Kevin J. Compton, Howard Straubing, and Denis Thérien. Regular languages in NC^1. *J. Comput. Syst. Sci.*, 44(3):478–499, 1992.

5 David A. Mix Barrington, Howard Straubing, and Denis Thérien. Non-uniform automata over groups. *Inf. Comput.*, 89(2):109–132, 1990.

6 David A. Mix Barrington and Denis Thérien. Finite monoids and the fine structure of NC^1. *J. ACM*, 35(4):941–952, 1988.

7 Laura Chaubard, Jean-Éric Pin, and Howard Straubing. Actions, wreath products of C-varieties and concatenation product. *Theoretical Computer Science*, 356(1-2):73–89, 2006.

8 Luc Dartois. *Méthodes algébriques pour la théorie des automates*. PhD thesis, Université Paris Diderot, Paris, 2014.

9 Luc Dartois and Charles Paperman. Adding modular predicates. *CoRR*, abs/1401.6576, 2014. URL: `http://arxiv.org/abs/1401.6576`.

10 Samuel Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, New York, 1974.

11 Samuel Eilenberg. *Automata, Languages, and Machines*, volume B. Academic Press, New York, 1976.

12 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

13 Ricard Gavaldà and Denis Thérien. Algebraic characterizations of small classes of Boolean functions. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 331–342. Springer, 2003.

14 Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 6–20. ACM, 1986.

15 Kenneth Krohn and John L. Rhodes. Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. In *Trans. Amer. Math. Soc.*, volume 116, pages 450–464, 1965.

16 Alexis Maciel, Pierre Péladeau, and Denis Thérien. Programs over semigroups of dot-depth one. *Theor. Comput. Sci.*, 245(1):135–148, 2000.

17 Ward D. Maurer and John L. Rhodes. A property of finite simple non-abelian groups. In *Amer. Math. Soc.*, volume 16, pages 552–554, 1965.

18 Pierre McKenzie, Pierre Péladeau, and Denis Thérien. NC^1: The automata-theoretic viewpoint. *Computational Complexity*, 1:330–359, 1991.

19 Charles Paperman. *Circuits booléens, prédicats modulaires et langages réguliers*. PhD thesis, Université Paris Diderot, Paris, 2014.

20 Pierre Péladeau. *Classes de circuits booléens et variétés de monoïdes*. PhD thesis, Université Pierre-et-Marie-Curie (Paris-VI), Paris, France, 1990.

21 Pierre Péladeau, Howard Straubing, and Denis Thérien. Finite semigroup varieties defined by programs. *Theor. Comput. Sci.*, 180(1-2):325–339, 1997.

22 Jean-Éric Pin. *Varieties of formal languages*. North Oxford, London and Plenum, New-York, 1986. (Traduction de Variétés de langages formels).

23 Jean-Éric Pin and Howard Straubing. Some results on $\mathcal{C}$-varieties. *RAIRO-Theor. Inf. Appl.*, 39(1):239–262, 2005.

24 Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes*, 41(4):333–338, 1987.

25 Jan Reiterman. The Birkhoff theorem for finite algebras. *algebra universalis*, 14(1):1–10, 1982.

26 Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages, 2nd GI Conference*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer, 1975.

27 Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 77–82. ACM, 1987.

28 Pascal Tesson. *Computational Complexity Questions Related to Finite Monoids and Semigroups*. PhD thesis, McGill University, Montreal, Canada, 2003.

29 Pascal Tesson and Denis Thérien. The computing power of programs over finite monoids. *J. Autom. Lang. Comb.*, 7(2):247–258, November 2001.

30 Pascal Tesson and Denis Thérien. Diamonds are forever: the variety DA. *Semigroups, algorithms, automata and languages*, 1:475–500, 2002.

31 Bret Tilson. Categories as algebra: An essential ingredient in the theory of monoids. *J. of Pure and Applied Algebra*, 48(1-2), 1987.

A Missing proofs from Section 4

Let M be a monoid in **DA** whose identity we will denote by 1.

Given two elements u, u' of M we say that $u \leq_J u'$ if there are elements v, v' of M such that $u' = vuv'$. We write $u \sim_J u'$ if $u \leq_J u'$ and $u' \leq_J u$. We write $u <_J u'$ if $u \leq_J u'$ and $u' \not\sim_J u$. Given two elements u, u' of M we say that $u \leq_R u'$ if there is an element v of M such that $u' = uv$. We write $u \sim_R u'$ if $u \leq_R u'$ and $u' \leq_R u$. We write $u <_R u'$ if $u \leq_R u'$ and $u' \not\sim_R u$. Given two elements u, u' of M we say that $u \leq_L u'$ if there is an element v of M such that $u' = vu$. We write $u \sim_L u'$ if $u \leq_L u'$ and $u' \leq_L u$. We write $u <_L u'$ if $u \leq_L u'$ and $u' \not\sim_L u$.

We shall use the following well-known fact about these preorders and equivalence relations (see [22, Chapter 3, Proposition 1.4]).

▶ **Lemma 13.** *For all elements u and v of M, if $u \leq_R v$ and $u \sim_J v$, then $u \sim_R v$. Similarly, if $u \leq_L v$ and $u \sim_J v$, then $u \sim_L v$.*

An element r of M is R-bad for u if $u <_R ur$. Similarly an element r of M is L-bad for v if $u <_L rv$. It follows from $M \in$ **DA** that being R-bad or L-bad only depends on the $\sim_R$ or $\sim_L$ class, respectively:

▶ **Lemma 14.** *[Folklore] If M is in* **DA**, *then $u \sim_R u'$ and $ur \sim_R u$ implies $u'r \sim_R u$. Similarly $u \sim_L u'$ and $ru \sim_L u$ implies $ru' \sim_L u$.*

Let Δ be a finite set of words and Σ be the corresponding finite alphabet, Δ being safe, and let $n \in \mathbb{N}$. We are now going to prove the main technical lemma that allows us to assert that after fixing a constant number of positions in the input of a program over M, it can still be completed into a word of Δ^*, but the program cannot make the difference between any two possible completions anymore. To prove the lemma, we define a relation $\prec$ on the set of quadruplets (λ, P, u, v) where λ is a mask of length n, P is a program over M for words of length n and u and v are two elements. We will say that an element $(\lambda_1, P_1, u_1, v_1)$ is strictly smaller than $(\lambda_2, P_2, u_2, v_2)$, written $(\lambda_1, P_1, u_1, v_1) \prec (\lambda_2, P_2, u_2, v_2)$ if λ_1 is a submask of λ_2, P_1 is a subprogram of P_2 and one of the following cases occurs:

1. $u_2 <_R u_1$ and $v_1 = v_2$ and P_1 is a suffix of P_2 and $u_1P_1(w)v_1 = u_2P_2(w)v_2$ for all safe completions w of λ_1;
2. $v_2 <_L v_1$ and $u_1 = u_2$ and P_1 is a prefix of P_2 and $u_1P_1(w)v_1 = u_2P_2(w)v_2$ for all safe completions w of λ_1;
3. $u_2 = u_1$ and $v_1 = 1$ and P_1 is a prefix of P_2 and $u_1P_1(w)v_1 <_J u_2P_2(w)v_2$ for all safe completions w of λ_1;
4. $v_2 = v_1$ and $u_1 = 1$ and P_1 is a suffix of P_2 and $u_1P_1(w)v_1 <_J u_2P_2(w)v_2$ for all safe completions w of λ_1.

Note that, since M is finite, this relation is well-founded (that is, it has no infinite decreasing chain, an infinite sequence of quadruplets $\mu_0, \mu_1, \mu_2, \dots$ such that $\mu_{i+1} \prec \mu_i$ for all $i \in \mathbb{N}$) and the maximal length of any decreasing chain depends only on M.

▶ **Lemma 8** (restated). *Let λ be a Δ-compatible mask of length n, let P be a program over M of range n, let u and v be elements of M. Then there is an element t of M and a Δ-compatible submask λ' of λ obtained by fixing a number of free positions independent from n such that any safe completion w of λ' verifies $uP(w)v = t$.*

Proof. The proof goes by induction on $\prec$.

Let λ be a Δ-compatible mask of length n, let P be a program over M for words of length n, let u and v be elements of M such that (λ, P, u, v) is of height h, and assume that for any quadruplet (λ', P', u', v') strictly smaller than (λ, P, u, v), the lemma is verified. Consider the following conditions concerning the quadruplet (λ, P, u, v):

- **(a)** there does not exist any instruction (x, f) of P such that for some letter a the submask λ' of λ formed by setting position x to a (if it wasn't already the case) is Δ-compatible and $f(a)$ is R-bad for u;
- **(b)** v is not R-bad for u;
- **(c)** there does not exist any instruction (x, f) of P such that for some letter a the submask λ' of λ formed by setting position x to a (if it wasn't already the case) is Δ-compatible and $f(a)$ is L-bad for v;
- **(d)** u is not L-bad for v.

We will now do a case analysis based on which of these conditions are violated or not.

Case 1: condition 1 is violated. So there exists some instruction (x, f) of P such that for some letter a the submask λ' of λ formed by setting position x to a (if it wasn't already the case) is Δ-compatible and $f(a)$ is R-bad for u. Let i be the smallest number of such an instruction.

Let P' be the subprogram of P until instruction $i-1$. Let w be a safe completion of λ. By minimality of i and by Lemma 14, it follows that $u \sim_R uP'(w)$.

So, because $f(a)$ is R-bad for u, any safe completion w of λ', which is also a safe completion of λ, is such that $u \sim_R uP'(w) <_R uP'(w)f(a) \leq_R uP(w)v$ by Lemma 14, hence $uP'(w) <_J uP(w)v$ by Lemma 13. So $(\lambda', P', u, 1) \prec (\lambda, P, u, v)$, therefore, by induction we get a Δ-compatible submask λ_1 of λ' and a monoid element t_1 such that $uP'(w) = t_1$ for all safe completions w of λ_1.

Let P'' be the subprogram of P starting from instruction $i+1$. Notice that, since $u \sim_R t_1$ (by what we have proven just above), $u <_R t_1f(a)$ (by Lemma 14) and $t_1f(a)P''(w)v = uP'(w)f(a)P''(w)v = uP(w)v$ for all safe completions w of λ_1. Hence, $(\lambda_1, P'', t_1f(a), v)$ is strictly smaller than (λ_1, P, u, v) and by induction we get a Δ-compatible submask λ_2 of λ_1 and a monoid element t such that $t_1f(a)P''(w)v = t$ for all safe completions w of λ_2.

Hence any safe completion w of λ_2 is such that

$$uP(w)v = uP'(w)f(a)P''(w)v = t_1 f(a)P''(w)v = t\ .$$

Hence λ_2 and t are the desired solutions.

Case 2: condition 1 is verified but condition 2 is violated, so v is R-bad for u and Case 1 does not apply.

Let w be a safe completion of λ: for any instruction (x, f) of P, as the submask λ' of λ formed by setting position x to w_x (if it wasn't already the case) is Δ-compatible (by the fact that Δ is safe and w is a safe completion of λ), $f(w_x)$ cannot be R-bad for u, otherwise condition 1 would be violated, so $u \sim_R uf(w_x)$. Hence, by Lemma 14, $u \sim_R uP(w)$ for all safe completions w of λ. Notice then that $u \sim_R uP(w) <_R uP(w)v$ (by Lemma 14), hence $uP(w) <_J uP(w)v$ (by Lemma 13) for all safe completions w of λ. So $(\lambda, P, u, 1) \prec (\lambda, P, u, v)$, therefore we obtain a monoid element t_1 and a Δ-compatible submask λ' of λ by induction such that $uP(w) = t_1$ for all completions w of λ'. $t = t_1 v$ is the desired element of M.

Case 3: condition 3 is violated. So there exists some instruction (x, f) of P such that for some letter a the submask λ' of λ formed by setting position x to a (if it wasn't already the case) is Δ-compatible and $f(a)$ is L-bad for v.

We proceed as for Case 1 by symmetry.

Case 4: condition 3 is verified but condition 4 is violated, so u is L-bad for v and Case 3 does not apply.

We proceed as for Case 2 by symmetry.

Case 5: conditions 1, 2, 3 and 4 are verified.

As it Case 2 and Case 4 we get that $u \sim_R uP'(w)$ and $v \sim_L P''(w)v$ for any prefix P' of P, any suffix P'' of P and all safe completions w of λ. Moreover, since condition 2 and condition 4 are verified, by Lemma 14, we get that $uP(w)v \sim_R u$ and $uP(w)v \sim_L v$ for all safe completions w of λ.

Let w_0 be a completion of λ that is in Δ^*. Let λ' be the submask of λ fixing all dangerous positions of λ using w_0. Then, for any completion w of λ', which is a safe completion of λ by construction, we have that $uP(w)v \sim_R u$ and $uP(w)v \sim_L v$. As M is aperiodic, this implies that there is a t in M such that $uP(w)v = t$ for all completions w of λ' (see [22, Chapter 3, Proposition 4.2]).

This concludes the proof of the lemma. ◀

B $\mathcal{SUL}_k$ is a variety of languages

A *variety of languages* is a class of languages over arbitrary finite alphabets closed under Boolean operations, quotients and inverses of morphisms (i.e. if L is a language in the class over a finite alphabet Σ, if Γ is some other finite alphabet and $\varphi\colon \Gamma^* \to \Sigma^*$ is a morphism of monoids, then $\varphi^{-1}(L)$ is also in the class).

Eilenberg showed [11, Chapter VII, Section 3] that there is a bijective correspondence between varieties of monoids and varieties of languages: to each variety of monoids $\mathbf{V}$ we can bijectively associate $\mathcal{L}(\mathbf{V})$ the variety of languages whose syntactic monoids belong to $\mathbf{V}$ and, conversely, to each variety of languages $\mathcal{V}$ we can bijectively associate $\mathbf{M}(\mathcal{V})$ the variety of monoids generated by the syntactic monoids of the languages of $\mathcal{V}$, and these correspondences are mutually inverse.

We denote by $\mathcal{SUL}_k$ the class of regular languages that are Boolean combinations of languages in $\mathcal{SUM}_k$.

In this appendix, we show that, for all $k \in \mathbb{N}$, $\mathcal{SUL}_k$ is a variety of languages. As $\mathbf{DA_k}$ is the variety of monoids generated by the syntactic monoids of the languages in $\mathcal{SUL}_k$, by Eilenberg's theorem, we know that, conversely, all the regular languages whose syntactic monoids lie in $\mathbf{DA_k}$ are in $\mathcal{SUL}_k$.

Closure under Boolean operations is obvious by construction. Closure under quotients and inverses of morphisms is respectively given by the following two lemmas and by the fact that both quotients and inverses of morphisms commute with Boolean operations.

Given a word u over a given finite alphabet Σ, we will denote by $\mathrm{alph}(u)$ the set of letters of Σ that appear in u.

▶ **Lemma 15.** *For all $k \in \mathbb{N}$, for all $L \in \mathcal{SUM}_k$ over a finite alphabet Σ and $u \in \Sigma^*$, $u^{-1}L$ and Lu^{-1} both are a union of languages in $\mathcal{SUM}_k$ over Σ.*

Proof. We prove it by induction on k.

- Base case: $k = 0$.
  Let $L \in \mathcal{SUM}_0$ over a finite alphabet Σ and $u \in \Sigma^*$. This means that $L = A^*$ for some $A \subseteq \Sigma$. We have two cases: either $\mathrm{alph}(u) \nsubseteq A$ and then $u^{-1}L = Lu^{-1} = \emptyset$; or $\mathrm{alph}(u) \subseteq A$ and then $u^{-1}L = Lu^{-1} = A^* = L$. So $u^{-1}L$ and Lu^{-1} both are a union of languages in $\mathcal{SUM}_0$ over Σ. The base case is hence proved.
- Inductive step. Let $k \in \mathbb{N}_{>0}$ and assume that the lemma is true for all $k' \in \mathbb{N}, k' < k$.
  Let $L \in \mathcal{SUM}_k$ over a finite alphabet Σ and $u \in \Sigma^*$. This means that either L is in $\mathcal{SUM}_{k-1}$ and the lemma is proved by applying the inductive hypothesis directly for L and u, or $L = L_1 a L_2$ for some languages $L_1 \in \mathcal{SUM}_i$ and $L_2 \in \mathcal{SUM}_j$ and some letter $a \in \Sigma$ with $i + j = k - 1$ and, either no word of L_1 contains the letter a or no word of L_2 contains the letter a. We shall only treat the case in which a does not appear in any of the words of L_1; the other case is treated symmetrically.
  There are again two cases to consider, depending on whether a does appear in u or not. If $a \notin \mathrm{alph}(u)$, then it is straightforward to check that $u^{-1}L = (u^{-1}L_1)aL_2$ and $Lu^{-1} = L_1 a(L_2 u^{-1})$. By the inductive hypothesis, we get that $u^{-1}L_1$ is a union of languages in $\mathcal{SUM}_i$ over Σ and that L_2u^{-1} is a union of languages in $\mathcal{SUM}_j$ over Σ. Moreover, it is direct to see that no word of $u^{-1}L_1$ contains the letter a. By distributivity of concatenation over union, we finally get that $u^{-1}L$ and Lu^{-1} both are a union of languages in $\mathcal{SUM}_k$ over Σ.
  If $a \in \mathrm{alph}(u)$, then let $u = u_1 a u_2$ with $u_1, u_2 \in \Sigma^*$ and $a \notin \mathrm{alph}(u_1)$. It is again straightforward to see that

$$u^{-1}L = \begin{cases} {u_2}^{-1}L_2 & \text{if } u_1 \in L_1 \\ \emptyset & \text{otherwise} \end{cases}$$

  and

$$Lu^{-1} = L_1 a(L_2 u^{-1}) \cup \begin{cases} L_1 {u_1}^{-1} & \text{if } u_2 \in L_2 \\ \emptyset & \text{otherwise} \end{cases}.$$

  Again, by the inductive hypothesis, we get that $L_1{u_1}^{-1}$ is a union of languages in $\mathcal{SUM}_i$ over Σ and that both ${u_2}^{-1}L_2$ and L_2u^{-1} are unions of languages in $\mathcal{SUM}_j$ over Σ. And, again, by distributivity of concatenation over union, we get that $u^{-1}L$ and Lu^{-1} both are a union of languages in $\mathcal{SUM}_k$ over Σ.
  This concludes the inductive step and therefore the proof of the lemma. ◀

▶ **Lemma 16.** *For all $k \in \mathbb{N}$, for all $L \in \mathcal{SUM}_k$ over a finite alphabet Σ and $\varphi\colon \Gamma^* \to \Sigma^*$ a morphism of monoids where Γ is another finite alphabet, $\varphi^{-1}(L)$ is a union of languages in $\mathcal{SUM}_k$ over Γ.*

Proof. We prove it by induction on k.

- Base case: $k = 0$.
  Let $L \in \mathcal{SUM}_0$ over a finite alphabet Σ and $\varphi\colon \Gamma^* \to \Sigma^*$ a morphism of monoids where Γ is another finite alphabet. This means that $L = A^*$ for some $A \subseteq \Sigma$. It is straightforward to check that $\varphi^{-1}(L) = B^*$ where $B = \{b \in \Gamma \mid \varphi(b) \in A^*\}$. B^* is certainly a union of languages in $\mathcal{SUM}_0$ over Σ. The base case is hence proved.
- Inductive step. Let $k \in \mathbb{N}_{>0}$ and assume that the lemma is true for all $k' \in \mathbb{N}, k' < k$.
  Let $L \in \mathcal{SUM}_k$ over a finite alphabet Σ and $\varphi\colon \Gamma^* \to \Sigma^*$ a morphism of monoids where Γ is another finite alphabet. This means that either L is in $\mathcal{SUM}_{k-1}$ and the lemma is proved by applying the inductive hypothesis directly for L and φ, or $L = L_1 a L_2$ for some languages $L_1 \in \mathcal{SUM}_i$ and $L_2 \in \mathcal{SUM}_j$ and some letter $a \in \Sigma$ with $i + j = k - 1$ and, either no word of L_1 contains the letter a or no word of L_2 contains the letter a. We shall only treat the case in which a does not appear in any of the words of L_1; the other case is treated symmetrically.
  Let us define $B = \{b \in \Gamma \mid a \in \mathrm{alph}(\varphi(b))\}$ as the set of letters of Γ whose image word by φ contains the letter a. For each $b \in B$, we shall also let $\varphi(b) = u_{b,1} a u_{b,2}$ with $u_{b,1}, u_{b,2} \in \Sigma^*$ and $a \notin u_{b,1}$. It is not too difficult to see that we then have

  $$\varphi^{-1}(L) = \bigcup_{b \in B} \varphi^{-1}(L_1 {u_{b,1}}^{-1}) b \varphi^{-1}({u_{b,2}}^{-1} L_2) \ .$$

  By the inductive hypothesis, by Lemma 15 and by the fact that inverses of morphisms commute with unions, we get that $\varphi^{-1}(L_1 {u_{b,1}}^{-1})$ is a union of languages in $\mathcal{SUM}_i$ over Γ and that $\varphi^{-1}({u_{b,2}}^{-1} L_2)$ is a union of languages in $\mathcal{SUM}_j$ over Γ. Moreover, it is direct to see that no word of $\varphi^{-1}(L_1 {u_{b,1}}^{-1})$ contains the letter b for all $b \in B$. By distributivity of concatenation over union, we finally get that $\varphi^{-1}(L)$ is a union of languages in $\mathcal{SUM}_k$ over Γ.
  This concludes the inductive step and therefore the proof of the lemma. ◀

C Collapse

In this appendix we prove Proposition 12, stating that when M is in $\mathbf{DA_k}$ then any program over M is equivalent to one of length $\mathrm{O}(n^k)$.

Recall that if P is a program over some monoid M of range n, then $P(w)$ denotes the element of M resulting from the execution of the program P on w. It will be convenient here to also work with the word over M resulting from the sequence of executions of each instruction of P on w. We denote this word by $EP(w)$.

The result is a consequence of the following lemma and the fact that for any acceptance set $F \subseteq M$, a word $w \in \Sigma^n$ (where Σ is the input alphabet) is accepted iff $EP(w) \in L$ where L is a language in $\mathcal{SUL}_k$, a Boolean combination of languages in $\mathcal{SUM}_k$.

▶ **Lemma 17.** *Let Σ be a finite alphabet, M a finite monoid, and n, k natural numbers.*

For any program P over M of range n, any set $\Gamma \subseteq M$ and any language K over Γ in $\mathcal{SUM}_k$, there exists a subsequence Q of the sequence of instructions P of length $\mathrm{O}(n^{\max\{k,1\}})$ such that for any subsequence Q' of the sequence of instructions P containing Q as a subsequence, we have for all words w over Σ of length n:

$$EP(w) \in K \Leftrightarrow EQ'(w) \in K \ .$$

Proof. A program P over M of range n is a finite sequence (p_i, f_i) of instructions where each p_i is a natural number which is at most n and each f_i is a function from Σ to M. We denote by l the number of instructions of P. For each set $I \subseteq [l]$ we denote by $P[I]$ the program over M consisting of the subsequence of instructions of P obtained after removing all instructions whose index is not in I. In particular, $P[1, m]$ denotes the initial sequence of instructions of P, until instruction number m.

We prove the lemma by induction on k.

- Inductive step. Let $k \geq 2$ and assume the lemma proved for all $k' < k$. Let n be a natural number, P a program over M of range n and length l, $\Gamma \subseteq M$ and any language K over Γ in $\mathcal{SUM}_k$. By definition, $K = K_1 \gamma K_2$ for some languages $K_1 \in \mathcal{SUM}_i$ and $K_2 \in \mathcal{SUM}_j$ with $i + j = k - 1$. Moreover either γ does not occur in any of the words of K_1 or it does not occur in any of the words of K_2. We only treat the case where γ does not appear in any of the words in K_1. The other case is treated similarly by symmetry.

  For each $p \leq n$ and each $a \in \Sigma$ consider within the sequence of instructions of P the first instruction of the form (p, f) with $f(a) = \gamma$. We let I_γ be the set of indices of these instructions for all a and p. Notice that the size of I is in $\mathrm{O}(n)$.

  For all $i \in I_\gamma$, we let $J_{i,1}$ be the set of indices of the instructions within $P[1, i-1]$ obtained by induction for K_1 and $J_{i,2}$ be the same for $P[i+1, l]$ and K_2.

  We now let I be the union of I_γ and $J_{i,1}$ and $J'_{i,2} = \{j + i \mid j \in J_{i,2}\}$ for all $i \in I_\gamma$. We claim that $P[I]$ has the desired properties.

  First notice that by induction the sizes of $J_{i,1}$ and $J'_{i,2}$ for all $i \in I_\gamma$ are in $\mathrm{O}(n^{\max\{k-1,1\}})$ $= \mathrm{O}(n^{k-1})$ and because the size of I_γ is linear in n, the size of I is in $\mathrm{O}(n^k) = \mathrm{O}(n^{\max\{k,1\}})$ as required.

  Now take $w \in \Sigma^n$.

  Assume now that $EP(w) \in K$. Let i be the position in $EP(w)$ of label γ witnessing the membership in K. Let (p_i, f_i) be the corresponding instruction of P. In particular we have that $f_i(w_{p_i}) = \gamma$. Because γ does not occur in any word of K_1, for all $j < i$ such that $p_j = p_i$ we cannot have $f_j(w_{p_j}) = \gamma$. Hence $i \in I_\gamma$. By induction we have that $EP[1, i-1][J](w) \in K_1$ for any set J containing $J_{i,1}$ and $EP[i+1, l][J](w) \in K_2$ for any set J containing $J_{i,2}$. Hence, if we set $I_1 = \{j \in I \mid j < i\}$ as the subset of I of elements less than i and $I_2 = \{j - i \in I \mid j > i\}$ as the subset of I of elements greater than i translated by $-i$, we have $EP[I](w) = EP[1, i-1][I_1](w)\gamma EP[i+1, l][I_2](w) \in K_1 \gamma K_2 = K$ as desired.

  Assume finally that $EP[I](w) \in K$. Let i be the index in I whose instruction provides the letter γ witnessing the fact that $EP[I](w) \in K$. If $i \in I_\gamma$ we conclude easily by induction. If not this shows that there is an instruction (p_j, f_j) with $j < i$, $j \in I$, $p_j = p_i$ and $f_j(w_{p_j}) = \gamma$. But that would contradict the fact that γ cannot occur in K_1.

- Base case. There are two subcases to consider.
  - $k = 1$. Let n be a natural number, P a program over M of range n and length l, $\Gamma \subseteq M$ and any language K over Γ in $\mathcal{SUM}_1$.

    Then $K = A_1^* \gamma A_2^*$ for some finite alphabets $A_1 \subseteq \Gamma$ and $A_2 \subseteq \Gamma$. Moreover either $\gamma \notin A_1$ or $\gamma \notin A_2$. We only treat the case where γ does not belong to A_1, the other case is treated similarly by symmetry.

    We use the same idea as in the inductive step.

    For each $p \leq n$, each $\alpha \in \Gamma$ and $a \in \Sigma$ consider within the sequence of instructions of P the first and last instruction of the form (p, f) with $f(a) = \alpha$. We let I be the set of indices of these instructions for all a, α and p. Notice that the size of I is in $\mathrm{O}(n) = \mathrm{O}(n^{\max\{k,1\}})$.

We claim that $P[I]$ has the desired properties. Take $w \in \Sigma^n$.
Assume now that $EP(w) \in K$. Let i be the position in $EP(w)$ of label γ witnessing the membership in K. Let (p_i, f_i) be the corresponding instruction of P. In particular we have that $f_i(w_{p_i}) = \gamma$ and this is the γ witnessing the membership in K. Because $\gamma \notin A_1$, for all $j < i$ such that $p_j = p_i$ we cannot have $f_j(w_{p_j}) = \gamma$. Hence $i \in I$. From $EP(w) \in K$ it follows that $EP[I \cap [\![1, i-1]\!]](w) \in A_1$ and $EP[I \cap [\![i+1, l]\!]](w) \in A_2$ showing that $EP[I](w) = EP[I \cap [\![1, i-1]\!]](w)\gamma EP[I \cap [\![i+1, l]\!]](w) \in K$ as desired.
Assume finally that $EP[I](w) \in K$. This means that $EP[I \cap [\![1, i-1]\!]](w) \in A_1^*$ and $EP[I \cap [\![i+1, l]\!]](w) \in A_2^*$. Let i be the index in I whose instruction provides the letter γ witnessing the fact that $EP[I](w) \in K$. If there is an instruction (p_j, f_j), with $j < i$ and $f_j(w_{p_j}) \notin A_1$ then either $j \in I$ and we get a direct contradiction with the fact that $EP[I \cap [\![1, i-1]\!]](w) \in A_1^*$, or $j \notin I$ and we get a smaller $j' \in I$ with the same property, contradicting again the fact that $EP[I \cap [\![1, i-1]\!]](w) \in A_1^*$. Hence for all $j < i$, $f_j(w_{p_j}) \in A_1$. By symmetry we have that for all $j > i$, $f_j(w_{p_j}) \in A_2$, showing that $EP(w) \in A_1^* \gamma A_2^* = K$ as desired.

- $k = 0$. Let n be a natural number, P a program over M of range n and length l, $\Gamma \subseteq M$ and any language K over Γ in $\mathcal{SUM}_0$.
Then $K = A^*$ for some finite alphabet $A \subseteq \Gamma$.
We again use the same idea as before.
For each $p \leq n$, each $\alpha \in \Gamma$ and $a \in \Sigma$ consider within the sequence of instructions of P the first instruction of the form (p, f) with $f(a) = \alpha$. We let I be the set of indices of these instructions for all a, α and p. Notice that the size of I is in $\mathrm{O}(n) = \mathrm{O}(n^{\{k,1\}})$.
We claim that $P[I]$ has the desired properties. Take $w \in \Sigma^n$.
Assume now that $EP(w) \in K$. As $EP[I](w)$ is a subword of $EP(w)$, it follows directly that $EP[I](w) \in A^* = K$ as desired.
Assume finally that $EP[I](w) \in K$. If there is an instruction (p_j, f_j), with $j \in [l]$ and $f_j(w_{p_j}) \notin A$ then either $j \in I$ and we get a direct contradiction with the fact that $EP[I](w) \in A^* = K$, or $j \notin I$ and we get a smaller $j' \in I$ with the same property, contradicting again the fact that $EP[I](w) \in A^* = K$. Hence for all $j \in [l]$, $f_j(w_{p_j}) \in A$, showing that $EP(w) \in A^* = K$ as desired. ◀

Regular Language Distance and Entropy*

Austin J. Parker[1], Kelly B. Yancey[2], and Matthew P. Yancey[3]

1 **Institute for Defense Analyses – Center for Computing Sciences, Bowie, MD, USA**
ajpark2@super.org
2 **University of Maryland, College Park MD, USA, and**
Institute for Defense Analyses - Center for Computing Sciences, Bowie, MD, USA
kbyancey1@gmail.com
3 **Institute for Defense Analyses - Center for Computing Sciences, Bowie, MD, USA**
mpyancey1@gmail.com

Abstract

This paper addresses the problem of determining the distance between two regular languages. It will show how to expand Jaccard distance, which works on finite sets, to potentially-infinite regular languages.

The entropy of a regular language plays a large role in the extension. Much of the paper is spent investigating the entropy of a regular language. This includes addressing issues that have required previous authors to rely on the upper limit of Shannon's traditional formulation of channel capacity, because its limit does not always exist. The paper also includes proposing a new limit based formulation for the entropy of a regular language and proves that formulation to both exist and be equivalent to Shannon's original formulation (when it exists). Additionally, the proposed formulation is shown to equal an analogous but formally quite different notion of topological entropy from Symbolic Dynamics – consequently also showing Shannon's original formulation to be equivalent to topological entropy.

Surprisingly, the natural Jaccard-like entropy distance is trivial in most cases. Instead, the *entropy sum* distance metric is suggested, and shown to be granular in certain situations.

1998 ACM Subject Classification F.4.3 Formal Languages, G.2.2 Graph Theory

Keywords and phrases regular languages, channel capacity, entropy, Jaccard, symbolic dynamics

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.3

1 Introduction

In this paper we study distances between regular expressions. There are many motivations for this analysis. Activities in bioinformatics, copy-detection [9], and network defense sometimes require large numbers of regular expressions be managed. Metrics aid in indexing and management of those regular expressions [4]. Further, understanding the distance between regular languages requires an investigation of the structure of regular languages that we hope eliminates the need for similar theoretical investigations in the future.

A natural definition of the distance between regular languages L_1 and L_2 containing strings of symbols from Σ is: $\lim_{n\to\infty} \frac{|(L_1 \Delta L_2) \cap \Sigma^n|}{|(L_1 \cup L_2) \cap \Sigma^n|}$ (where $L_1 \Delta L_2 = (L_1 \cup L_2) \setminus (L_1 \cap L_2)$ is the symmetric difference). However, the definition has a fundamental flaw: the limit does

* For a full version see [22].

not always exist. Consider the distance between $(aa)^*$ and a^*. When n is even, the fraction is 0, while when n is odd the fraction is 1. Thus, the limit given above is not well defined for those two languages.

This paper addresses that flaw and examines the question of entropy and distance between regular languages in a more general way. A fundamental contribution will be a limit-based distance related to the above that (1) exists, (2) can be computed from the Deterministic Finite Automaton for the associated regular languages, and (3) does not invalidate expectations about the distance between languages.

The core idea is two-fold: (1) to rely on the number of strings *up-to* a given length rather than strings *of* a given length and (2) to use Cesáro averages to smooth out the behavior of the limit. These ideas led us to develop the Cesáro Jaccard distance, which is proven to be well-defined in Theorem 7.

Tied up in this discussion will be the *entropy* of a regular language, which is again a concept whose common definition needs tweaking due to limit-related considerations.

This paper is structured as follows. In Section 2 we discuss related work and define terms that will be used in the paper. Of particular importance is Table 1, which includes all of the distance functions defined in the paper. As the Jaccard distance is a natural entry point into distances between sets, Section 3 will discuss the classical Jaccard distance and how best to extend it to infinite sets. Section 4 will discuss notions of regular language entropy, introducing a new formulation and proving it correct from both a channel capacity and a topological entropy point of view. Section 5 will introduce some distances based on entropy, and show that some of them behave well, while others do not. Finally, Section 6 provides a conclusion and details some potential future work.

2 Background

2.1 Related Work

Chomsky and Miller's seminal paper on regular languages [6] does not address distances between regular languages. It uses Shannon's notion of channel capacity (equation 7 from [6]) for the entropy of a regular language: $h(L) = \lim_{n\to\infty} \frac{\log|L\cap\Sigma^n|}{n}$.

While Shannon says of that limit that "the limit in question will exist as a finite number in most cases of interest" [27], its limit does not always exist for regular languages (consider $(\Sigma^2)^*$). This motivates much of the analysis in this paper. Chomsky and Miller also examine the number of sentences *up to* a given length, foreshadowing some other results in this paper. However, their analysis was based upon an assumption with deeper flaws than that the limit exists. In this paper we address those issues.

Several works since Chomsky and Miller have used this same *of length exactly n* formula to define the entropy of a regular language [3, 9, 17]. These works define entropy as Chomsky and Miller, but add the caveat that they use the upper limit when the limit does not exist. Here we provide foundation for those works by showing the upper limit to be correct (Theorem 13). Further, this paper suggests an equivalent expression for entropy that may be considered more elegant: it is a limit that exists as a finite number for all regular languages which equals the traditional notion of entropy when that limit exists.

Chomsky and Miller's technique was to develop a recursive formula for the number of words accepted by a regular language. That recursive formula comes from the characteristic polynomial of the adjacency matrix for an associated automaton. The eigenvalues of the adjacency matrix describe the growth of the language (we use the same technique, but apply stronger theorems from linear algebra that were discovered several decades after

Chomsky and Miller's work). The recursive formula can also be used to develop a generating function to describe the growth of the language (see [25]). Bodirsky, Gärtner, Oertzen, and Schwinghammer [2] used the generating functions to determine the growth of a regular language over alphabet Σ relative to $|\Sigma|^n$, and Kozik [16] used them to determine the growth of a regular language relative to a second regular language. Our approaches share significant details: they relate the growth of a regular language to the poles of its generating function – which are the zeroes of the corresponding recurrence relation – which are the eigenvalues of the associated adjacency matrix. Our technique establishes the "size" of a regular language independent of a reference alphabet or language.

There is work examining distances between *unary* regular languages, or regular languages on the single character alphabet ($|\Sigma| = 1$) [11]. It introduces a definition for Jaccard distance that will appear in this paper: $1 - \lim_{n\to\infty} \frac{|L_1 \cap L_2 \cap (\bigcup_{i=0}^{n} \Sigma^i)|}{|(L_1 \cup L_2) \cap (\bigcup_{i=0}^{n} \Sigma^i)|}$. Further, it gives a closed form for calculating that distance between two unary regular languages.

Besides the stronger results, our work differs from that of [2, 11, 16] in the analysis of the distance functions presented: in particular, one can conclude (as a consequence of Theorem 17) that the above equation is mostly trivial – it returns 0 or 1 "most" of the time.

More recently, Cui *et al* directly address distances between regular languages using a generalization of Jaccard distance [9]. That paper usefully expands the concept of Jaccard distance to regular languages by (1) using entropy to handle infinite sized regular languages (they use the upper limit notion of entropy described above), and (2) allowing operations other than intersection to be used in the numerator. Further, Cui *et al* suggest and prove properties of several specific distance functions between regular languages. The distance functions in this paper do not generalize the Jaccard distance in the same way, but are proven to be metrics or pseudo-metrics.

Ceccherini-Silberstein *et al* investigate the entropy of specific kinds of subsets of regular languages [3]. They present a novel proof of a known fact from Symbolic Dynamics. They use the same upper limit notion of entropy as above. Other entropy formulations include the number of prefixes of a regular language [5], but this has only been proven equivalent to entropy under restricted circumstances.

Symbolic dynamics [19] studies, among other things, an object called a sofic shift. Sofic shifts are analogous to deterministic finite automata and their shift spaces are related to regular languages. The formulation of entropy used in this field does not suffer from issues of potential non-existence. This paper includes a proof that the *topological entropy* of a sofic shift is equivalent to language-centric formulations in this paper: see Theorem 13.

Other related results from symbolic dynamics include an investigation into the computability of a sofic shift's entropy [28] and a discussion of the lack of relationship between entropy and complexity [18]. There is another proposal for the topological entropy of formal languages [26] that is zero for all regular languages (and hence not helpful as a distance function for regular languages).

A probabilistic automaton is an automaton with a probability distribution applied to outgoing transitions from each state. The words of a regular language thus inherit a probability. Using standard distance functions on probability distributions (such as L_p and Kullback-Leibler divergence), several distance functions [7, 8, 21] have been created for probabilistic languages. Note that in this model, the probability of a word exponentially decreases with its length, and hence these distance functions can be effectively estimated by words of bounded length. Chan [4] also describes several distance functions using only words of bounded length. Our paper will uncover features of several distance functions, which will fit nicely into the above frameworks.

Table 1 The distance functions considered in this paper are listed in this table.

$J'_n(L_1, L_2)$	n Jaccard Distance	$\frac{\lvert W_n(L_1 \triangle L_2)\rvert}{\lvert W_n(L_1 \cup L_2)\rvert}$
$J_n(L_1, L_2)$	$n_\leq$ Jaccard Distance	$\frac{\lvert W_{\leq n}(L_1 \triangle L_2)\rvert}{\lvert W_{\leq n}(L_1 \cup L_2)\rvert}$
$J_C(L_1, L_2)$	Cesàro Jaccard	$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} J_i(L_1, L_2)$
$H(L_1, L_2)$	Entropy Distance	$\frac{h(L_1 \triangle L_2)}{h(L_1 \cup L_2)}$
$H_S(L_1, L_2)$	Entropy Sum Distance	$h(L_1 \cap \overline{L_2}) + h(\overline{L_1} \cap L_2)$

2.2 Definitions and Notation

In this paper Σ will denote a set of *symbols* or the alphabet. *Strings* are concatenations of these symbols. All log operations in this paper will be taken base 2. Raising a string to the power n will represent the string resulting from n concatenations of the original string. A similar notion applies to sets. In this notation, Σ^5 represents all strings of length 5 composed of symbols from Σ. The Kleene star, $*$, when applied to a string (or a set) will represent the set containing strings resulting from any number of concatenations of that string (or of strings in that set), including the empty concatenation. Thus, Σ^* represents all possible strings comprised of symbols in Σ, including the empty string.

A *regular language* is a set $L \subset \Sigma^*$ which can be represented by a *Deterministic Finite Automaton*, DFA for short. A *DFA* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a set of states, Σ is the set of symbols, δ is a partial function from $Q \times \Sigma$ to Q, $q_0 \in Q$ is the *initial state* and $F \subset Q$ is a set of *final states*. A regular language can also be constructed by recursive applications of concatenation (denoted by placing regular expressions adjacent to one another), disjunction (denoted $|$), and Kleene star (denoted $*$), to strings and the empty string. That this construction and the DFA are equivalent is well known [14].

The DFA $(Q, \Sigma, \delta, q_0, F)$ can be thought of as a directed graph whose vertices are Q with edges from q to q' iff there is an $s \in \Sigma$ such that $q' = \delta(q, s)$. The transition function δ provides a labeling of the graph, where each edge (q, q') is labeled by the symbol s such that $\delta(q, s) = q'$. Note that there may be multiple edges between nodes, each with a different label. The adjacency matrix A for a DFA is the adjacency matrix for the corresponding graph. Thus, entries in A are given by $a_{q,q'}$, where $a_{q,q'}$ is the number of edges from vertex q to vertex q'.

For a regular language L, let $W_n(L)$ denote the set of words in L of length exactly n, i.e. $W_n(L) = L \cap \Sigma^n$, and let $W_{\leq n}(L)$ denote the set of words in L of length at most n, i.e. $W_{\leq n}(L) = L \cap (\bigcup_{i=0}^{n} \Sigma^i)$.

Finally, we will discuss when certain distance functions are metrics. A *metric* on the space X is a function $d : X \times X \to \mathbb{R}$ that satisfies

1. $d(x, y) \geq 0$ with equality if and only if $x = y$ for all $x, y \in X$
2. $d(x, y) = d(y, x)$ for all $x, y \in X$
3. $d(x, z) \leq d(x, y) + d(y, z)$ for all $x, y, z \in X$.

An *ultra-metric* is a stronger version of a metric, with the triangle inequality (the third condition above) replaced with the ultra-metric inequality: $d(x, z) \leq \max\{d(x, y), d(y, z)\}$ for all $x, y, z \in X$. Also, there exists a weaker version, called a *pseudo-metric*, which allows $d(x, y) = 0$ when $x \neq y$.

3 Jaccard Distances

The Jaccard distance is a well-known distance function between finite sets. For finite sets A and B, the *Jaccard distance* between them is given by $\frac{|A\triangle B|}{|A\cup B|} = 1 - \frac{|A\cap B|}{|A\cup B|}$ where $A\triangle B$ represents the symmetric difference between the two sets (if $A \cup B = \emptyset$ then the Jaccard distance is 0). This classical Jaccard distance is not defined for infinite sets and as such, is not a suitable distance function for infinite regular languages and will need to be modified.

3.1 Jaccard Distances using W_n and $W_{\leq n}$

A natural method for applying Jaccard distance to regular languages is to fix n, defined as follows:

▶ **Definition 1** (n Jaccard Distance)**.** Suppose L_1 and L_2 are regular languages. Define the n *Jaccard distance* by $J'_n(L_1, L_2) = \frac{|W_n(L_1\triangle L_2)|}{|W_n(L_1\cup L_2)|}$ if $|W_n(L_1 \cup L_2)| > 0$, otherwise $J'_n(L_1, L_2) = 0$.

For fixed n, the above is a pseudo-metric since it is simply the Jaccard distance among sets containing only length n strings. The following proposition points out one deficiency of J'_n.

▶ **Proposition 2.** There exists a set $S = \{L_1, L_2, L_3\}$ of infinite unary regular languages with $L_2, L_3 \subset L_1$ such that for all n there exists an $i \neq j$ such that $J'_n(L_i, L_j) = 0$.

One may also use $W_{\leq n}$ in the definition of a Jaccard-based distance function.

▶ **Definition 3** ($n_\leq$ Jaccard Distance)**.** For regular languages L_1 and L_2, define the $n_\leq$ *Jaccard distance* by $J_n(L_1, L_2) = \frac{|W_{\leq n}(L_1\triangle L_2)|}{|W_{\leq n}(L_1\cup L_2)|}$ if $|W_{\leq n}(L_1 \cup L_2)| > 0$, otherwise $J_n(L_1, L_2) = 0$.

The issue with J'_n pointed out by Proposition 2 can be proven to not be a problem for J_n: see the first point of Theorem 4. On the other hand, the second point of Theorem 4 shows that no universal n exists.

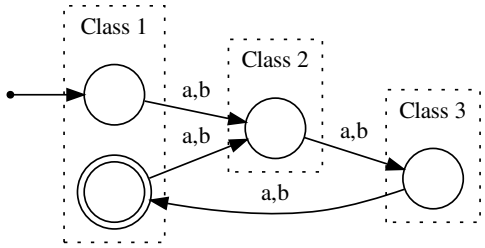
▶ **Theorem 4.** *The function J_n defined above is a pseudo-metric and satisfies the following:*

1. *Let $S = \{L_1, \ldots, L_k\}$ be a set of regular languages. There exists an n such that J_n is a metric over S. Moreover, we may choose n such that $n \leq \max_{i,j}(s(L_i)+1)(s(L_j)+1)-1$ where $s(L_i)$ represents the number of states in the minimal DFA corresponding to L_i.*
2. *For any fixed n there exist regular languages L, L' with $L \neq L'$ such that $J_n(L, L') = 0$.*

For any pseudo-metric, the relation $d(x, y) = 0$ is an equivalence relation. Thus, if we mod out by this equivalence relation, the pseudo-metric becomes a metric.

Due to the fact that one must choose a fixed n, J_n and J'_n cannot account for the infinite nature of regular languages. Limits based on J_n and J'_n are a natural next step. However, the natural limits involving J'_n and J_n do not always exist. An example showing this was given for J'_n in the beginning of the introduction (Section 1). A similar example applies to J_n. Consider the languages given by $L_1 = (a|b)^*$ and $L_2 = ((a|b)^2)^*$ ($\Sigma = \{a, b\}$). For these languages, $\lim_{n\to\infty} J_{2n}(L_1, L_2) = 2/3$ and $\lim_{n\to\infty} J_{2n+1}(L_1, L_2) = 1/3$. Hence, $\lim_{n\to\infty} J_n(L_1, L_2)$ does not exist.

The next theorem gives conditions for when the limit of J'_n exists as n goes to infinity. Before the theorem is stated we will need some more terminology. Suppose L is a regular language and M is the corresponding DFA. This DFA is a labeled directed graph. An *irreducible component* of M is a strongly connected component of the graph. That is, an



$$A^i = \begin{cases} \begin{pmatrix} 0 & 2^i & 0 & 0 \\ 0 & 0 & 2^i & 0 \\ 0 & 0 & 0 & 2^i \\ 0 & 2^i & 0 & 0 \end{pmatrix} & \text{if } i \equiv 1 \bmod 3 \\ \begin{pmatrix} 0 & 0 & 2^i & 0 \\ 0 & 0 & 0 & 2^i \\ 0 & 2^i & 0 & 0 \\ 0 & 0 & 2^i & 0 \end{pmatrix} & \text{if } i \equiv 2 \bmod 3 \\ \begin{pmatrix} 0 & 0 & 2^i & 0 \\ 0 & 2^i & 0 & 0 \\ 0 & 0 & 2^i & 0 \\ 0 & 0 & 0 & 2^i \end{pmatrix} & \text{if } i \equiv 0 \bmod 3 \end{cases}$$

Figure 1 The DFA for a period 3 language and the associated adjacency matrix raised to the i^{th} power.

irreducible component is composed of a maximal set of vertices such that for any pair, there is a directed path between them.

The *period* of an irreducible graph (or associated adjacency matrix) is the largest integer p such that the vertices can be grouped into classes $Q_0, Q_1, \ldots, Q_{p-1}$ such that if $x \in Q_i$, then all of the out neighbors of x are in Q_j, where $j = i + 1(mod\ p)$. The period of a reducible graph is the least common multiple of the periods of its irreducible components. See Figure 1 for an example of a regular language whose DFA has period 3. For a more formal definition of periodicity see [19]. If the graph (or matrix) has period 1 it will be called *aperiodic*. Matrices that are irreducible and aperiodic are called *primitive*. The definition of primitive presented here is equivalent to the condition that there is an n such that all entries of the adjacency matrix A raised to the n-th power (A^n) are positive [20]. This is illustrated in Figure 1, where the graph is periodic and reducible and all powers of that matrix contain multiple zeroes.

▶ **Theorem 5.** *Suppose L_1 and L_2 are regular languages. If each irreducible component of the DFA associated to $L_1 \triangle L_2$ and $L_1 \cup L_2$ are aperiodic, then $\lim_{n\to\infty} J'_n(L_1, L_2)$ converges.*

Let us build intuition prior to proving Theorem 5, which will also frame the question of convergence in the next subsection. We will first discuss Theorem 5 in the case where the DFA associated to regular languages $L_1 \triangle L_2$ and $L_1 \cup L_2$ are primitive. Suppose $A_\triangle$ and $A_\cup$ are the adjacency matrices for $L_1 \triangle L_2$ and $L_1 \cup L_2$ respectively. Perron-Frobenius theory tells us that the eigenvalue of largest modulus of a primitive matrix is real and unique. Let $(v_\triangle, \lambda_\triangle)$ and $(v_\cup, \lambda_\cup)$ be eigenpairs composed of the top eigenvalues for $A_\triangle$ and $A_\cup$ respectively. Notice that $i_\triangle A^n_\triangle f_\triangle$, where $i_\triangle$ is the row vector whose jth entry is 1 if j is an initial state in $A_\triangle$ and 0 otherwise (a similar definition for final states defining column vector $f_\triangle$ holds), represents words in $L_1 \triangle L_2$ of length n. If we write $f_\triangle = c_1 v_\triangle + c_2 w$ and $f_\cup = d_1 v_\cup + d_2 y$, then $i_\triangle A^n_\triangle f_\triangle$ converges to $\lambda^n_\triangle c_1 i_\triangle v_\triangle$, and $i_\cup A^n_\cup f_\cup$ converges to $\lambda^n_\cup d_1 i_\cup v_\cup$ as n goes to infinity. This convergence is guaranteed because $\lambda_\cup$ and $\lambda_\triangle$ are unique top eigenvalues. Thus,

$$\lim_{n\to\infty} J'_n(L_1, L_2) = \lim_{n\to\infty} \left(\frac{\lambda_\triangle}{\lambda_\cup}\right)^n \frac{c_1 i_\triangle v_\triangle}{d_1 i_\cup v_\cup}$$

and the limit converges ($\lambda_\triangle \leq \lambda_\cup$ because $L_1 \triangle L_2 \subseteq L_1 \cup L_2$).

The general case of Theorem 5, which does not assume $L_1 \triangle L_2$ and $L_1 \cup L_2$ have irreducible matrices, is more complicated. However, the outline of the argument is the same, and we will sketch it here. The key difference is the use of newer results. An understanding of the asymptotic behavior of A^n for large n was finally beginning to be developed several decades after Chomsky and Miller investigated regular languages. In 1981 Rothblum [23] proved that for each non-negative matrix A with largest eigenvalue λ, there exists $q \geq 1$ (which happens to be the period of A) and polynomials $S_0(x), S_1(x), \ldots, S_{q-1}(x)$ (whose domain is the set of real numbers and whose coefficients are matrices) such that for all whole numbers $0 \leq k \leq q-1$ we have that $\lim_{n\to\infty} (A/\lambda)^{qn+k} - S_k(n) = 0$. We will refer to this result later in the paper, where we will simply call it **Rothblum's Theorem** (a slow treatment of this theory with examples can be found in [24]). So the rest of the proof to Theorem 5 is observing that $q = 1$ in the case we are interested in, and so $\lim_{n\to\infty} J'_n(L_1, L_2)$ converges.

3.2 Cesàro Jaccard

For a sequence of numbers $a_1, a_2, \ldots$, a Cesàro summation is $\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} a_i$ when the limit exists. The intuition behind a Cesàro summation is that it may give the "average value" of the limit of the sequence, even when the sequence does not converge. For example, the sequence $a_j = e^{\alpha ij}$ (where $i^2 = -1$) has Cesàro summation 0 for all real numbers $\alpha \neq 0$. This follows from the fact that rotations of the circle are uniquely ergodic [13]. Not all sequences have a Cesàro summation, even when we restrict our attention to sequences whose values lie in $[0, 1]$. For example, the sequence b_i, where $b_i = 1$ when $2^{2n} < i < 2^{2n+1}$ for some $n \in \mathbb{N}$ and $b_i = 0$ otherwise has no Cesàro summation. However, we will be able to show that the Cesàro average of Jaccard distances does exist.

To that end, another limit based distance is the Cesàro average of the J_n or J'_n.

▶ **Definition 6** (Cesàro Jaccard Distance)**.** Suppose L_1 and L_2 are regular languages. Define the *Cesàro Jaccard distance* by $J_C(L_1, L_2) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} J_i(L_1, L_2)$.

The Cesàro Jaccard distance is theoretically better than the above suggestions in Section 3.1 since it can be shown to exist for all regular languages.

▶ **Theorem 7.** *Let L_1 and L_2 be two regular languages. Then, $J_C(L_1, L_2)$ is well-defined. That is, $\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} J_i(L_1, L_2)$ exists.*

We will breifly sketch the proof to Theorem 7. Recall that $|W_n(L_1 \triangle L_2)|$ and $|W_n(L_1 \cup L_2)|$ can be calculated using powers of specific matrices. If we take Q to be the least common multiple of the period from each of the matrices associated with $|W_n(L_1 \triangle L_2)|$ and $|W_n(L_1 \cup L_2)|$, we can immediately see that $\lim_{n\to\infty} J'_{Qn+k}(L_1, L_2)$ exists, via Rothblum's Theorem. Moreover, it will equal zero if they have different values for the largest eigenvalue or the degree of $S_k(x)$. But if they have the same value for the largest eigenvalue and degree of $S_k(x)$, then $\lim_{n\to\infty} J'_{Qn+k}(L_1, L_2)$ will be the ratio of the leading coefficients of the polynomials $S_k(x)$ for the two matrices. The proof finishes by observing that $J'_C(L_1, L_2) = \frac{1}{n} \sum_{i=1}^{n} J'_i(L_1, L_2)$ will be the average of these values.

We will require a new result to show that the more interesting value $J_C(L_1, L_2)$ is well-defined (part (2) of the theorem is similar to a result in [23]).

▶ **Theorem 8.** *Let A be the adjacency matrix for a DFA representing a regular language L, and let λ be the largest eigenvalue of A. Let q and $S_0(x), S_1(x), \ldots, S_{q-1}(x)$ be as in Rothblum's theorem; let d be the largest degree of the polynomials $S_0(x), S_1(x), \ldots, S_{q-1}(x)$. Let $s_\ell = \lim_{n\to\infty} n^{-(d+1)} \sum_{i=1}^{n} S_\ell(i)$ and $t_\ell = \lim_{n\to\infty} n^{-d} S_\ell(n)$.*

1. *If* $\lambda < 1$*, then* L *is finite.*
2. *If* $\lambda = 1$*, then* $\lim_{n\to\infty} \frac{1}{n^{d+1}} \sum_{i=1}^{n} A^i = \sum_{i=0}^{q-1} s_\ell$.
3. *If* $\lambda > 1$*, then* $\lim_{n\to\infty} \frac{1}{(qn+k)^d} \lambda^{-(qn+k)} \sum_{i=1}^{qn+k} A^i = \frac{1}{1-\lambda^{-q}} \sum_{\ell=k-q+1}^{k} \lambda^{\ell-k} t_\ell$ *where the indices of the* t_i *are taken modulo* q.

Using our new result in place of Rothblum's theorem, we now see that $J_C(L_1, L_2)$ is well-defined. Note that in $J'_C(L_1, L_2)$ each congruence class k is handled independently and the final answer is the average of such results. On the other hand, in $J_C(L_1, L_2)$ each congruence class k has a limit that is a combination of results from all of the congruence classes. Thus the total answer is dominated by the overall asymptotic behavior and not just small periodic undercurrents. We illustrate this point via the next example.

▶ **Example 9.** Let $L_1 = ((a|b)^2)^*|c^*$ and $L_2 = ((a|b)^2)^*|d^*$. The languages L_1 and L_2 have $((a|b)^2)^*$ in common and so mutually shared words up to length n grow exponentially. The languages disagree on c^* and d^*, whose words only grow polynomially. Hence, L_1 and L_2 are very similar and should have a small distance. However, J'_C gives equal weight to words of even length and odd length, even though the languages are mostly made up of even-length words.

Rigorously, we have that $\lim_{n\to\infty} J_{2n}(L_1, L_2) = 0$ and $\lim_{n\to\infty} J'_{2n}(L_1, L_2) = 0$. Furthermore, $\lim_{n\to\infty} J_{2n+1}(L_1, L_2) = 0$ and $\lim_{n\to\infty} J'_{2n+1}(L_1, L_2) = 1$. Thus, $J_C(L_1, L_2) = 0$, while $J'_C(L_1, L_2) = \frac{1}{2}$.

We conclude this section with a fact about the Cesáro Jaccard distance.

▶ **Fact 10.** The Cesàro Jaccard distance inherits the pseudo-metric property from J_n.

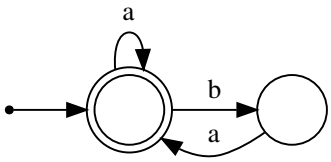
4 Entropy

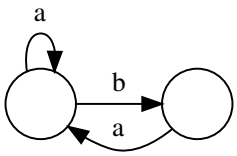
In this section we develop the idea of topological entropy for a certain type of dynamical system and show how it relates to a quantity that we have identified as the language entropy. Then, we will show how Cesáro Jaccard is related to entropy.

4.1 Topological Entropy

Topological entropy is a concept from dynamical systems where the space is a compact metric space and the map defined there is continuous [19]. In dynamics, successive applications of the map are applied and the long term behavior of the system is studied. An orbit of a point x for the map T is the set $\{T^n(x) \ : \ n \in \mathbb{Z}\}$. Topological entropy is an abstract concept meant to determine the exponential growth of distinguishable orbits of the dynamical system up to arbitrary scale. A positive quantity for topological entropy reflects chaos in the system [1]. This concept was motivated by Kolmogorov and Sinai's theory of measure-theoretic entropy in ergodic theory [15, 29], which in turn is related to Shannon entropy [27]. An example of a topological dynamical system is a sofic shift, which is a symbolic system that is intricately related to DFA. Instead of defining the topological entropy of a sofic shift symbolically, which is classical, we will use the graph theoretic description.

A *sofic shift* can be thought of as the space of biinfinite walks (i.e. walks with no beginning and no end) on a right-solving labeled directed graph (a right-solving labeled graph has a unique label for each edge leaving a given node). Suppose G is a directed graph where V is the set of vertices and E is the set of edges of G. Furthermore, suppose that every edge in E is labeled with a symbol from Σ, and that there is at most one outgoing edge from each





aaa, *aaba*, *ba*, *aaaaaaa*, *aaabaabaa*, ... *aabaabaabaaaabaaaaaabaaaaab* ...

Figure 2 A DFA with some accepted strings and a sofic shift with a portion of a derived biinfinite string.

vertex with a given label (i.e. *right-solving*). Note that this construction is similar to a DFA, however there are no initial and final states. A biinfinite walk on G with a specified base vertex is an infinite walk in both directions (forward and backward) from the base vertex on the graph. This biinfinite walk corresponds to a biinfinite string of symbols from Σ. See Figure 2.

We will call a finite block of symbols *admissible* if there is a biinfinite string of symbols corresponding to a biinfinite walk on G and this finite block appears somewhere within the biinfinite string. Note that all sufficiently long words in the DFA's language will contain a substring of almost the same length that is an admissible block, while not all admissible blocks will be in the associated DFA's language. Denote the set of admissible blocks of length n corresponding to G by $B_n(G)$. The *topological entropy* of the sofic shift represented by the right-solving labeled graph G is denoted by $h_t(G)$ and is defined by

$$h_t(G) = \lim_{n\to\infty} \frac{\log |B_n(G)|}{n}.$$

Using Perron-Frobenius theory it has been proven that the topological entropy of a sofic shift represented by a right-solving labeled graph G is equal to the log base 2 of the spectral radius of the adjacency matrix of G [19]. That is, the topological entropy is given by the log of the adjacency matrix's largest modulus eigenvalue. Algorithms for computing eigenvalues are well known and run in time polynomial in the width of the matrix [12].

As you can see, sofic shifts are very similar to DFA. Given a DFA, M, one can construct a sofic shift by thinking of M as a labeled directed graph and creating the *trim graph* by removing all states that are not part of an accepting path. Information regarding initial and final states is no longer needed. Note that the graph M is naturally right-solving because of the determinism of DFA. It is also easiest to remove from M all vertices that do not have both an outgoing and incoming edge (since we are now interested in biinfinite walks). The resulting graph is called the *essential graph*. At this point one is free to apply the above definition and compute the topological entropy of the sofic shift corresponding to the DFA. This quantity can be computed by analyzing the irreducible components.

▶ **Theorem 11** ([19]). *Suppose that G is the labeled directed graph associated to a sofic shift. If $G_1, \ldots, G_k$ are the irreducible components of G, then $h_t(G) = \max_{1\le i\le k} h_t(G_i)$.*

In the next subsection we will introduce the language entropy and show that it is the same as the topological entropy of the sofic shift corresponding to a DFA.

4.2 Language Entropy

Traditionally, the entropy of a regular language L (also called the *channel capacity* [6] or *information rate* [10]) is defined as $\limsup_{n\to\infty} \frac{\log|W_n(L)|}{n}$. This limit may not exist and so an upper limit is necessary. We will show that this upper limit is realized by the topological entropy of the corresponding sofic shift and define another notion of language entropy, which is preferable since an upper limit is not necessary.

▶ **Definition 12** (Language Entropy)**.** Given a regular language L define the *language entropy* by $h(L) = \lim_{n\to\infty} \frac{\log|W_{\leq n}(L)|}{n}$.

▶ **Theorem 13.** *Let L be a non-empty regular language over the set of symbols Σ, and let G be the labeled directed graph of the associated sofic shift. We have that*

$$\limsup_{n\to\infty} \frac{\log|W_n(L)|}{n} = h_t(G).$$

Moreover, for a fixed language L there exists a constant c such that there is an increasing sequence of integers n_i satisfying $0 < n_{i+1} - n_i \leq c$ and

$$\lim_{i\to\infty} \frac{\log|W_{n_i}(L)|}{n_i} = h_t(G).$$

As a corollary to this theorem we obtain an important statement regarding the connection between topological entropy (from dynamical systems) and language entropy (similar to Shannon's channel capacity). The following statement is consistent with remarks made by Chomsky and Miller [6] that involved undefined assumptions; we show rigorously that this formula is correct for all DFA.

▶ **Corollary 14.** *Let L be a non-empty regular language over the set of symbols Σ, and let G be the labeled directed graph of the associated sofic shift. Then,*

$$h(L) = \lim_{n\to\infty} \frac{\log|W_{\leq n}(L)|}{n} = h_t(G).$$

There are some simple properties of language entropy which will be useful later. The first is a simple re-phrasing of Corollary 14.

▶ **Lemma 15.** *For any regular language L, we have that $|W_{\leq n}(L)| = 2^{n(h(L)+o(1))}$.*

▶ **Lemma 16.** *Suppose L_1 and L_2 are regular languages over Σ. The following hold:*

1. *If $L_1 \subseteq L_2$, then $h(L_1) \leq h(L_2)$.*
2. *$h(L_1 \cup L_2) = max(h(L_1), h(L_2))$*
3. *$\max(h(L_1), h(\overline{L_1})) = \log|\Sigma|$*
4. *If $h(L_1) < h(L_2)$, then $h(L_2 \setminus L_1) = h(L_2)$.*
5. *If L_1 is finite, then $h(L_1) = 0$.*

4.3 Relationship between Entropy and Cesáro Jaccard

In Section 3.2 we proved that the Cesàro Jaccard distance is well-defined. As you will see, Cesáro Jaccard and entropy are mostly disjoint in what they measure.

▶ **Theorem 17.** *Let L_1, L_2 be two regular languages.*

1. *If $h(L_1 \triangle L_2) \neq h(L_1 \cup L_2)$, then $J_C(L_1, L_2) = 0$.*
2. *If $h(L_1 \cap L_2) \neq h(L_1 \cup L_2)$, then $J_C(L_1, L_2) = 1$.*
3. *If $0 < J_C(L_1, L_2) < 1$, then the following equal each other:*
   $h(L_1)$, $h(L_2)$, $h(L_1 \cap L_2)$, $h(L_1 \triangle L_2)$, $h(L_1 \cup L_2)$.

To better understand this theorem, consider the following examples corresponding to the three cases of the theorem: (1) let $L_1 = ((a|b)^2)^*|c^*$ and $L_2 = ((a|b)^2)^*|d^*$ as in Example 9, (2) let $L_1 = (a|b)^*|c^*$ and $L_2 = (d|e)^*|c^*$, and (3) let $L_1 = (aa)^*$ and $L_2 = a^*$ as in the Introduction.

5 Entropy Distances

Entropy provides a natural method for dealing with the infinite nature of regular languages. Because it is related to the eigenvalues of the regular language's DFA, it is computable in polynomial time given a DFA for the language. Note that the DFA does not have to be minimal. We can therefore compute the entropy of set-theoretic combinations of regular languages (intersection, disjoint union, etc) and use those values to determine a distance between the languages.

5.1 Entropy Distance

A natural Jaccard-esque distance function based on entropy is the entropy distance.

▶ **Definition 18** (Entropy Distance)**.** Suppose L_1 and L_2 are regular languages. Define the *entropy distance* to be $H(L_1, L_2) = \frac{h(L_1 \triangle L_2)}{h(L_1 \cup L_2)}$ if $h(L_1 \cup L_2) > 0$, otherwise $H(L_1, L_2) = 0$.

This turns out to be equivalent to a Jaccard limit with added log operations:

▶ **Corollary 19.** *Suppose L_1 and L_2 are regular languages. The following relation holds:*

$$\lim_{n\to\infty} \frac{\log |W_{\leq n}(L_1 \triangle L_2)|}{\log |W_{\leq n}(L_1 \cup L_2)|} = H(L_1, L_2).$$

Note that H is not always a good candidate for a distance function as it only produces non-trivial results for languages that have the same entropy.

▶ **Proposition 20.** Suppose L_1 and L_2 are regular languages. If $h(L_1) \neq h(L_2)$, then $H(L_1, L_2) = 1$.

As further evidence that H is not a good candidate for a distance function, we show it is an ultra-pseudo-metric. The ultra-metric condition, i.e. $d(x, z) \leq \max(d(x, y), d(y, z))$, is so strong that it can make it difficult for the differences encoded in the metric to be meaningful for practical applications.

▶ **Theorem 21.** *The function H is an ultra-pseudo-metric.*

5.2 Entropy Sum

In this subsection we will define a new (and natural) distance function for infinite regular languages. We call this distance function the *entropy sum distance*. We will prove that not only is this distance function a pseudo-metric, it is also sometimes granular. Granularity lends insight into the quality of a metric. Intuitively, granularity means that for any two points in the space, you can find a point between them. A metric d on the space X is *granular* if for every two points $x, z \in X$, there exists $y \in X$ such that $d(x, y) < d(x, z)$ and $d(y, z) < d(x, z)$, i.e. $d(x, z) > \max(d(x, y), d(y, z))$.

▶ **Definition 22** (Entropy Sum Distance)**.** Suppose L_1 and L_2 are regular languages. Define the *entropy sum distance* to be $H_S(L_1, L_2) = h(L_1 \cap \overline{L_2}) + h(\overline{L_1} \cap L_2)$.

The entropy sum distance was inspired by first considering the entropy of the symmetric difference directly, i.e. $h(L_1 \triangle L_2)$. However, since entropy measures the entropy of the most complex component (Theorem 11), more information is gathered by using a sum as above in the definition of entropy sum.

▶ **Theorem 23.** *The function H_S is a pseudo-metric.*

The next two propositions display when granularity is achieved and when it is not.

▶ **Proposition 24.** Let L_1 and L_2 be regular languages such that $h(L_1 \cap \overline{L_2}), h(\overline{L_1} \cap L_2) > 0$. Then, there exists two regular languages $R_1 \neq R_2$ such that $H_S(L_1, L_2) > \max(H_S(L_1, R_i), H_S(R_i, L_2))$ for each i.

▶ **Proposition 25.** Let L_1 and L_2 be regular languages such that $h(\overline{L_1} \cap L_2) = 0$. For all regular languages L we have that $H_S(L_1, L_2) \leq \max(H_S(L_1, L), H_S(L, L_2))$.

6 Conclusion and Future Work

This paper has covered some issues related to the entropy of regular languages and the distance between regular languages. It has proven correct the common upper limit formulation of language entropy and has provided a limit based entropy formula that can be shown to exist. Jaccard distance was shown to be related to language entropy, and various limit based extensions of the Jaccard distance were shown to exist or not exist. The natural entropy based distance function was shown to be an ultra-pseudo-metric, and some facts were proven about the function that show it likely to be impractical. Finally, the paper introduces an entropy-based distance function and proves that function to be a pseudo-metric, as well as granular under certain conditions.

In this paper several formulations of entropy are developed, and it is natural to consider which would be the best to use. In a practical sense it does not matter since all formulations are equivalent (Theorem 13) and can be computed using Shannon's determinant-based method [27]. However, conceptually, it can be argued that $\lim_{n\to\infty} \frac{log|W_{\leq n}(L)|}{n}$ is the preferable formulation. First, there is a notational argument that prefers using limits that exist. This is a limit that exists (Corollary 14), whereas many other limit formulations do not. Second, this limit captures more readily the concept of "number of bits per symbol" that Shannon intended. Because regular languages can have strings with staggered lengths, using W_n forces the consideration of possibly empty sets of strings of a given length. This creates dissonance when the language has non-zero entropy. Instead, the monotonically growing $W_{\leq n}$ more clearly encodes the intuition that the formulation is expressing the number of bits needed to express the next symbol among all words in the language.

Apart from expanding to consider context-free languages and other languages ([10]), one investigation that is absent from this paper is the determination of similarity between languages that are disjoint but obviously similar (i.e. aa^* and ba^*). A framework for addressing such problems is provided in [9], but finding metrics capturing such similarities can be fodder for future efforts.

References

1 F. Blanchard, E. Glasner, S. Kolyada, and A. Maass. On Li-Yorke pairs. *J. Reine Angew. Math.*, 547:51–68, 2002.

2 M. Bodirsky, T. Gärtner, T. von Oertzen, and J. Schwinghammer. Efficiently computing the density of regular languages. In *LATIN 2004: Theoretical informatics*, volume 2976 of *Lecture Notes in Comput. Sci.*, pages 262–270. Springer, Berlin, 2004. doi:10.1007/978-3-540-24698-5_30.

3 T. Ceccherini-Silberstein, A. Machì, and F. Scarabotti. On the entropy of regular languages. *Theoretical computer science*, 307(1):93–102, 2003.

4 C. Chan, M. Garofalakis, and R. Rastogi. Re-tree: an efficient index structure for regular expressions. *The VLDB Journal—The International Journal on Very Large Data Bases*, 12(2):102–119, 2003.

5 C. Chang. Algorithm for the complexity of finite automata. *31st Workshop on Combinatorial Mathematics and Computation Theory*, pages 216–220, 2014.

6 N. Chomsky and G. Miller. Finite state languages. *Information and Control*, 1(2):91–112, 1958. doi:10.1016/S0019-9958(58)90082-2.

7 C. Cortes, M. Mohri, and A. Rastogi. On the computation of some standard distances between probabilistic automata. In *Implementation and application of automata*, volume 4094 of *Lecture Notes in Comput. Sci.*, pages 137–149. Springer, Berlin, 2006. doi:10.1007/11812128_14.

8 C. Cortes, M. Mohri, A. Rastogi, and M. Riley. Efficient computation of the relative entropy of probabilistic automata. In *LATIN 2006: Theoretical informatics*, volume 3887 of *Lecture Notes in Comput. Sci.*, pages 323–336. Springer, Berlin, 2006. doi:10.1007/11682462_32.

9 C. Cui, Z. Dang, T. Fischer, and O. Ibarra. Similarity in languages and programs. *Theoretical Computer Science*, 498:58–75, 2013.

10 C. Cui, Z. Dang, T. Fischer, and O. Ibarra. Information rate of some classes of non-regular languages: an automata-theoretic approach (extended abstract). In *Mathematical foundations of computer science 2014. Part I*, volume 86343 of *Lecture notes in Comput. Sci.*, pages 232–243. Springer, Heidelberg, 2014.

11 Jürgen Dassow, Gema M. Martín, and Francisco J. Vico. A similarity measure for cyclic unary regular languages. *Fundam. Inform.*, 96(1-2):71–88, 2009. doi:10.3233/FI-2009-168.

12 J. Francis. The QR transformation a unitary analogue to the LR transformation — part 1. *The Computer Journal*, 4(3):265–271, 1961.

13 B. Hasselblatt and A. Katok. *A first course in dynamics: With a panorama of recent developments*. Cambridge University Press, New York, 2003.

14 J. Hopcroft and J. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley Publishing Company, Inc., 1979.

15 A. Kolmogorov. Entropy per unit time as a metric invariant of automorphisms. In *Dokl. Akad. Nauk SSSR*, volume 124, pages 754–755, 1959.

16 J. Kozik. Conditional densities of regular languages. In *Proceedings of the Second Workshop on Computational Logic and Applications (CLA 2004)*, volume 140 of *Electron. Notes Theor. Comput. Sci.*, pages 67–79 (electronic). Elsevier, Amsterdam, 2005. doi:10.1016/j.entcs.2005.06.023.

17 W. Kuich. On the entropy of context-free languages. *Information and Control*, 16(2):173–200, 1970.

18 W. Li. On the relationship between complexity and entropy for Markov chains and regular languages. *Complex systems*, 5(4):381–399, 1991.

19 D. Lind and B. Marcus. *Symbolic dynamics and coding*. Cambridge, 1995.

20 J. Marklof and C. Ulcigrai. Lecture notes for dynamical systems and ergodic theory, 2015-2016. http://www.maths.bris.ac.uk/∼majm/DSET/index.html.

21 M-J. Nederhof and G. Satta. Computation of distances for regular and context-free probabilistic languages. *Theoret. Comput. Sci.*, 395(2-3):235–254, 2008. doi:10.1016/j.tcs.2008.01.010.

22 A. Parker, K. Yancey, and M. Yancey. Regular language distance and entropy. *arXiv*, 602.07715, 2015.

23 U. Rothblum. Expansion of sums of matrix powers. *SIAM Review*, 23:143–164, 1981.

24 U. Rothblum. Chapter 9, nonnegative matrices and stochastic matrices. In *Handbook of Linear Algebra*. (eds: L. Hogben), Chapman and Hall / CRC, 2007.

25 Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science. Springer, 1978. doi:10.1007/978-1-4612-6264-0.

26 F. Schneider and D. Borchmann. Topological entropy of formal languages. *arXiv*, 1507.03393, 2015.

27 C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

28 J. Simonsen. On the computability of the topological entropy of subshifts. *Discrete mathematics and theoretical computer science*, 8(1):83–95, 2006.

29 Y. Sinai. On the notion of entropy of a dynamical system. In *Dokl Akad Nauk SSSR*, volume 124, pages 768–771, 1959.

The Complexity of Boolean Surjective General-Valued CSPs*

Peter Fulla[1] **and Stanislav Živný**[2]

1 **Department of Computer Science, University of Oxford, UK**
peter.fulla@cs.ox.ac.uk
2 **Department of Computer Science, University of Oxford, UK**
standa.zivny@cs.ox.ac.uk

Abstract

Valued constraint satisfaction problems (VCSPs) are discrete optimisation problems with a $\overline{\mathbb{Q}}$-valued objective function given as a sum of fixed-arity functions, where $\overline{\mathbb{Q}} = \mathbb{Q} \cup \{\infty\}$ is the set of extended rationals.

In Boolean surjective VCSPs variables take on labels from $D = \{0, 1\}$ and an optimal assignment is required to use both labels from D. A classic example is the *global min-cut* problem in graphs. Building on the work of Uppman, we establish a dichotomy theorem and thus give a complete complexity classification of Boolean surjective VCSPs. The newly discovered tractable case has an interesting structure related to projections of downsets and upsets. Our work generalises the dichotomy for $\{0, \infty\}$-valued constraint languages (corresponding to CSPs) obtained by Creignou and Hébrard, and the dichotomy for $\{0, 1\}$-valued constraint languages (corresponding to Min-CSPs) obtained by Uppman.

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

Keywords and phrases constraint satisfaction problems, surjective CSP, valued CSP, min-cut, polymorphisms, multimorphisms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.4

1 Introduction

The (s, t)-Min-Cut problem asks, given a digraph $G = (V, E)$ with source $s \in V$, sink $t \in V$, and edge weights $w : E \to \mathbb{Q}_{>0}$, for a subset $C \subseteq V$ with $s \in C$ and $t \notin C$ minimising $w(C) = \sum_{(u,v)\in E, u\in C, v\notin C} w(u, v)$ [22]. This fundamental problem is an example of a Boolean valued constraint satisfaction problem (VCSP).

Let D be an arbitrary finite set called the *domain*. A *valued constraint language*, or just a language, Γ is a set of weighted relations; each weighted relation $\gamma \in \Gamma$ is a function $\gamma : D^{\mathrm{ar}(\gamma)} \to \overline{\mathbb{Q}}$, where $\mathrm{ar}(\gamma) \in \mathbb{N}$ is the *arity* of γ and $\overline{\mathbb{Q}} = \mathbb{Q} \cup \{\infty\}$ is the set of extended rationals. If $|D| = 2$ then Γ is called a *Boolean* language. An *instance* $I = (V, D, \phi_I)$ of the VCSP on domain D is given by a finite set of n variables $V = \{x_1, \ldots, x_n\}$ and an objective function $\phi_I : D^n \to \overline{\mathbb{Q}}$ expressed as a weighted sum of *valued constraints* over V, i.e. $\phi_I(x_1, \ldots, x_n) = \sum_{i=1}^{q} w_i \cdot \gamma_i(\mathbf{x}_i)$, where γ_i is a weighted relation, $w_i \in \mathbb{Q}_{\geq 0}$ is the *weight*

* The authors were supported by a Royal Society Research Grant. Stanislav Živný was supported by a Royal Society University Research Fellowship. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

and $\mathbf{x}_i \in V^{\mathrm{ar}(\gamma_i)}$ the *scope* of the ith valued constraint. (We note that we allow zero weights and for $w_i = 0$ we define $w_i \cdot \infty = \infty$.) Given an instance I, the goal is to find an assignment $s : V \to D$ of domain labels to the variables that *minimises* ϕ_I. Given a language Γ, we denote by VCSP(Γ) the class of all instances I that use only weighted relations from Γ in their objective function. Valued CSPS are also called general-valued CSPs to emphasise that (decision) CSPs are a special case of valued CSPs.

To continue with the (s,t)-Min-Cut example, let $D = \{0,1\}$. We will use the following three weighted relations: $\gamma : D^2 \to \overline{\mathbb{Q}}$ is defined by $\gamma(x,y) = 1$ if $x = 0$ and $y = 1$, and $\gamma(x,y) = 0$ otherwise; $\rho_d : D \to \overline{\mathbb{Q}}$, for $d \in D$, is defined by $\rho_d(x) = 0$ if $x = d$ and $\rho_d(x) = \infty$ if $x \neq d$. Now, given an (s,t)-Min-Cut instance $G = (V,E)$, $s,t \in V = \{x_1,\ldots,x_n\}$, and $w : E \to \mathbb{Q}_{>0}$ as before, the problem of finding an optimal (s,t)-Min-Cut in G is equivalent to solving the following instance of VCSP(Γ_{cut}), where $\Gamma_{\mathsf{cut}} = \{\gamma, \rho_0, \rho_1\}$: $I = (V, D, \phi_I)$ and $\phi_I(x_1,\ldots,x_n) = \sum_{(u,v)\in E} w(u,v) \cdot \gamma(x_u, x_v) + \rho_0(s) + \rho_1(t)$.

It is well known that the (s,t)-Min-Cut problem is solvable in polynomial time. Since every instance I of VCSP(Γ_{cut}) can be reduced to an instance of the (s,t)-Min-Cut problem, I is solvable in polynomial time. Thus, Γ_{cut} is an example of a *tractable* constraint language. When VCSP(Γ) is NP-hard, we call Γ an *intractable* language.

It is natural to ask about the complexity of VCSP(Γ) in terms of Γ. For *Boolean* valued constraint languages, we have a complete answer: Cohen et al. [8] showed that every Boolean valued constraint language is either tractable or intractable, thus obtaining what is known as a dichotomy theorem. In fact, [8] identified eight different types of tractable valued constraint languages; one of these types corresponds to submodularity [22] and includes Γ_{cut}. The dichotomy theorem from [8] is an extension of Schaefer's celebrated result, which gave a dichotomy for $\{0,\infty\}$-valued constraint languages [21], and the work of Creignou [9], who gave a dichotomy theorem for $\{0,1\}$-valued constraint languages.

The (global) Min-Cut problem asks, given a graph $G = (V,E)$ and edge weights $w : E \to \mathbb{Q}_{>0}$, for a subset $C \subseteq V$ with $\emptyset \subsetneq C \subsetneq V$ minimising $w(C) = \sum_{\{u,v\}\in E, |\{u,v\}\cap C|=1} w(u,v)$ [22]. This fundamental problem is an example of a Boolean *surjective* VCSP. Given a VCSP instance $I = (V, D, \phi_I)$, in the surjective setting the goal is to find a surjective assignment $s : V \to D$ minimising ϕ_I; here s is called surjective if for every $d \in D$ there is $x \in V$ such that $s(x) = d$. For Boolean VCSPs with $D = \{0,1\}$, this simply means that the all-zero and all-one assignments are not allowed. Given a language Γ, we denote by $\mathrm{VCSP_s}(\Gamma)$ the class of all instances I of the surjective VCSP that use only weighted relations from Γ in their objective function.

Let $D = \{0,1\}$ and define $\gamma : D \to \overline{\mathbb{Q}}$ by $\gamma(x,y) = 0$ if $x = y$ and $\gamma(x,y) = 1$ if $x \neq y$. $\mathrm{VCSP_s}(\{\gamma\})$ captures Min-Cut as every instance of $\mathrm{VCSP_s}(\{\gamma\})$ is a Min-Cut instance and vice versa. Since Min-Cut is solvable in polynomial time (say, by a reduction to the (s,t)-Min-Cut problem but other algorithms exist [23]), $\{\gamma\}$ is an example of a surjectively tractable, or *s-tractable* for short, valued constraint language. As before, if $\mathrm{VCSP_s}(\Gamma)$ is NP-hard we call Γ a surjectively intractable, or *s-intractable* for short, language.

Surjective VCSPs

What can we say about the complexity of $\mathrm{VCSP_s}(\Gamma)$ *for arbitrary* Γ*? In particular, is every* Γ *s-tractable or s-intractable? What is the mathematical structure of s-tractable languages?*

First, observe that for a language Γ defined on D, we have $\mathrm{VCSP}(\Gamma) \leq_p \mathrm{VCSP_s}(\Gamma)$. Indeed, given an instance I of VCSP(Γ), construct a new instance I' of $\mathrm{VCSP_s}(\Gamma)$ by adding $|D|$ extra variables. Then, any solution to I can be extended to a surjective solution to I' of the same value and conversely, any (surjective) solution to I' induces a solution to I of the same value.

Second, observe that for a language Γ defined on D, we have $\mathrm{VCSP_s}(\Gamma) \leq_p \mathrm{VCSP}(\Gamma \cup \mathcal{C}_D)$, where $\mathcal{C}_D$ is the set of constants on D; that is, $\mathcal{C}_D = \{\rho_d \mid d \in D\}$, where ρ_d is defined by $\rho_d(x) = 0$ if $x = d$ and $\rho_d(x) = \infty$ if $x \neq d$. Indeed, given an instance of $\mathrm{VCSP_s}(\Gamma)$, constants can be used to go through all $O(n^{|D|})$ ways to assign all the labels from D to a $|D|$-subset of the n variables, each resulting in an instance of $\mathrm{VCSP}(\Gamma \cup \mathcal{C}_D)$. Consequently, a tractable language Γ defined on D with $\mathcal{C}_D \subseteq \Gamma$ is also an s-tractable language.

In this paper we deal with Boolean valued constraint languages defined on $D = \{0, 1\}$. By the two observations above, the only Boolean valued constraint languages for which tractability could be different from s-tractability are tractable languages that do not include constants.

For Boolean $\{0, \infty\}$-valued languages, Schaefer's dichotomy [21] gives six tractable cases, four of which include constants. Creignou and Hébrard showed that the remaining two cases (0-valid and 1-valid) are s-intractable, thus obtaining a dichotomy in the surjective setting [10].

For Boolean $\{0, 1\}$-valued languages, Creignou's dichotomy [9] gives three tractable cases, one of which includes constants. Uppman showed that the remaining two cases (0-valid and 1-valid) are s-tractable if they are almost-min-min or almost-max-max, respectively, and s-intractable otherwise, thus obtaining a dichotomy in the surjective setting [24].

Contributions. As our main contribution we classify all Boolean valued constraint languages (i.e. $\overline{\mathbb{Q}}$-valued languages) as s-tractable or s-intractable. Our result extends the classifications from [10] and [24]. Six of the eight tractable cases identified for Boolean valued constraint languages [8] include constants and thus are also s-tractable. The remaining two cases (0-optimal and 1-optimal[1]) are s-tractable if they satisfy a certain condition. This condition, defined formally in Definition 5, says that both the feasibility and optimality relations of every weighted relation in the language have to be a projection of a downset (in the 0-optimal case), or a projection of an upset (in the 1-optimal case). This shows that, surprisingly, s-tractability of valued constraint languages (that are not covered by the tractable languages with constants) does not depend on the rational-values in the weighted relations. It is only the structure of the underlying feasibility and optimality relations that matters. (However, the running time of our algorithm depends on these values.) Identifying this condition and establishing that it captures the precise borderline of s-tractability is our main contribution.

The hardness part of our result is proved in the same spirit as for $\{0, \infty\}$-valued and $\{0, 1\}$-valued languages by carefully analysing the types of weighted relations that can be obtained in gadgets in the surjective setting, and relying on the explicit dichotomy for Boolean VCSPs [8].

While 0-optimal and 1-optimal languages are trivially tractable for VCSPs, the algorithm for surjective VCSPs over the newly identified languages is nontrivial. The s-tractability part of our result is established by a reduction from $\overline{\mathbb{Q}}$-valued $\mathrm{VCSP_s}$ to the *Generalised Min-Cut* problem (defined in Section 3), in which we require to find in polynomial time all α-optimal solutions, where α is a constant depending on the (finite) valued constraint language. The algorithm for the Generalised Min-Cut problem is essentially the same as in [24]. We show that the algorithm works in the more general setting with one part of the objective function being a $(\mathbb{Q}_{\geq 0} \cup \{\infty\})$-valued superadditive set function given by an oracle; see Section 3 for the details. By providing a tighter analysis we are able to improve the bound on the running time from roughly $O(n^{3^{3\alpha}})$ to $O(n^{20\alpha})$, thus answering one of the open problems from [24].

[1] A weighted relation is 0-optimal (1-optimal) if the all-zero (all-one) tuple minimises it.

We also show that the dependence of the running time on the language is unavoidable unless P = NP (cf. Example 27).

All omitted proofs are available in the full version of the paper [14].

Related work. Recent years have seen some remarkable progress on the computational complexity of CSPs and VCSPs parametrised by the (valued) constraint language, see [1] for a survey. We highlight the resolution of the "bounded width conjecture" [2] and the result that a dichotomy for CSPs, conjectured in [11], implies a dichotomy for VCSPs [19, 18]. All this work is for arbitrary (and thus not necessarily Boolean) finite domains and relies on the so-called algebraic approach initiated in [6] and nicely described in a recent survey [3]. One of the important aspects of the algebraic approach is the assumption that constants are present in (valued) constraint languages. (This is without loss of generality with respect to polynomial-time solvability.) It is the lack of constants in the surjective setting that makes it difficult, if not impossible, to employ the algebraic approach in this setting. See the work of Chen [7] for an initial attempt.

For a binary (unweighted) relation γ, $\mathrm{VCSP_s}(\{\gamma\})$ has been studied under the name of surjective γ-Colouring [4, 20] and vertex-compaction [26]. We remark that our notion of surjectivity is global. For the γ-Colouring problem, a local version of surjectivity has also been studied [13, 12].

2 Preliminaries

We denote by $\leq_p$ the standard polynomial-time Turing reduction. If $A \leq_p B$ and $B \leq_p A$ we write $A \equiv_p B$.

We use the notation $[n] = \{1, \dots, n\}$. For any tuple $\mathbf{x} \in D^r$, we refer to its ith element as x_i. For $\mathbf{x}, \mathbf{y} \in D^r$, we define $\mathbf{x} \leq \mathbf{y}$ if and only if $x_i \leq y_i$ for all $i \in [r]$.

We define *relations* as a special case of weighted relations with range $\{c, \infty\}$, where value $c \in \mathbb{Q}$ is assigned to tuples that are elements of the relation in the conventional sense. We will use both views interchangeably and choose $c = 0$ unless stated otherwise. Relations are also called unweighted or *crisp*.

If $\mathbf{s} \in [r]^n$ is a tuple of coordinates then for any $\mathbf{x} \in D^r$ we denote its projection to $\mathbf{s}$ by $\mathrm{Pr}_{\mathbf{s}}(\mathbf{x}) = (x_{s_1}, \dots, x_{s_n}) \in D^n$. For any relation ρ, we define $\mathrm{Pr}_{\mathbf{s}}(\rho) = \{\mathrm{Pr}_{\mathbf{s}}(\mathbf{x}) \mid \mathbf{x} \in \rho\}$. Note that the coordinates in $\mathbf{s}$ may repeat.

We denote by $\rho_=$ the binary equality relation $\{(x, x) \mid x \in D\}$. Recall from Section 1 that we denote, for any $d \in D$, by ρ_d the unary relation $\{(d)\}$, i.e. $\rho_d(x) = 0$ if $x = d$ and $\rho_d(x) = \infty$ if $x \neq d$.

▶ **Definition 1.** For a weighted relation $\gamma : D^r \to \overline{\mathbb{Q}}$, we denote by

- $\mathrm{Feas}(\gamma) = \{\mathbf{x} \in D^r \mid \gamma(\mathbf{x}) < \infty\}$ the underlying *feasibility relation*; and by
- $\mathrm{Opt}(\gamma) = \{\mathbf{x} \in \mathrm{Feas}(\gamma) \mid \gamma(\mathbf{x}) \leq \gamma(\mathbf{y}) \text{ for every } \mathbf{y} \in D^r\}$ the relation of *optimal* tuples.

We define $\mathrm{Feas}(\Gamma) = \{\mathrm{Feas}(\gamma) \mid \gamma \in \Gamma\}$ and $\mathrm{Opt}(\Gamma) = \{\mathrm{Opt}(\gamma) \mid \gamma \in \Gamma\}$.

An assignment $s : V \to D$ for a VCSP instance $I = (V, D, \phi_I)$ with $V = \{x_1, \dots, x_n\}$ is called *feasible* if $\phi_I(s(x_1), \dots, s(x_n)) < \infty$; s is called *optimal* if $\phi_I(s) \leq \phi_I(s')$ for every assignment s'.

Recall from Section 1 that any set of weighted relation Γ is called a valued constraint language. Γ is called *s-tractable* if for any finite subset $\Gamma' \subseteq \Gamma$ any instance of $\mathrm{VCSP_s}(\Gamma')$ can be solved in polynomial time. Γ is called *s-intractable* if $\mathrm{VCSP}(\Gamma')$ is NP-hard for some finite $\Gamma' \subseteq \Gamma$.

We apply a k-ary operation $h : D^k \to D$ to k r-tuples componentwise; i.e. $h(\mathbf{x}^1, \dots, \mathbf{x}^k) = (h(x_1^1, x_1^2, \dots, x_1^k), h(x_2^1, x_2^2, \dots, x_2^k), \dots, h(x_r^1, x_r^2, \dots, x_r^k))$.

The following notion is at the heart of the algebraic approach to decision CSPs [6].

▶ **Definition 2.** Let γ be a weighted relation on D. A k-ary operation $h : D^k \to D$ is a *polymorphism* of γ (and γ is *invariant under* or *admits* h) if, for every $\mathbf{x}^1, \dots, \mathbf{x}^k \in \mathrm{Feas}(\gamma)$, we have $h(\mathbf{x}^1, \dots, \mathbf{x}^k) \in \mathrm{Feas}(\gamma)$. We say that h is a polymorphism of a language Γ if it is a polymorphism of every $\gamma \in \Gamma$.

The following notion, which involves a collection of k k-ary polymorphisms, plays an important role in the complexity classification of Boolean valued constraint languages [8].

▶ **Definition 3.** Let γ be a weighted relation on D. A list $\langle h_1, \dots, h_k \rangle$ of k-ary polymorphisms of γ is a k-ary *multimorphism* of γ (and γ *admits* $\langle h_1, \dots, h_k \rangle$) if, for every $\mathbf{x}^1, \dots, \mathbf{x}^k \in \mathrm{Feas}(\gamma)$, we have

$$\sum_{i=1}^{k} \gamma(h_i(\mathbf{x}^1, \dots, \mathbf{x}^k)) \;\leq\; \sum_{i=1}^{k} \gamma(\mathbf{x}^i)\,.$$

$\langle h_1, \dots, h_k \rangle$ is a multimorphism of a language Γ if it is a multimorphism of every $\gamma \in \Gamma$.

Boolean VCSPs

For the rest of the paper let $D = \{0, 1\}$. We define some important operations on D. For any $a \in D$, c_a is the constant unary operation such that $c_a(x) = a$ for all $x \in D$. Operation $\neg$ is the unary negation, i.e. $\neg(0) = 1$ and $\neg(1) = 0$. Binary operation min (max) returns the smaller (larger) of its two arguments with respect to the order $0 < 1$. Ternary operation Mn (for minority) is the unique ternary operation on D satisfying $\mathrm{Mn}(x, x, y) = \mathrm{Mn}(x, y, x) = \mathrm{Mn}(y, x, x) = y$ for all $x, y \in D$. Ternary operation Mj (for majority) is the unique ternary operation on D satisfying $\mathrm{Mj}(x, x, y) = \mathrm{Mj}(x, y, x) = \mathrm{Mj}(y, x, x) = x$ for all $x, y \in D$.

▶ **Theorem 4** ([8])**.** *Let Γ be a Boolean valued constraint language. Then, Γ is tractable if it admits any the following eight multimorphisms $\langle c_0 \rangle$, $\langle c_1 \rangle$, $\langle \min, \min \rangle$, $\langle \max, \max \rangle$, $\langle \min, \max \rangle$, $\langle \mathrm{Mn}, \mathrm{Mn}, \mathrm{Mn} \rangle$, $\langle \mathrm{Mj}, \mathrm{Mj}, \mathrm{Mj} \rangle$, $\langle \mathrm{Mj}, \mathrm{Mj}, \mathrm{Mn} \rangle$. Otherwise, Γ is intractable.*

We note that Theorem 4 is a generalisation of Schaefer's classification of $\{0, \infty\}$-valued constraint languages [21] and Creignou's classification of $\{0, 1\}$-valued constraint languages [9].

The following definition is new in this paper and crucial for our main result.

▶ **Definition 5.** A relation ρ is a *downset (upset)* if for any $\mathbf{x}, \mathbf{y}$ such that $\mathbf{x} \geq \mathbf{y}$ ($\mathbf{x} \leq \mathbf{y}$) and $\mathbf{x} \in \rho$ it holds $\mathbf{y} \in \rho$. We will refer to relations that are a projection of a downset (upset) as *PDS (PUS)*. A weighted relation γ is called a *PDS (PUS) weighted relation* if both $\mathrm{Feas}(\gamma)$ and $\mathrm{Opt}(\gamma)$ are PDS (PUS). A language Γ is called PDS (PUS) if every weighted relation from Γ is PDS (PUS).

▶ **Example 6.** Relation $\rho = \{(0, 0), (0, 1), (1, 0)\}$ is a downset and hence also a PDS, while $\rho' = \{(0, 0, 0), (0, 1, 1), (1, 0, 0)\}$ is a PDS (as $\rho' = \mathrm{Pr}_{(1,2,2)}(\rho)$) but not a downset. Relation $\rho_=$ is a PDS relation and also a PDS weighted relation.

As one of the reviewers pointed out, PDS (PUS) relations are characterised by the binary polymorphism $x \wedge \neg y$ ($\neg x \vee y$).

▶ **Observation 7.** *Any PDS relation can be written as a sum of a downset and binary equality relations. More formally, if $\rho : D^r \to \{0, \infty\}$ is a PDS then we can write*

$$\rho(x_1, \ldots, x_r) \;=\; \rho'(x_{\pi(1)}, \ldots, x_{\pi(r')}) + \sum_{j=r'+1}^{r} \rho_{=}(x_{\pi(j)}, x_{i_j}),$$

where $\rho' : D^{r'} \to \{0, \infty\}$ is a downset, π is a permutation of $[r]$, and $i_j \in \{\pi(1), \ldots, \pi(r')\}$ for every $r' + 1 \le j \le r$.

The following result is our main contribution.

▶ **Theorem 8 (Main).** *Let Γ be a Boolean valued constraint language. Then, Γ is s-tractable if it admits any of the following six multimorphisms $\langle \min, \min \rangle$, $\langle \max, \max \rangle$, $\langle \min, \max \rangle$, $\langle \mathrm{Mn}, \mathrm{Mn}, \mathrm{Mn} \rangle$, $\langle \mathrm{Mj}, \mathrm{Mj}, \mathrm{Mj} \rangle$, $\langle \mathrm{Mj}, \mathrm{Mj}, \mathrm{Mn} \rangle$, or Γ is PDS, or Γ is PUS. Otherwise, Γ is s-intractable.*

Theorem 8 generalises the following two previously established results.

▶ **Theorem 9** ([10])**.** *Let Γ be a Boolean $\{0, \infty\}$-valued constraint language. Then, Γ is s-tractable if it admits any of the following four polymorphisms* min*,* max*,* Mn*,* Mj*. Otherwise, Γ is s-intractable.*

▶ **Theorem 10** ([24])**.** *Let Γ be a Boolean $\{0, 1\}$-valued constraint language. Then, Γ is s-tractable if it admits the $\langle \min, \max \rangle$ multimorphism, or Γ is PDS, or Γ is PUS. Otherwise, Γ is s-intractable.*

Theorem 10 is stated differently in [24] as the definition of PDS (PUS) languages is introduced in this paper. In fact, it was not a priori clear what the right condition for tractability should be for $\overline{\mathbb{Q}}$-valued constraint languages. Note that for $\{0, 1\}$-valued languages, the condition Feas(Γ) being PDS or PUS is vacuously true. Thus, a $\{0, 1\}$-valued language Γ is PDS (PUS) if Opt(Γ) is PDS (PUS) and this is equivalent to Γ being almost-min-min (almost-max-max) [24].

Recall that $\neg$ is the unary negation operation. For a weighted relation γ, we define $\neg(\gamma)$ to be the weighted relation $\neg(\gamma)(\mathbf{x}) = \gamma(\neg(\mathbf{x}))$. For a language Γ, we define $\neg(\Gamma) = \{\neg(\gamma) \mid \gamma \in \Gamma\}$.

The following observation follows from Definition 5.

▶ **Observation 11.** *A valued constraint language Γ is PDS if and only if $\neg(\Gamma)$ is PUS.*

▶ **Lemma 12.** *For a Boolean valued constraint language Γ, $\mathrm{VCSP}_{\mathrm{s}}(\Gamma) \equiv_p \mathrm{VCSP}_{\mathrm{s}}(\neg(\Gamma))$.*

Proof of Theorem 8. The s-tractability of languages admitting any of the six multimorphisms in the statement of the theorem follows from Theorem 4 via the reduction $\mathrm{VCSP}_{\mathrm{s}}(\Gamma) \le_p \mathrm{VCSP}(\Gamma \cup \{\rho_0, \rho_1\})$ discussed in the introduction. The s-tractability of PDS languages follows from Theorem 19, proved in Section 3. The s-tractability of PUS languages is then a simple corollary of Theorem 19, Observation 11, and Lemma 12.

The s-intractability of the remaining languages is proved in the full version of the paper [14]. The key in the hardness proof is to identify certain operators on weighted relations that preserve s-tractability, polymorphisms for crisp relations and multimorphisms for weighted relations, thus allowing for the construction of hardness gadgets. These operators include scaling by a nonnegative rational, adding a rational, permutation of arguments, identification of arguments, addition, the Feas($\cdot$) and Opt($\cdot$) operators, and pinning labels to variables (assuming the corresponding crisp constant is available). ◀

3 Tractability of PDS languages

We prove that PDS languages are s-tractable by a reduction to a generalised variant of the Min-Cut problem. The problem and the reduction are stated in Subsection 3.1. The tractability of the Generalised Min-Cut problem is established in Subsection 3.2.

3.1 Reduction to the Generalised Min-Cut problem

Let V be a finite set. A *set function* on V is a function $f : 2^V \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$ with $f(\emptyset) = 0$.

▶ **Definition 13.** A set function $f : 2^V \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$ is *increasing* if $f(X) \leq f(Y)$ for all $X \subseteq Y \subseteq V$; it is *superadditive* if $f(X) + f(Y) \leq f(X \cup Y)$ for all disjoint $X, Y \subseteq V$; it is *posimodular* if $f(X) + f(Y) \geq f(X \setminus Y) + f(Y \setminus X)$ for all $X, Y \subseteq V$; and finally it is *submodular* if $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$ for all $X, Y \subseteq V$.

It is known and easy to show that any superadditive set function is also increasing.

▶ **Example 14.** Let U be a finite set and $T \subseteq U$ a non-empty subset. We define a set function f on U by $f(X) = 1$ if $T \subseteq X$ and $f(X) = 0$ otherwise. Intuitively, this corresponds to a soft NAND constraint if we interpret T as its scope and X as the set of variables assigned *true*. The set function f is superadditive, and hence also increasing.

We now formally define the Min-Cut problem introduced in Section 1.

▶ **Definition 15.** An instance of the *Min-Cut* (MC) problem is given by a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{Q}_{>0}$. The goal is to minimise the objective function g, which is a set function on V defined by $g(X) = \sum_{\{u,v\} \in E, |\{u,v\} \cap X| = 1} w(u, v)$.

Note that g from Definition 15 is posimodular.

Any set X such that $\emptyset \subsetneq X \subsetneq V$ is called a *solution* of the MC problem. Note that a cut $(X, V \setminus X)$ corresponds to two solutions, namely X and $V \setminus X$. Any solution that minimises the objective function g is called *optimal*, and any optimal solution with no proper subset being an optimal solution is called *minimal*. Note that any two different minimal optimal solutions X, Y are disjoint, as $X \setminus Y$ and $Y \setminus X$ are also optimal solutions (by the posimodularity of g).

We now define the Generalised Min-Cut problem, which is key to establishing Theorem 8.

▶ **Definition 16.** Let $G = (V, E)$ be an undirected graph with edge weights $w : E \to \mathbb{Q}_{>0}$. Let g be the objective function of the Min-Cut problem on G and f a superadditive set function on V given by an oracle. A solution to the *Generalised Min-Cut* (GMC) problem is any set X such that $\emptyset \subsetneq X \subsetneq V$, and the objective is to minimise the value of $f(X) + g(X)$.

We will denote this minimum by λ. A solution achieving the minimum is called *optimal*. In case of $0 < \lambda < \infty$, we will also be looking for all α-*optimal* solutions (for $\alpha \geq 1$) to the problem, i.e. solutions X such that $f(X) + g(X) \leq \alpha\lambda$. As shown in Subsection 3.2, Theorem 26, this can be done in polynomial time (for a fixed α).

Uppman [24] used the term "Generalised Min-Cut problem" for a special case of Definition 16, in which the superadditive function f is given explicitly as a weighted sum of soft NANDs (cf. Example 14).

Our reduction from the surjective VCSP over a PDS language to the GMC problem is based on the following notion.

▶ **Definition 17.** Let γ be an r-ary weighted relation on domain $D = \{0, 1\}$. We will associate any r-tuple $\mathbf{x} \in D^r$ with the set $X = \{i \in [r] \mid x_i = 1\}$ and use them interchangeably.

Let J be an instance of the GMC problem on vertices $[r]$ with $J(X)$ denoting the objective value for any $\mathbf{x} \in D^r$ (including the all-zero and all-one tuples, which correspond to non-solutions $\emptyset$ and $[r]$). For any $\alpha \geq 1$, we say that J *α-approximates* γ if $J(X) \leq \gamma(\mathbf{x}) \leq \alpha \cdot J(X)$ for all $\mathbf{x} \in D^r$.

▶ **Lemma 18.** *Let γ be a weighted relation such that* $\mathrm{Feas}(\gamma)$ *is a downset,* $\mathrm{Opt}(\gamma)$ *is a PDS, and $\gamma(\mathbf{x}) = 0$ for $\mathbf{x} \in \mathrm{Opt}(\gamma)$. There is a constant α and an instance of the GMC problem that α-approximates γ.*

Proof. We define a set function f_{Feas} on $[r]$ as $f_{\mathrm{Feas}}(X) = 0$ if $\mathbf{x} \in \mathrm{Feas}(\gamma)$ and $f_{\mathrm{Feas}}(X) = \infty$ otherwise. Because $\mathrm{Feas}(\gamma)$ is a downset, f_{Feas} is superadditive.

By Observation 7, we can write $\mathrm{Opt}(\gamma)$ as a sum of a downset ρ on coordinates $A \subseteq [r]$ and equalities $x_i = x_j$ for $(i, j) \in E$ with $|A| + |E| = r$. Let $\mathbf{x}|_A$ denote the projection of an r-tuple $\mathbf{x}$ to coordinates A in the same order as in ρ. We define a set function f_{Opt} on $[r]$ as $f_{\mathrm{Opt}}(X) = 0$ if $\mathbf{x}|_A \in \rho$ and $f_{\mathrm{Opt}}(X) = |X \cap A|$ otherwise. Because ρ is a downset, f_{Opt} is superadditive.

We define a GMC instance J' on vertices $[r]$, unit-weight edges E, and the superadditive set function $f_{\mathrm{Feas}} + f_{\mathrm{Opt}}$. By the construction, it holds

$$J'(X) = \infty \iff f_{\mathrm{Feas}}(X) = \infty \iff \mathbf{x} \notin \mathrm{Feas}(\gamma) \iff \gamma(\mathbf{x}) = \infty \tag{1}$$

and

$$\begin{aligned} J'(X) = 0 &\iff f_{\mathrm{Feas}}(X) = f_{\mathrm{Opt}}(X) = 0 \ \wedge\ |\{i, j\} \cap X| \neq 1 \text{ for all } (i, j) \in E && (2)\\ &\iff \mathbf{x} \in \mathrm{Feas}(\gamma) \ \wedge\ \mathbf{x}|_A \in \rho \ \wedge\ x_i = x_j \text{ for all } (i, j) \in E && (3)\\ &\iff \mathbf{x} \in \mathrm{Opt}(\gamma) \iff \gamma(\mathbf{x}) = 0\,. && (4) \end{aligned}$$

Moreover, for any X such that $0 < J'(X) < \infty$ it holds $1 \leq J'(X) \leq r$.

If γ is crisp then the instance J' 1-approximates γ; otherwise let $\delta_{\min}, \delta_{\max}$ denote the minimum and maximum of $\{\gamma(\mathbf{x}) \mid 0 < \gamma(\mathbf{x}) < \infty\}$. We scale the weights of the edges and the superadditive function of J' by a factor of $\delta_{\min}/r$ to obtain an instance J such that $J(X) \leq \gamma(\mathbf{x})$ for all X. Setting $\alpha = r \cdot \delta_{\max}/\delta_{\min}$ then gives $\gamma(\mathbf{x}) \leq \alpha \cdot J(X)$ for all X. ◀

Finally, we state the reduction.

▶ **Theorem 19.** *Let Γ be a Boolean valued constraint language. If Γ is PDS, then it is s-tractable.*

Proof. Let $\Gamma' \subseteq \Gamma$ be a finite language. The feasibility relation $\mathrm{Feas}(\gamma)$ for any $\gamma \in \Gamma'$ is a PDS and hence, by Observation 7, a sum of a downset and binary equality relations. A crisp equality constraint $\rho_=(x, y)$ in an instance can be omitted after identifying the variables x and y. Therefore, we will assume that $\mathrm{Feas}(\gamma)$ is a downset. Moreover, we will assume that the minimum value assigned by γ is 0, as changing values $\gamma(\mathbf{x})$ by the same constant for all $\mathbf{x} \in D^{\mathrm{ar}(\gamma)}$ affects all assignments equally.

By Lemma 18, for any $\gamma \in \Gamma'$, there is a constant α_γ and a GMC instance J_γ that α_γ-approximates γ. Let α be the smallest integer such that $\alpha \geq \alpha_\gamma$ for all $\gamma \in \Gamma'$.

Given a $\mathrm{VCSP_s}(\Gamma')$ instance I with an objective function $\phi_I(x_1, \ldots, x_n) = \sum_{i=1}^{q} w_i \cdot \gamma_i(\mathbf{x}^i)$, we construct a GMC instance J that α-approximates ϕ_I. For $i \in [q]$, we relabel the vertices of J_{γ_i} to match the variables in the scope $\mathbf{x}^i$ of the ith constraint (i.e. vertex j is relabelled

to x^i_j) and identify vertices in case of repeated variables. We also scale both the weights of the edges of J_{γ_i} and the superadditive function by w_i. The instance J is obtained by adding up the GMC instances J_{γ_i} for all $i \in [q]$.

Let $\mathbf{x} \in D^n$ denote a surjective assignment minimising ϕ_I and $\mathbf{y} \in D^n$ an optimal solution to J with $J(Y) = \lambda$. Because J α-approximates ϕ_I, it holds

$$\lambda \leq J(X) \leq \phi_I(\mathbf{x}) \leq \phi_I(\mathbf{y}) \leq \alpha \cdot J(Y) = \alpha\lambda\,, \tag{5}$$

and hence $\mathbf{x}$ is an α-optimal solution to J. By Lemma 20 in Subsection 3.2, we can determine whether $\lambda = 0$, in which case any optimal solution to J is also optimal for ϕ_I; and whether $\lambda = \infty$. If $0 < \lambda < \infty$, we find all α-optimal solutions by Theorem 26 in Subsection 3.2. ◀

3.2 Tractability of the Generalised Min-Cut problem

Proofs of Lemmas 20, 21, and 23 can be found in the full version of the paper [14].

▶ **Lemma 20.** *There is a polynomial-time algorithm that, given an instance of the GMC problem, either finds a solution X with $f(X) + g(X) = 0$, or determines that $\lambda = \infty$, or determines that $0 < \lambda < \infty$.*

In view of Lemma 20, we can assume that $0 < \lambda < \infty$. Our goal is to show that, for a given $\alpha \geq 1$, all α-optimal solutions to a GMC instance can be found in polynomial time. This will be proved in Theorem 26; before that we need to prove several auxiliary lemmas on properties of the MC and GMC problems.

▶ **Lemma 21.** *For any instance J of the GMC problem on a graph $G = (V, E)$ and any non-empty set $V' \subseteq V$, there is an instance J' on the induced subgraph $G[V']$ that preserves the objective value of all solutions $X \subsetneq V'$. In particular, any α-optimal solution X of J such that $X \subsetneq V'$ is α-optimal for J' as well.*

▶ **Lemma 22.** *Let X be an optimal solution to an instance of the GMC problem over vertices V with $\lambda < \infty$, and Y a minimal optimal solution to the underlying MC problem. Then $X \subseteq Y$, $X \subseteq V \setminus Y$, or X is an optimal solution to the underlying MC problem.*

Proof. Assume that $X \not\subseteq Y$ and $X \not\subseteq V \setminus Y$. If $Y \subseteq X$, we have $f(Y) \leq f(X)$ as f is increasing, and hence $f(Y) + g(Y) \leq f(X) + g(X) < \infty$. Therefore, Y is optimal for the GMC problem and X is optimal for the MC problem. In the rest, we assume that $Y \not\subseteq X$.

By the posimodularity of g we have $g(X) + g(Y) \geq g(X \setminus Y) + g(Y \setminus X)$. Because $Y \setminus X$ is a proper non-empty subset of Y, it holds $g(Y \setminus X) > g(Y)$, and hence $g(X) > g(X \setminus Y)$. But then $f(X) + g(X) > f(X \setminus Y) + g(X \setminus Y)$ as $\infty > f(X) \geq f(X \setminus Y)$. Set $X \setminus Y$ is non-empty, and therefore contradicts the optimality of X. ◀

The following lemma relates the number of optimal solutions and the number of minimal optimal solutions to the MC problem. Note that this bound is tight for (unweighted) paths and cycles with at most one path attached to each vertex.

▶ **Lemma 23.** *For any instance of the MC problem on a connected graph with n vertices and p minimal optimal solutions, there are at most $p(p-1) + 2(n-p)$ optimal solutions.*

▶ **Lemma 24.** *For any instance of the GMC problem on n vertices with $0 < \lambda < \infty$, the number of optimal solutions is at most $n(n-1)$. There is an algorithm that finds all of them in polynomial time.*

Note that the bound of $n(n-1)$ optimal solutions precisely matches the known upper bound of $\binom{n}{2}$ for the number of minimum cuts [17]; the bound is tight for cycles.

Proof. Let $t(n)$ denote the maximum number of optimal solutions for such instances on n vertices. We prove the bound by induction on n. If $n = 1$, there are no solutions and hence $t(1) = 0$. For $n \geq 2$, let $Y_1, \ldots, Y_p$ be the minimal optimal solutions to the underlying MC problem. As there exists at least one minimum cut and the minimal optimal solutions are all disjoint, it holds $2 \leq p \leq n$.

Suppose that the minimal optimal solutions cover all vertices, i.e. $\bigcup Y_i = V$. By Lemma 22, any optimal solution to the GMC problem is either a proper subset of some Y_i or an optimal solution to the underlying MC problem. Restricting solutions to a proper subset of Y_i is by Lemma 21 equivalent to considering a GMC problem instance on vertices Y_i, and hence the number of such optimal solutions is bounded by $t(|Y_i|) \leq |Y_i| \cdot (|Y_i| - 1)$. Note that the sum $\sum |Y_i| \cdot (|Y_i| - 1)$ is maximised when $p - 1$ of the sets Y_i are singletons and the size of the remaining one equals $n - p + 1$. If the graph is connected then, by Lemma 23, there are at most $p(p-1) + 2(n-p)$ optimal solutions to the underlying MC problem. Adding these upper bounds we get

$$p(p-1) + 2(n-p) + \sum_{i=1}^{p} |Y_i| \cdot (|Y_i| - 1) \tag{6}$$

$$\leq p(p-1) + 2(n-p) + (p-1) \cdot 1 \cdot 0 + (n-p+1) \cdot (n-p) \tag{7}$$

$$= n(n-1) - 2(p-2) \cdot (n-p) \tag{8}$$

$$\leq n(n-1)\,. \tag{9}$$

If the graph is disconnected, the sets $Y_1, \ldots, Y_p$ are precisely its connected components. The optimal solutions to the underlying MC problem are precisely unions of connected components (with the exception of $\emptyset$ and V), which means that there can be exponentially many of them. However, only the sets $Y_1, \ldots, Y_p$ themselves can be optimal solutions to the GMC problem: We have $0 < \lambda \leq f(Y_i) + g(Y_i) = f(Y_i)$. Because f is superadditive, it holds $f(Y_{i_1} \cup \cdots \cup Y_{i_k}) \geq f(Y_{i_1}) + \cdots + f(Y_{i_k}) \geq k\lambda$ for any distinct $i_1, \ldots, i_k$, and hence no union of two or more connected components can be an optimal solution to the GMC problem. This gives us an upper bound of $p \leq p(p-1) + 2(n-p)$, and the rest follows as in the previous case.

Finally, suppose that $\bigcup Y_i \neq V$, and hence the graph is connected. This case can be handled similarly as for $\bigcup Y_i = V$. (The full version of the paper [14] includes a complete proof.)

Using a procedure generating all minimum cuts [25], it is straightforward to turn the above proof into a recursive algorithm that finds all optimal solutions in polynomial time. ◀

▶ **Lemma 25.** *Let $\alpha, \beta \geq 1$. Let X be an α-optimal solution to an instance of the GMC problem over vertices V with $0 < \lambda < \infty$, and Y an optimal solution to the underlying MC problem. If $g(Y) < \lambda/\beta$, then*

$$(f(X \setminus Y) + g(X \setminus Y)) + (f(X \cap Y) + g(X \cap Y)) < \left(\alpha + \frac{2}{\beta}\right) \lambda\,; \tag{10}$$

if $g(Y) \geq \lambda/\beta$, then X is an $\alpha\beta$-optimal solution to the underlying MC problem.

Proof. If $g(Y) \geq \lambda/\beta$, it holds $g(X) \leq f(X) + g(X) \leq \alpha\lambda \leq \alpha\beta \cdot g(Y)$, and hence X is an $\alpha\beta$-optimal solution to the underlying MC problem. In the rest we assume that $g(Y) < \lambda/\beta$.

Because g is posimodular, we have

$$g(X) + g(Y) \geq g(X \setminus Y) + g(Y \setminus X) \tag{11}$$
$$g(Y) + g(Y \setminus X) \geq g(X \cap Y) + g(\emptyset)\,, \tag{12}$$

and hence

$$g(X) + 2g(Y) \geq g(X \setminus Y) + g(X \cap Y)\,. \tag{13}$$

By superadditivity of f, it holds $f(X) \geq f(X \setminus Y) + f(X \cap Y)$. The claim then follows from the fact that $f(X) + g(X) + 2g(Y) < (\alpha + 2/\beta)\lambda$. ◀

Finally, we prove that α-optimal solutions to the GMC problem can be found in polynomial time.

▶ **Theorem 26.** *For any instance of the GMC problem on n vertices with $0 < \lambda < \infty$ and $\alpha \in \mathbb{Z}_{\geq 1}$, the number of α-optimal solutions is at most $n^{20\alpha-15}$. There is an algorithm that finds all of them in polynomial time.*

Note that for a cycle on n vertices, the number of α-optimal solutions to the MC problem is $\Theta(n^{2\alpha})$, and thus the exponent in our bound is asymptotically tight in α.

Proof. Let $\beta \in \mathbb{Z}_{\geq 3}$ be a parameter. Throughout the proof, we relax the integrality restriction on α and require only that $\alpha\beta$ is an integer. For $\alpha = 1$, the claim follows from Lemma 24, therefore we assume $\alpha \geq 1 + 1/\beta$ in the rest of the proof.

Define $\ell(x) = \frac{2(\beta+1)}{\beta-2} \cdot (\beta x - 3)$. We will prove that the number of α-optimal solutions is at most $n^{\ell(\alpha)}$; taking $\beta = 4$ then gives the claimed bound. Function ℓ was chosen as a slowest growing function satisfying the following properties required in this proof: It holds $\ell(x) + \ell(y) \leq \ell(x + y - 3/\beta)$ for any x, y, and $\ell(x) \geq 2\beta x$ for any $x \geq 1 + 1/\beta$.

We prove the bound by induction on $n + \alpha\beta$. As it trivially holds for $n \leq 2$, we will assume $n \geq 3$ in the rest of the proof. Let Y be an optimal solution to the underlying MC problem with $k = |Y| \leq n/2$. If $g(Y) \geq \lambda/\beta$ then, by Lemma 25, any α-optimal solution to the GMC problem is an $\alpha\beta$-optimal solution to the underlying MC problem. Because $g(Y) \geq \lambda/\beta > 0$, the graph is connected, and hence there are at most $2^{2\alpha\beta}\binom{n}{2\alpha\beta} \leq n^{2\alpha\beta} \leq n^{\ell(\alpha)}$ such solutions by [17]. (In detail, [17, Theorem 6.2] shows that the number of $\alpha\beta$-optimal cuts in an n-vertex graph is $2^{2\alpha\beta-1}\binom{n}{2\alpha\beta}$, and every cut corresponds to two solutions.)

From now on we assume that $g(Y) < \lambda/\beta$ and hence inequality (10) holds. Upper bounds in this case may be quite loose; in particular, we will use the following inequalities:

$$(k/n)^{\ell(\alpha)} \leq (k/n)^{\ell(1+1/\beta)} = (k/n)^{2(\beta+1)} \leq (k/n)^8 \leq (k/n)(1/2)^7 = k/128n \tag{14}$$
$$(1/n)^{2\beta} \leq (1/n)^6 \leq (1/n)(1/3)^5 < 1/128n\,. \tag{15}$$

Consider any α-optimal solution to the GMC problem X.

If $X \subsetneq Y$ then, by Lemma 21, X is an α-optimal solution to an instance on vertices Y. By the induction hypothesis, there are at most $k^{\ell(\alpha)} \leq (k/128n) \cdot n^{\ell(\alpha)}$ such solutions.

Similarly, if $X \subsetneq V \setminus Y$, then X is an α-optimal solution to an instance on vertices $V \setminus Y$, and there are at most $(n-k)^{\ell(\alpha)} = (1-k/n)^{\ell(\alpha)} \cdot n^{\ell(\alpha)} \leq (1-k/n) \cdot n^{\ell(\alpha)}$ such solutions.

If $Y \subsetneq X$, then $X \setminus Y$ is an $(\alpha - 1 + 2/\beta)$-optimal solution on vertices $V \setminus Y$ by (10) and the fact that $f(X \cap Y) + g(X \cap Y) \geq \lambda$. Similarly, if $V \setminus Y \subsetneq X$, then $X \cap Y$ is an $(\alpha - 1 + 2/\beta)$-optimal solution on vertices Y. In either case, we bound the number of such solutions depending on the value of α: For $\alpha < 2 - 2/\beta$, there are trivially none; for

$\alpha = 2 - 2/\beta$, Lemma 24 gives a bound of $n(n-1) \leq n^{\ell(\alpha)-2\beta}$; and for $\alpha > 2 - 2/\beta$ we get an upper bound of $n^{\ell(\alpha-1+2/\beta)} \leq n^{\ell(\alpha)-2\beta}$ by the induction hypothesis. The number of solutions is thus at most $n^{\ell(\alpha)-2\beta} \leq (1/128n) \cdot n^{\ell(\alpha)}$ for any α.

Finally, we consider X such that $\emptyset \subsetneq X \setminus Y \subsetneq V \setminus Y$ and $\emptyset \subsetneq X \cap Y \subsetneq Y$, i.e. $X \setminus Y$ and $X \cap Y$ are solutions on vertices $V \setminus Y$ and Y respectively. Let i be the integer for which

$$\left(1 + \frac{i}{\beta}\right)\lambda \leq f(X \cap Y) + g(X \cap Y) < \left(1 + \frac{i+1}{\beta}\right)\lambda\,. \tag{16}$$

Then, by (10), it holds $f(X \setminus Y) + g(X \setminus Y) < (\alpha - 1 - (i-2)/\beta)\lambda$. Therefore, $X \cap Y$ is a $(1 + (i+1)/\beta)$-optimal solution on vertices Y and $X \setminus Y$ is an $(\alpha - 1 - (i-2)/\beta)$-optimal solution on vertices $V \setminus Y$. Because $0 \leq i \leq (\alpha - 2)\beta + 1$, we can bound the number of such solutions by the induction hypothesis as at most

$$k^{\ell\left(1+\frac{i+1}{\beta}\right)} \cdot (n-k)^{\ell\left(\alpha-1-\frac{i-2}{\beta}\right)} \leq \left(\frac{k}{n}\right)^{\ell\left(1+\frac{i+1}{\beta}\right)} \cdot n^{\ell\left(1+\frac{i+1}{\beta}\right)+\ell\left(\alpha-1-\frac{i-2}{\beta}\right)} \tag{17}$$

$$\leq \left(\frac{k}{n}\right)^{2(\beta+1)} \cdot \frac{1}{2^i} \cdot n^{\ell(\alpha)}\,, \tag{18}$$

which is at most $2 \cdot (k/128n) \cdot n^{\ell(\alpha)}$ in total for all i.

By adding up the bounds we get that the number of α-optimal solutions is at most $n^{\ell(\alpha)}$. A polynomial-time algorithm that finds the α-optimal solutions follows from the above proof using a procedure generating all $\alpha\beta$-optimal cuts [25]. ◀

4 Conclusions

While the complexity of (valued) constraint languages is, as in this paper, studied mostly for finite languages, all known results also hold for languages of infinite size. We now show that this is *not* the case for Boolean surjective VCSPs.

▶ **Example 27.** We give an example of an infinite Boolean valued constraint language Γ that is a PDS language but $\mathrm{VCSP_s}(\Gamma)$ is NP-hard.

Let $D = \{0, 1\}$. For any $w \in \mathbb{Z}_{\geq 1}$, we define $\gamma_w : D^3 \to \overline{\mathbb{Q}}$ by $\gamma(0,0,0) = 0$, $\gamma(\cdot,\cdot,0) = w$, $\gamma(x,y,1) = 2$ if $x = y$ and $\gamma(x,y,1) = 1$ if $x \neq y$. Note that $\mathrm{Feas}(\gamma_w) = D^3$ and $\mathrm{Opt}(\gamma_w) = \{(0,0,0)\}$ are PDS relations. Let $\Gamma = \{\gamma_w \mid w \in \mathbb{Z}_{\geq 1}\}$.

Given an instance $G = (V, E)$ of the Max-Cut problem with $V = \{x_1, \ldots, x_n\}$ and no isolated vertices, we choose a value $w > 2|E|$ and construct a $\mathrm{VCSP_s}(\Gamma)$ instance $I = (V \cup \{z\}, D, \phi_I)$ with $\phi_I(x_1, \ldots, x_n, z) = \sum_{\{i,j\} \in E, i<j} \gamma_w(x_i, x_j, z)$. Cuts in G are in one-to-one correspondence with assignments to I satisfying $z = 1$. In particular, a cut of value k corresponds to an assignment to I of value $k + 2(|E| - k) = 2|E| - k$. Moreover, any surjective assignment that assigns label 0 to variable z is of value at least $w > 2|E| \geq 2|E| - k$. Thus, solving I amounts to solving Max-Cut in G.

An obvious open problem is to consider surjective VCSPs on a three-element domain. A complexity classification is known for $\{0, \infty\}$-valued languages [5] and $\mathbb{Q}$-valued languages [15] (the latter generalises the $\{0, 1\}$-valued case obtained in [16]). In fact [18] implies a dichotomy for $\overline{\mathbb{Q}}$-valued languages on a three-element domain. However, all these results depend on the notion of core and the presence of constants in the language, and thus it is unclear how to use them to obtain a complexity classification in the surjective setting. Moreover, one special case of the CSP on a three-element domain is the *3-No-Rainbow-Colouring* problem [4], whose complexity status is open.

References

1. Libor Barto. Constraint satisfaction problem and universal algebra. *ACM SIGLOG News*, 1(2):14–24, 2014. doi:10.1145/2677161.2677165.
2. Libor Barto and Marcin Kozik. Constraint Satisfaction Problems Solvable by Local Consistency Methods. *Journal of the ACM*, 61(1), 2014. Article No. 3. doi:10.1145/2556646.
3. Libor Barto, Andrei Krokhin, and Ross Willard. Polymorphisms, and how to use them. In Andrei Krokhin and Stanislav Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol7.15301.1.
4. Manuel Bodirsky, Jan Kára, and Barnaby Martin. The complexity of surjective homomorphism problems - a survey. *Discrete Applied Mathematics*, 160(12):1680–1690, 2012. doi:10.1016/j.dam.2012.03.029.
5. Andrei Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006. doi:10.1145/1120582.1120584.
6. Andrei Bulatov, Andrei Krokhin, and Peter Jeavons. Classifying the Complexity of Constraints using Finite Algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005. doi:10.1137/S0097539700376676.
7. Hubie Chen. An algebraic hardness criterion for surjective constraint satisfaction. *Algebra universalis*, 72(4):393–401, 2014. doi:10.1007/s00012-014-0308-x.
8. David A. Cohen, Martin C. Cooper, Peter G. Jeavons, and Andrei A. Krokhin. The Complexity of Soft Constraint Satisfaction. *Artificial Intelligence*, 170(11):983–1016, 2006. doi:10.1016/j.artint.2006.04.002.
9. Nadia Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *Journal of Computer and System Sciences*, 51(3):511–522, 1995. doi:10.1006/jcss.1995.1087.
10. Nadia Creignou and Jean-Jacques Hébrard. On generating all solutions of generalized satisfiability problems. *ITA*, 31(6):499–511, 1997.
11. Tomás Feder and Moshe Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.
12. Jiří Fiala and Jan Kratochvíl. Locally constrained graph homomorphisms - structure, complexity, and applications. *Computer Science Review*, 2(2):97–111, 2008. doi:10.1016/j.cosrev.2008.06.001.
13. Jiří Fiala and Daniël Paulusma. A complete complexity classification of the role assignment problem. *Theoretical Computer Science*, 349(1):67–81, 2005. doi:10.1016/j.tcs.2005.09.029.
14. Peter Fulla and Stanislav Živný. The complexity of Boolean surjective general-valued CSPs. *CoRR*, abs/1702.04679, 2017. URL: http://arxiv.org/abs/1702.04679.
15. Anna Huber, Andrei Krokhin, and Robert Powell. Skew bisubmodularity and valued CSPs. *SIAM Journal on Computing*, 43(3):1064–1084, 2014. doi:10.1137/120893549.
16. Peter Jonsson, Mikael Klasson, and Andrei A. Krokhin. The approximability of three-valued MAX CSP. *SIAM Journal on Computing*, 35(6):1329–1349, 2006. doi:10.1137/S009753970444644X.
17. David R. Karger. Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 21–30, 1993.
18. Vladimir Kolmogorov, Andrei A. Krokhin, and Michal Rolínek. The complexity of general-valued CSPs. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*. IEEE Computer Society, 2015.

19 Marcin Kozik and Joanna Ochremiak. Algebraic properties of valued constraint satisfaction problem. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP'15)*, volume 9134 of *Lecture Notes in Computer Science*, pages 846–858. Springer, 2015. doi:10.1007/978-3-662-47672-7_69.

20 Barnaby Martin and Daniël Paulusma. The computational complexity of disconnected cut and $2K_2$-partition. *Journal of Combinatorial Theory, Series B*, 111:17–37, 2015. doi:10.1016/j.jctb.2014.09.002.

21 Thomas J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC'78)*, pages 216–226. ACM, 1978. doi:10.1145/800133.804350.

22 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

23 Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997. doi:10.1145/263867.263872.

24 Hannes Uppman. Max-Sur-CSP on Two Elements. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP'12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 38–54. Springer, 2012. doi:10.1007/978-3-642-33558-7_6.

25 Vijay V. Vazirani and Mihalis Yannakakis. Suboptimal Cuts: Their Enumeration, Weight and Number (Extended Abstract). In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP'92)*, pages 366–377. Springer-Verlag, 1992. URL: http://dl.acm.org/citation.cfm?id=646246.684856.

26 Narayan Vikas. Algorithms for partition of some class of graphs under compaction and vertex-compaction. *Algorithmica*, 67(2):180–206, 2013. doi:10.1007/s00453-012-9720-9.

On the Expressive Power of Quasiperiodic SFT*

Bruno Durand[1] and Andrei Romashchenko[2]

1 **Univ. Montpellier & LIRMM, Montpellier, France**
2 **CNRS, Paris, France & LIRMM, Montpellier, France**
on leave from IITP RAS, Moscow, Russia

Abstract

In this paper we study the *shifts*, which are the shift-invariant and topologically closed sets of configurations over a finite alphabet in $\mathbb{Z}^d$. The *minimal shifts* are those shifts in which all configurations contain exactly the same patterns. Two classes of shifts play a prominent role in symbolic dynamics, in language theory and in the theory of computability: the *shifts of finite type* (obtained by forbidding a finite number of finite patterns) and the *effective shifts* (obtained by forbidding a computably enumerable set of finite patterns). We prove that every effective minimal shift can be represented as a factor of a projective subdynamics on a minimal shift *of finite type* in a bigger (by 1) dimension. This result transfers to the class of minimal shifts a theorem by M. Hochman known for the class of all effective shifts and thus answers an open question by E. Jeandel. We prove a similar result for quasiperiodic shifts and also show that there exists a quasiperiodic shift of finite type for which Kolmogorov complexity of all patterns of size $n \times n$ is $\Omega(n)$.

1998 ACM Subject Classification F.4.1 Computability theory, G.2.0 Discrete mathematics, G.2.1 Combinatorics

Keywords and phrases minimal SFT, tilings, quasiperiodicity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.5

1 Introduction

The study of symbolic dynamics was initially motivated as a discretization of classic dynamical systems, [9]. Later, the focus of attention in this area shifted towards the questions related to computability theory. The central notion of symbolic dynamics is a *shift* (a.k.a. *subshift*), which is a set of configurations in $\mathbb{Z}^d$ over a finite alphabet, defined by a set of forbidden patterns. Two major notions – two classes of shifts – play now a crucial role in symbolic dynamics: shifts of *finite type* (SFT, the shift defined by a finite set of forbidden patterns) and *effective* shifts (a.k.a. *effectively closed* – shifts with an enumerable set of forbidden patterns). These classes are distinct: every SFT is effective, but in general the reverse implication does not hold. However, the differences between these classes is surprisingly subtle. It is known that every effective shift can be simulated in some sense by an SFT of higher dimension. More precisely, every effective shift in $\mathbb{Z}^d$ can be represented as a factor of the projective subdynamics of an SFT of dimension increased by 1, see [10, 6, 1].

Usually, the proofs of computability results in symbolic dynamics involve sophisticated algorithmic gadgets embedded in dynamical systems. The resulting constructions are typically intricate and somewhat artificial. So, even if the shifts (effective or SFT) in general are proven to have a certain algorithmic property, the known proof may be inappropriate for "natural"

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

dynamical systems. Thus, it is interesting to understand the limits of the known algorithmic techniques and find out whether the remarkable properties of algorithmic complexity can be extended to "simple" and mathematically "natural" types of shifts.

One of the classic natural types of dynamical systems is the class of *minimal shifts*. Minimal shifts are those containing no proper shift, or equivalently the shifts where all configurations have exactly the same patterns. The role minimal shifts play in symbolic dynamics is similar to the role simple groups play in group theory (in particular every nonempty shift contains a nonempty minimal shift, see a discussion in [4]). Notice that all minimal shifts are quasiperiodic (but the converse is not true). Intuitively it seems that the structure of a minimal shift must be simple (in terms of dynamical systems). Besides, minimal shifts cannot be "too complex" in algorithmic terms. Indeed, it is known that every effective minimal shift has a computable language of patterns, and it contains at least one computable configuration [10] (which is in general not the case for effective shifts and even for SFT). Nevertheless, minimal shifts can have quite nontrivial algorithmic properties [13, 11].

We have mentioned above that every effective shift $\mathcal{S}$ can be represented as a factor of a projective subdynamics of an SFT $\mathcal{S}'$ (of higher dimension). In the previously known proofs of this result [10, 6, 1], even if $\mathcal{S}$ is minimal, the structure of the corresponding SFT $\mathcal{S}'$ (that simulates by its projective subdynamics the given $\mathcal{S}$) can be very sophisticated (and far from being minimal). So, a natural question arises (E. Jeandel, [12]): is it true that every effective minimal (or quasiperiodic) shift can be represented as a factor of a projective subdynamics on a minimal (respectively, quasiperiodic) SFT of higher dimension? In this paper we give a positive answer to that questions.

The full proof of the main result of this paper is rather cumbersome for the following reason: we use the technique of self-simulating tilings (e.g., [6, 7, 16]) combined with some combinatorial lemmas on quasiperiodic configurations. Unfortunately, there is no clean separation between the generic technique of self-simulating tilings and the supplementary features embedded in this type of tilings, so we cannot use the (previously known) technique of self-simulation as a "black box". We have to re-explain the core techniques of fixed-point programming embedded in tilings and adjust the supplementary features within the construction. While explaining the proofs, we have to balance clarity with formality, and given the usual space limits of the conference paper we have to sketch some standard parts of the proof. An extended version of this paper is published on `arXiv:1705.01876`.

1.1 Notation and basic definitions

Let Σ be a finite set (an alphabet). Fix an integer $d > 0$. A Σ-*configuration* is a mapping $\mathbf{f} : \mathbb{Z}^d \to \Sigma$. A $\mathbb{Z}^d$-*shift* (or just a *shift* if d is clear from the context) is a set of configuration that is (i) shift-invariant (with respect to the translations along each coordinate axis), and (ii) closed in Cantor's topology.

A *pattern* is a mapping from a finite subset in $\mathbb{Z}^d$ to Σ (a coloring of a finite set of $\mathbb{Z}^d$). Every shift can be defined by a set of forbidden finite patterns F (a configuration belongs to the shift if and only if it does not contain any pattern from F). A shift is called *effective* (or *effectively closed*) if it can be defined by a computably enumerable set of forbidden patterns. A shift is called a *shift of finite type* (SFT), if it can be defined by a finite set of forbidden patterns.

A special class of a 2-dimensional SFT is defined in terms of *Wang tiles*. In this case we interpret the alphabet Σ as a set of *tiles* – unite squares with colored sides, assuming that all colors belong to some finite set C (we assign one color to each side of a tile, so technically Σ is a subset of C^4). A (valid) *tiling* is a set of all configurations where every two neighboring

tiles match, i.e., share the same color on the adjacent sides. Wang tiles are powerful enough to simulate any SFT in a very strong sense: for each SFT $\mathcal{S}$ there exists a set of Wang tiles τ such that the set of all τ-tilings is isomorphic to $\mathcal{S}$. In this paper we focus on tilings since Wang tiles perfectly suit the technique of self-simulation.

A shift $\mathcal{S}$ (in the full shift $\Sigma^{\mathbb{Z}^d}$) can be interpreted as a dynamical system. There are d shifts along each of the coordinates, and each of these shifts map $\mathcal{S}$ to itself. So, the group $\mathbb{Z}^d$ naturally acts on $\mathcal{S}$.

For any shift $\mathcal{S}$ on $\mathbb{Z}^d$ and for any k-dimensional sublattice L in $\mathbb{Z}^d$, the *L-projective subdynamics* $\mathcal{S}_L$ of $\mathcal{S}$ is the set of configurations of $\mathcal{S}$ restricted on L. The L-projective subdynamics of a $\mathbb{Z}^d$-shift can be understood as a $\mathbb{Z}^k$-shift (notice that L naturally acts on $\mathcal{S}_L$). In particular, for every $d' < d$ we have a *standard* $\mathbb{Z}^{d'}$-projective subdynamics on the shift $\mathcal{S}$ generated by the lattice spanned on the first d' coordinate axis. In the proofs of Theorems 1-2 we deal with the standard $\mathbb{Z}^{(d-1)}$-projective subdynamics on $\mathbb{Z}^d$-shifts.

A configuration ω is called *recurrent* if every pattern that appears in ω at least once, must then appear in this configuration infinitely often. A configuration ω is called *quasiperiodic* (or *uniformly recurrent*) if every pattern P that appears in ω at least once, must appear in every pattern Q large enough in ω. Notice that every periodic configuration is also quasiperiodic. It is easy to see that if a shift $\mathcal{S}$ is minimal, then every $\omega \in \mathcal{S}$ is quasiperiodic.

For a quasiperiodic configuration ω, its *function of a quasiperiodicity* is a mapping $\varphi : \mathbb{N} \to \mathbb{N}$ such that every finite pattern of diameter n either never appears in ω, or it appears in every pattern of size $\varphi(n)$ in ω, see [4]. Similarly, a shift $\mathcal{S}$ has a function of quasiperiodicity φ, if φ is a function of a quasiperiodicity for every configuration in $\mathcal{S}$.

If a shift $\mathcal{S}$ is minimal, then all configurations in $\mathcal{S}$ have exactly the same finite patterns. For every minimal shift $\mathcal{S}$, the function of quasiperiodicity is finite (for every n) and even computable. Moreover, for an effective minimal shift, the set of all finite patterns (that can appear in any configuration) is computable, see [10, 3]. From this fact it follows that every effective and minimal shift contains a computable configuration.

1.2 The main results

Our first theorem claims that every effective quasiperiodic $\mathbb{Z}^d$-shift can be simulated by a quasiperiodic SFT in $\mathbb{Z}^{d+1}$.

▶ **Theorem 1.** *Let $\mathcal{A}$ be an effective quasiperiodic $\mathbb{Z}^d$-shift over some alphabet Σ_A. Then there exists a quasiperiodic SFT $\mathcal{B}$ (over another alphabet Σ_B) of dimension $d+1$ such that $\mathcal{A}$ is isomorphic to a factor of a d-dimensional projective subdynamics on $\mathcal{B}$.*

A similar result holds for effective minimal shifts:

▶ **Theorem 2.** *For every effective minimal $\mathbb{Z}^d$-shift $\mathcal{A}$ there exists a minimal SFT $\mathcal{B}$ in $\mathbb{Z}^{d+1}$ such that $\mathcal{A}$ is isomorphic to a factor of a d-dimensional projective subdynamics on $\mathcal{B}$.*

Theorem 1 implies the following somewhat surprising corollary (a quasiperiodic $\mathbb{Z}^2$-SFT can have highly "complex" languages of patterns):

▶ **Corollary 3.** *There exists a quasiperiodic SFT $\mathcal{A}$ of dimension 2 such that Kolmogorov complexity of every $(N \times N)$-pattern in every configuration of $\mathcal{A}$ is $\Omega(N)$.*

▶ Remark. A standalone pattern of size $N \times N$ over an alphabet Σ (with at least two letters) can have a Kolmogorov complexity up to $\Theta(N^2)$. However, this density of information cannot be enforced by local rules, because in every SFT in $\mathbb{Z}^2$ there exists a configuration such that Kolmogorov complexity of all $N \times N$-patterns is bounded by $O(N)$, [5]. Thus, the lower bound $\Omega(N)$ in Corollary 3 is optimal in the class of all SFT.

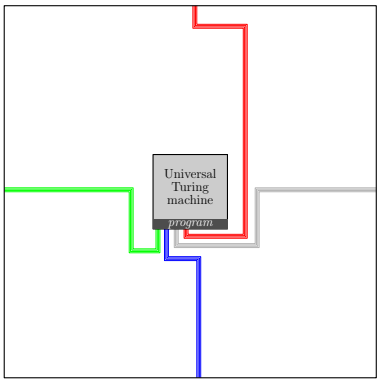


Figure 1 The structure of a macro-tile.

▶ Remark. Every effective (effectively-closed) minimal shift $\mathcal{A}$ is *computable* (given a pattern, we can algorithmically decide whether it belongs to the configurations of the shift). Patterns of high Kolmogorov complexity cannot be found algorithmically. So Corollary 3 cannot be extended to the class of minimal SFT.

To simplify notation and make the argument more visual, in what follows we focus on the case $d = 1$. The proofs extend to any $d > 1$ in a straightforward way, *mutatis mutandis*.

2 The general framework of self-simulating SFT

In what follows we extensively use the technique of *self-simulating tilesets* from [6] (this technique goes back to [8]). We use the idea of self-simulation to enforce a kind of *self-similar* structure in a tiling. In this section we remind the reader of the principal ingredients of this construction.

Let τ be a tileset and $N > 1$ be an integer. We call a τ-*macro-tile* an $N \times N$ square correctly tiled by tiles from τ. Every side of a τ-macro-tile contains a sequence of N colors (of tiles from τ); we refer to this sequence as a *macro-color*. A tileset τ *simulates* another tileset ρ, if there exists a set of τ-macro-tiles T such that

- there is one-to-one correspondence between ρ and T (the colors of two tiles from ρ match if and only if the macro-colors of the corresponding macro-tiles from T match),
- for every τ-tiling there exists a unique lattice of vertical and horizontal lines that splits this tiling into $N \times N$ macro-tiles from T, i.e., every τ-tiling represents a unique ρ-tiling.

For a large class of sufficiently "well-behaved" sequence of integers N_k we can construct a family of tilesets τ_k $(i = 0, 1, \ldots)$ such that each τ_{k-1} simulates the next τ_k with the zoom N_k (and, therefore, τ_0 simulates every τ_k with the zoom $L_k = N_1 \cdot N_2 \cdots N_k$).

If a k-level macro-tile M is a "cell" in a $(k+1)$-level macro-tile M', we refer to M' as a *father* of M; we call the $(k+1)$-level macro-tiles neighboring M' *uncles* of M.

In our construction each tile of τ_k "knows" its coordinates modulo N_k in the tiling: the colors on the left and on the bottom sides should involve (i, j), the color on the right side should involve $(i+1 \mod N_k, j)$, and the color on the top side, respectively, involves $(i, j+1 \mod N_k)$. So every τ_k-tiling can be uniquely split into blocks (macro-tiles) of size $N_k \times N_k$, where the coordinates of cells range from $(0, 0)$ in the bottom-left corner to $(N-1, N-1)$ in top-right corner. Intuitively, each tile "knows" its position in the corresponding macro-tile.

In addition to the coordinates, each tile in τ_k has some supplementary information encoded in the colors on its sides (the size of the supplementary information is always bounded by $O(1)$). In the middle of each side of a macro-tile we allocate $s_k \ll N_k$ positions where an

array of s_k bits represents a color of a tile from τ_{k+1} (these s_k bits are embedded in colors on the sides of s_k tiles of a macro-tile, one bit per a cell). We fix some cells in a macro-tile that serve as "communication wires" and then require that these tiles carry the same (transferred) bit on two sides (so the bits of "macro-colors" are transferred from the sides of macro-color towards its central part). The central part of a macro-tile (of size, say $m_k \times m_k$, where $m_k = \mathrm{poly}(\log N_k)$) is a *computation zone*; it represents a space-time diagram of a universal Turing machine (the tape is horizontal, time goes up), see Fig. 1.

The first line of the computation zone contains the following *fields* of the *input data*:

- **(i)** the program of a Turing machine π that verifies that a quadruple of macro-colors correspond to one valid macro-color,
- **(ii)** the binary expansion of the integer rank k of this macro-tile,
- **(iii)** the bits encoding the macro-colors – the position inside the "father" macro-tile of rank $(k+1)$ (two coordinates modulo N_{k+1}) and $O(1)$ bits of the supplementary information assigned to the macro-colors.

We require that the simulated computation terminates in an accepting state (if not, no correct tiling can be formed). The simulated computation guarantees that macro-tiles of level k are isomorphic to the tiles of τ_{k+1}. Notice that on each level k of the hierarchy we simulate in macro-tiles a computation of one and the same Turing machine π. Only the inputs for this machine (including the binary expansion of the rank number k) varies on different levels of the hierarchy.

This construction of a tileset can be implemented using the standard technique of self-referential programming, similar to the Kleene recursion theorem, as it is shown in [6]. The construction works if the size of a macro-tile (the zoom factor N_k) is large enough. First, we need enough space in a macro-tile to "communicate" s_k bits from each macro-colors to the computation zone; second, we need a large enough computation zone, so all accepting computations terminate in time m_k and on space m_k. In what follows we assume that $N_k = 3^{C^k}$ for some large enough k.

3 Embedding a bi-infinite sequence into a self-simulating tiling

In this section we adapt the technique from [6] and explain how to "encode" in a self-simulating tiling a bi-infinite sequence, and provide to the computation zones of macro-tiles of all ranks an access to the letters of the embedded sequences.

We are going to embed in our tiling a bi-infinite sequence $\mathbf{x} = (x_i)$ over an alphabet Σ. To this end we assume that each τ-tile "keeps" a letter from Σ that propagates without change in the vertical direction. Formally speaking, a letter from Σ should be a part of the top and bottom colors of every τ-tile (the letters assigned to both sides of a tile must be equal to each other). We want to guarantee that a Σ-sequence can be embedded in a τ-tiling, if and only if it belongs to some fixed effective $\mathcal{A}$ (so far quasiperiodicity is not assumed).

We want to "delegate" the factors of the embedded sequence to the computation zones of macro-tiles, where these factors will be validated (that is, we will check that they do not contain any forbidden subwords). While using tilings with growing zoom factor, we can guarantee that the size of the computation zone of a k-rank macro-tile grows with the rank k. So we have at our disposal the computational resources suitable to run all necessary validation tests on the embedded sequence. It remains to organize the propagation of the letters of the embedded sequence to the "conscious memory" (the computation zones) of macro-tiles of all ranks. In what follows we explain how this propagation is organized.

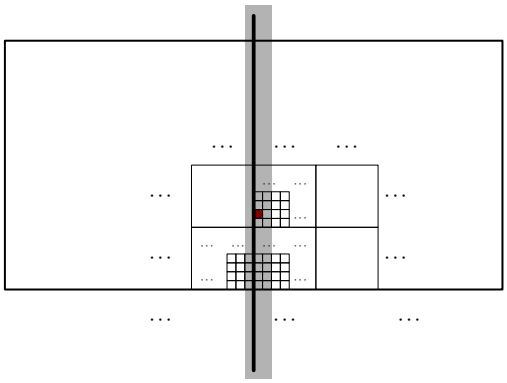


Figure 2 The zone of responsibility (the grey vertical stripe) for a macro-tile (the red square) is 3 times wider than the macro-tile itself.

Zone of responsibility of macro-tiles. In our construction, a macro-tile of level k is a square of size $L_k \times L_k$, with $L_k = N_1 \cdot N_2 \cdot \ldots \cdot N_k$ (where N_i is the zoom factor on level i of the hierarchy of macro-tiles). We say that a k-level macro-tile is *responsible* for the letters of the embedded sequence $\mathbf{x}$ assigned to the columns of (ground level) tiles of this macro-tile as well as to the columns of macro-tiles of the same rank on its left and on its right. That is, the *zone of responsibility* of a k-level macro-tile is a factor of length $3L_k$ from the embedded sequence, see Fig. 2. (The zones of responsibility of any two horizontally neighboring macro-tiles overlap.)

Letters assignment: The computation zone of a k-level macro-tile (of size $m_k \times m_k$) is too small to contain all letters from its zone of responsibility. So we require that the computation zone obtains as an input a (short enough) chunk of letters from its zone of responsibility. Let us say, that it is a factor of length $l_k = \log \log L_k$ from the stripe of $3L_k$ columns constituting the zone of responsibility of this macro-tile. We say that this chunk is *assigned* to this macro-tile.

The infinite stripe of vertically aligned k-level macro-tiles share the same zone of responsibility. However, different macro-tiles in such a stripe will obtain different assigned chunks. The choice of the assigned chunk varies from 0 to $(3L_k - l_k)$. We need to choose a position of a factor of length l_k in a word of length L_k. Let us say for certainty that for a macro-tile M of rank k the first position of the assigned chunk (in the stripe of length $3L_k$) is defined as the vertical position of M in the bigger macro-tile of rank $(k+1)$ (modulo $(3L_k - l_k)$).

▶ Remark. We have chosen the zoom factors N_k so that $N_{k+1} \gg 3L_k$. Hence, every chunk of length l_k from a stripe of width $3L_k$ is assigned to some of the macro-tiles "responsible" for these $3L_k$ letters. Since the zones of responsibility of neighboring k-level macro-tiles overlap by more than l_k, every finite factor of length l_k in the embedded sequence $\mathbf{x}$ is assigned to some k-level macro-tile (even if it involves columns of two macro-tiles of rank k).

Implementing the letters assignment by self-simulation. In the *letters assignment* paragraph above we presented some requirements – how the data must be propagated from the ground level (individual tiles) to k-level macro-tiles. Technically, for each k-level macro-tile $\mathcal{M}$ we specified which chunk of the embedded sequence should be a part of the data fields on the computation zone of $\mathcal{M}$. So far we have not explained how the assigned chunks arrive

to the high-level data fields. Now, we are going to explain how to implement the desired scheme of letter assignment in a self-simulating tiling. Technically, we append to the input data of the computation zones of macro-tiles some supplementary data fields:

(v) the block of l_k letters from the embedded sequence assigned to this macro-tile,

(vi) three blocks of bits of l_{k+1} letters of the embedded sequence assigned to this "father" macro-tile, and two "uncle" macro-tiles (the left and the right neighbors of the "father"),

(vii) the coordinates of the "father" macro-tile in the "grandfather" (of rank $(k+2)$).

Informally, each k-level macro-tile must check that the data in the fields (iv), (v) and (vi) is consistent. That is, if some letters from the fields (iv) and (v) correspond to the same vertical column (in the zone of responsibility), then these letters must be equal to each other. Also, if a k-level macro-tile plays the role of cell in the computation zone of the $(k+1)$-level father, it should check the consistency of its (v) and (vi) with the bits displayed in father's computation zone. Finally, we must ensure the coherence of the fields (v) and (vi) for each pair of neighboring k-level macro-tiles; so this data should make a part of the macro-colors.

Notice that the data from "uncles" macro-tiles is necessary to deal with the letters from the columns that physically belong to the neighboring macro-tiles. So the consistency of the fields (v) is imposed also on neighboring k-level macro-tiles that belong to different $(k+1)$-level fathers (the boarder line between these k-level macro-tiles is also the boarder line between their fathers).

The computations verifying the coherence of the new fields can be performed in polynomial time, and the required update of the construction fits the constraints on the parameter. See a more detailed discussion on "letter delegation" in [6, Section 7].

Final remarks: testing against forbidden factors. To guarantee that the embedded sequence $\mathbf{x}$ contains no forbidden patterns, each k-level macro-tile should allocate some part of its computation zone to enumerate (within the limits of available space and time) the forbidden pattern, and verify that the block of l_k letters assigned to this macro-tile contains none of the found forbidden factors.

The time and space allocated to enumerating the forbidden words grow as a function of k. To ensure that the embedded sequence contains no forbidden patterns, it is enough to guarantee that each forbidden pattern is found by macro-tiles of high enough rank, and every factor of the embedded sequence is compared (on some level of the hierarchy) with every forbidden factor. Thus, we have a general construction of a 2D tiling that simulates a given, effective 1D shift. In the next sections we explain how to make these tilings quasiperiodic in the case when the simulated 1D shift is also quasiperiodic.

4 Combinatorial lemmas: the direct product of quasiperiodic and periodic sequences

The technique from [6] allows to embed in a self-similar tiling a 1-dimensional sequence and handle factors of this sequence. However, the previously known constructions cannot guarantee minimality or even quasiperiodicity of the resulting tiling, even if the embedded sequences have very simple combinatorial structure. To achieve the property of quasiperiodicity we will need some new techniques. The new parts of the argument begins with two simple combinatorial lemmas concerning quasiperiodic sequences.

▶ **Lemma 4.** *(see [2, 15]) Let $\mathbf{x}$ be a bi-infinite recurrent sequence, w be a finite factor in $\mathbf{x}$, and q be a positive integer number. Then there exists an integer $t > 0$ such that another copy of w appears in $\mathbf{x}$ with a shift $q \cdot t$. In other words, there exists another instance of the same*

factor w with a shift divisible by q. Moreover, if $\mathbf{x}$ is quasiperiodic, then the gap $q \cdot t$ between neighboring appearances of w is bounded by some number L that depends on $\mathbf{x}$ and w (but not on a specific instance of the factor x in the sequence).

Notation: For a configuration $\mathbf{x}$ (over some finite alphabet) we denote with $\mathcal{S}(\mathbf{x})$ the shift that consists of all configurations $\mathbf{x}'$ containing only patterns from $\mathbf{x}$. If a shift $\mathcal{T}$ is minimal, then $\mathcal{S}(\mathbf{x}) = \mathcal{T}$ for all configurations $\mathbf{x} \in \mathcal{T}$.

▶ **Lemma 5.** *(a) Let $\mathcal{T}$ be an effective minimal shift. Then for every $\mathbf{x} = (x_i)$ from $\mathcal{T}$ and every periodic configuration $\mathbf{y} = (y_i)$ the direct product $\mathbf{x} \otimes \mathbf{y}$ (the bi-infinite sequence of pairs (x_i, y_i) for $i \in \mathbb{Z}$) generates a minimal shift, i.e., $\mathcal{S}(\mathbf{x} \otimes \mathbf{y})$ is minimal. (b) If in addition the sequence $\mathbf{x}$ are computable, then the set of patterns in $\mathcal{S}(\mathbf{x} \otimes \mathbf{y})$ is also computable.*

▶ Remark. In general, different configurations $\mathbf{x} \in \mathcal{T}$ in the product with one and the same periodic $\mathbf{y}$ can result in different shifts $\mathcal{S}(\mathbf{x} \otimes \mathbf{y})$.

Lemma 5 can be deduced from the fact that for every effective minimal shift the function of quasiperiodicity is computable, [10], and Lemma 4.

5 Towards quasiperiodic SFT

In this section we combine the combinatorial lemmas from the previous section with the technique of enforcing quasiperiodicity from [7], and prove our main results.

5.1 When macro-tiles are clones of each other

To show that (some) self-simulating tilings enjoy the property of quasiperiodicity, we need a tool to prove that every pattern in a tiling has "clones" (equal patterns) in each large enough fragment of this tiling. In our tiling every finite pattern is covered by a block of (at most) four macro-tiles of high enough rank, so we can focus on the search for "clones" in macro-tiles. The following lemma gives a natural characterization of the equality of two macro-tiles in a tiling: they must have the same information in their "conscious memory" (the data written on the tape of the Turing machine in the computation zone) and the same information hidden in their "deep subconscious" (the fragments of the embedded 1D sequence corresponding to the responsibility zones of these macro-tiles must be identical).

▶ **Lemma 6.** *Two macro-tiles of rank k are equal to each other if and only if they (a) contain the same bits in the fields (i) - (vi) in the input data on the computation zone, and (b) the factors of the encoded sequence corresponding to the zones of responsibility of these macro-tiles (in the corresponding vertical stripes of width $3N_k$) are equal to each other.*

Proof. Induction by the rank k. For the macro-tile of rank 1 the statement follows directly from the construction. For a pair of macro-tiles M_1 and M_2 of rank $(k+1)$ with identical data in the fields (i) - (vi) we observe that the corresponding "cells" in M_1 and M_2 (which are macro-tiles of rank k) contain the same data in their own fields (i) - (vi), since the communication wires of M_1 and M_2 carry the same information bits, their computation zones represent exactly the same computations, etc. If the factors (of length $3L_k$) from the encoded sequences in the zones of responsibility of M_1 and M_2 are also equal to each other, we can apply the inductive assumption. ◀

5.2 Supplementary features: constraints that can be imposed on the self-simulating tiling

The tiles involved in our self-simulating tiles set (as well as all macro-tile of each rank) can be classified into three types:

(a) the "skeleton" tiles that keep no information except for their coordinates in the father macro-tile; these tiles work as building blocks of the hierarchical structure;

(b) the "communication wires" that transmit the bits of macro-colors from the border line of the macro-tile to the computation zone;

(c) the tiles of the computation zone (intended to simulate the space-time diagram of the Universal Turing machine).

Each pattern that includes only "skeleton" tiles (or "skeleton" macro-tiles of some rank k) reappears infinitely often in all homologous position inside all macro-tiles of higher rank. Unfortunately, this property is not true for the patterns that involve the "communication zone" or the "communication wires". Thus, the general construction of a fixed-point tiling does not imply the property of quasiperiodicity. To overcome this difficulty we need some new technical tricks.

We can enforce the following additional properties (p1) - (p4) of a tiling with only a minor modification of the construction:

(p1) In each macro-tile, the size of the computation zone m_k is much less than the size of the macro-tile N. In what follows we need to reserve free space in a macro-tile to insert $O(1)$ (some constant number) of copies of each 2×2 pattern from the computation zone (of this macro-tile), right above the computation zone. This requirement is easy to meet. We assume that the size of a computation zone in a k-level macro-tile of size $N_k \times N_k$ is only $m_k = \text{poly}(\log N_k)$. So we can reserve an area of size $\Omega(m_k)$ above the computation zone, which is free of "communication wires" or any other functional gadgets (so far this area consisted of only skeleton tiles).

(p2) We require that the tiling inside the computation zone satisfies the property of 2×2-*determinacy*. If we know all the colors on the borderline of a 2×2-pattern inside of the computation zone (i.e., a tuple of 8 colors), then we can uniquely reconstruct the 4 tiles of this pattern. Again, to implement this property we do not need new ideas; this requirement is met if we represent the space-time diagram of a Turing machine in a natural way.

(p3) The communication channels in a macro-tile (the wires that transmit the information from the macro-color on the borderline of this macro-tile to the bottom line of its computation zone) must be isolated from each other. The distance between every two wires must be greater than 2. That is, each 2×2-pattern can touch at most one communication wire. Since the width of the wires in a k-level macro-tile is only $O(\log N_{k+1})$, we have enough free space to lay the "communication cables", so this requirement is easy to satisfy.

▶ Remark. Property (p3) is a new feature, it was not used in [7] or any other preceding constructions of self-simulating tilings.

(p4) In our construction the macro-colors of a k-level macro-tile are encoded by bit strings of some length $r_k = O(\log N_{k+1})$. We assumed that this encoding is natural in some way. So far the choice of encoding was of small importance; we only required that some natural manipulations with macro-colors can be implemented in polynomial time. Now, we add another (seemingly artificial) requirement: that each of r_k bits encoding the macro-colors (on the top, bottom, left and right sides of a macro-color) was equal to 0 and to 1 for quite a lot of macro-tiles (so the fact that some bit of some macro-color has this or that value,

must not be unique in a tiling). Technically, we require an even stronger property: at every position $s = 1, \dots, r_k$ and for every $i = 0, \dots, N_{k+1} - 1$ there must exist j_0, j_1 such that the s-th bit in the top, the left and the right macro-colors of the k-level macro-tile at the positions (i, j_0) and (i, j_1) in the $(k+1)$-level father macro-tile is equal to 0 and 1 respectively.

There are many (more or less artificial) ways to realize this constraint. For example, we may subdivide the array of r_k bits in three equal zones of size $r_k/3$ and require that for each macro-tile only one of these three zones contains the "meaningful" bits, and two other zones contain only zeros and ones respectively; we require then that the "roles" of these three zones cyclically exchange as we go upwards along a column of macro-tiles.

5.3 Enforcing quasiperiodicity

To achieve the property of quasiperiodicity, we should guarantee that every finite pattern that appears once in a tiling, must appear in each large enough square. If a tileset τ is self-similar, then in every τ-tiling each finite pattern can be covered by at most 4 macro-tiles (by a 2×2-pattern) of an appropriate rank. Thus, it is enough to show that every 2×2-block of macro-tiles of any rank k that appears in at least one τ-tiling, actually appears in this tiling in every large enough square.

Case 1: skeleton tiles. For a 2×2-block of four "skeleton" macro-tiles of level k this is easy. Indeed, we have exactly the same blocks with every vertical shift multiple of L_{k+1} (we have there a similar block of k-level "skeleton" macro-tiles within another macro-tile of rank $(k+1)$). A vertical shift does not change the embedded letters in the zone of responsibility, so we can apply Lemma 6.

To find a similar block of k-level "skeleton" macro-tiles with a different abscissa coordinate, we need a horizontal shift Q which is divisible by L_{k+1} (to preserve the position in the father macro-tile) and at the same time does not change the letters embedded in the zone of responsibility. This is possible due to Lemma 4, if the embedded sequence is quasiperiodic. Given a suitable horizontal shift, we can again apply Lemma 6.

Case 2: communication wires. Let us consider the case when a 2×2-block of k-level macro-tiles involves a part of a communication wire. Due to the property (p3) we may assume that only one wire is involved. The bit transmitted by this wire is either 0 or 1; in both cases, due to the property (p4) we can find another similar 2×2-block of k-level macro-tiles (at the same position within the father macro-tile of rank $(k+1)$ and with the same bit included in the communication wire) in every macro-tile of level $(k+2)$. In this case we need a vertical shift longer than in Case 1: we can find a duplicate of the given block with a vertical shift of size $O(L_{k+2})$.

As in Case 1, any vertical shift does not change the letters embedded in the zone of responsibility of the involved macro-tiles, and we can apply Lemma 6 immediately. If we are looking for a horizontal shift, we again use quasiperiodicity of the simulated shift and apply Lemma 4: there exists a horizontal shift that is divisible by L_{k+2} and does not change the letters embedded in the zone of responsibility. Then we again apply Lemma 6.

Case 3: computation zone. Now we consider the most difficult case: when a 2×2-block of k-level macro-tiles touches the computation zone. In this case we cannot obtain the property of quasiperiodicity for free, and we have to make one more (the last one) modification of our general construction of a self-simulating tiling.

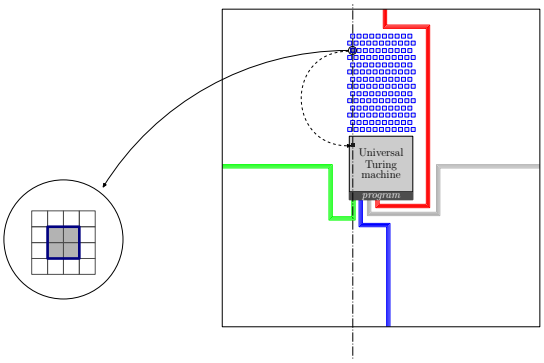


Figure 3 Positions of the slots for patterns from the computation zone.

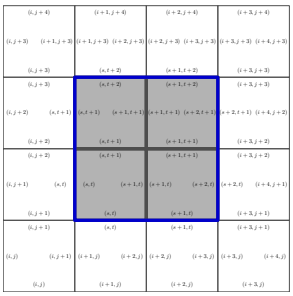


Figure 4 A slot for a 2×2-pattern from the computation zone.

Notice that for each 2×2-window that touches the computation zone of a macro-tile there exist only $O(1)$ ways to tile them correctly. For each possible position of a 2×2-window in the computation zone and for each possible filling of this window by tiles, we reserve a special 2×2-*slot* in a macro-tile, which is essentially a block of size 2×2 in the "free" zone of a macro-tile. It must be placed far away from the computation zone and from all communication wires, but in the same vertical stripe as the "original" position of this block, see Fig. 3. We have enough free space to place all necessary slots due to the property (p1). We define the neighbors around this slot in such a way that only one specific 2×2 pattern can patch it (here we use the property (p2)).

In our construction the tiles around this slot "know" their real coordinates in the bigger macro-tile, while the tiles inside the slot do not (they "believe" they are tiles in the computation zone, while in fact they belong to an artificial isolated *diversity preserving* "slot" far outside of any real computation), see Fig. 3 and Fig. 4. The frame of the slot consists of 12 "skeleton" tiles (the white squares in Fig. 4), they form a slot a 2×2-pattern from the computation zone (the grey squares in Fig. 4). In the picture we show the "coordinates" encoded in the colors on the sides of each tile. Notice that the colors of the bold lines (the blue lines between white and grey tiles and the bold black lines between grey tiles) should contain some information beyond coordinates – these colors involve the bits used to simulate a space-time diagram of the universal Turing machine. In this picture, the "real" coordinates of the bottom-left corner of this slot are $(i+1, j+1)$, while the "natural" coordinates of the pattern (when it appears in the computation zone) are (s, t).

We choose the positions of the "slots" in the macro-tile so that coordinates can be computed with a short program in time polynomial in $\log N$. We require that all slots are isolated from each other in space, so they do not damage the general structure of "skeleton" tiles building the macro-tiles.

Through construction, each of these slots is aligned with the "natural" position of the corresponding 2×2-block in the computation zone. This guarantees that the tiles in the computation zone and their "sibling" in the artificial slots share the same bits of the embedded sequences in the corresponding zone of responsibility. We have defined the slots so that the "conscious memory" of the tiles in the computation zone and in the corresponding slots is the same. Thus, we can apply Lemma 6 and conclude that a 2×2-blocks in diversity preserving slots are exactly equal to the corresponding 2×2-patterns in the computation zone. For a horizontal shift, similarly to the Cases 1–2 above, we use quasiperiodicity of the embedded sequences and apply Lemma 4.

▶ Remark (Concluding Remark). Formally speaking, we proved Lemma 6 before we introduced the last upgrades of our tileset. However, it is easy to verify that the updates of the main construction discussed in this Section do not affect the proof of that lemma.

Thus, we constructed a tileset τ such that every $L_k \times L_k$ pattern that appears in a τ-tiling must also appear in every large enough square in this tiling. So, the constructed tileset satisfies the requirements of Theorem 1.

The proof of Corollary 3. To prove Corollary 3 we only need to combine Theorem 1 with a fact from [14]: there exists a 1D shift $\mathcal{S}$ that is quasiperiodic, and for every configuration $\mathbf{x} \in \mathcal{S}$ the Kolmogorov complexity of all factors is linear, i.e., $K(x_i x_{i+1} \dots x_{i+n}) = \Omega(n)$ for all i. ◀

The proof of Theorem 2. First of all we notice that the proof of Theorem 1 discussed above does not imply Theorem 2. If we take an effective minimal 1D-shift $\mathcal{A}$ and plug it into the construction form the proof of Theorem 1, we obtain a tileset τ (simulating $\mathcal{A}$) which is *quasiperiodic* but not necessary *minimal*. The property of minimality can be lost even for a *periodic* shift $\mathcal{A}$. Indeed, assume that the minimal period $t > 0$ of the configurations in $\mathcal{A}$ is a factor of the size N_k of k-level macro-tiles in our self-simulating tiling, then we can extract from the resulting SFT τ nontrivial shifts T_i, $i = 0, 1, \dots, t-1$ corresponding to the position of the embedded 1D-configuration with respect to the grid of macro-tiles. To overcome this obstacle we will superimpose some additional constraints on the embedding of the simulated $\mathbb{Z}$-shift in a $\mathbb{Z}^2$-tiling. Roughly speaking, we will enforce only "standard" positioning of the embedded 1D sequences with respect to the grid of macro-tiles. This will not change the class of configurations that can be simulated (we still get all configurations from a given minimal shift $\mathcal{A}$), but the classes of all valid tilings will reduce to some minimal $\mathbb{Z}^2$-SFT.

The standardly aligned grid of macro-tiles: In general, the hierarchical structure of macro-tiles permits non-countably many ways of cutting the plane in macro-tiles of different ranks. We fix one particular version of this hierarchical structure and say that a grid of macro-tiles is *standardly aligned*, if for each level k the point $(0, 0)$ is the bottom-left corner of a k-level macro-tile. This means that the tiling is cut into k-level macro-tiles of size $L_k \times L_k$ by vertical lines with abscissae $x = L_k \cdot t'$ and ordinates $y = L_k \cdot t''$, with $t', t'' \in \mathbb{Z}$ (so the vertical line $(0, *)$ and the horizontal line $(*, 0)$ serve as separating lines for macro-tiles of all ranks). Of course, this structure of macro-tiles is computable.

The canonical representative of a minimal shift: A minimal effectively-closed 1D-shift $\mathcal{A}$ is always computable, i.e., the set of finite patterns that appear in configurations of this shift is computable. It follows immediately that $\mathcal{A}$ contains some computable configuration. Let us fix one computable configuration $\mathbf{x}$; in what follows we call it *canonical*.

The standard embedding of the canonical representative: We superimpose the standardly aligned grid of macro-tiles with the canonical representative of a minimal shift $\mathcal{A}$: we take the direct product of the hierarchical structures of the standardly aligned grid of macro-tiles with the canonical configuration $\mathbf{x}$ from $\mathcal{A}$ (that is, each tile with coordinates (i, j) "contains" the letter x_i from the canonical configuration).

Claim 1: Given a pattern w of size $n \le L_k$ and an integer i, we can algorithmically verify whether the factor w appears in the standard embedding of the canonical representative with the shift $(i \mod L_k)$ relative to the grid of k-level macro-tiles. This follows from Lemma 5(b) applied to the superposition of the canonical representative with the periodical grid of k-level macro-tiles).

▶ Remark. This verification procedure is computable, but its computational complexity can be very high. To perform the necessary computation we may need space and time much bigger than the length of w and L_k.

Upgrade of the main construction: Let us update the construction of self-simulating tiling from the proof of Theorem 1. So far we assumed that every macro-tile (of every level k) verifies that the delegated factor of the embedded sequences contains no factors forbidden for the shift $\mathcal{A}$. Now we make the constraint stronger: we require that the delegated factor contains only factors allowed in the shift $\mathcal{A}$ and placed in the positions (relative to the grid of macro-tiles) permitted for factors in the standard embedding of the canonical representative. This property is computable (Claim 1), so every forbidden pattern or a pattern in a forbidden position will be discovered in a computation in a macro-tile of some rank. The computational complexity of this procedure can be very high (see Remark after Claim 1), and we cannot guarantee that the forbidden patterns of small length are discovered by the computation in macro-tiles of small size. But we do guarantee that each forbidden pattern or a pattern in a forbidden position is discovered by a computation in some macro-tile of *high enough* rank.

Claim 2: The new tileset admits correct tilings of the plane. Indeed, at least one tiling is valid by the construction: the standard embedding of the canonical representative corresponds to a valid tiling of the plane, since macro-tiles of all rank never find any forbidden placement of patterns in the embedded sequence.

Claim 3: The new tileset simulates the shift $\mathcal{A}$. This follows immediately from the construction: the embedded sequence must be a configuration from $\mathcal{A}$.

Claim 4: For the constructed tileset τ the set of all tilings is a minimal shift. We need to show that *every* τ-tiling contains all patterns that can appear in *at least one* τ-tiling. Similarly to the proof of Theorem 1, it is enough to prove this property for 2×2-blocks of k-level macro-tile. The difference with the argument in the previous section is that for every 2×2-block of macro-tiles in one tiling T we must find a similar block of macro-tiles in another tiling T', so that this block has exactly the same position with respect to father macro-tile $\mathcal{M}$ of rank $(k + 1)$, and $\mathcal{M}$ and $\mathcal{M}'$ own exactly the same factor of the embedded sequence in their zones of responsibility. This is always possible due to Lemma 5(a) (applied to the canonical representative of $\mathcal{A}$ superimposed with the periodical grid of $(k + 1)$-level macro-tiles). This observation concludes the proof. ◀

Acknowledgments. We are indebted to Emmanuel Jeandel for raising and motivating the questions which led to this work. We are grateful to Gwenaël Richommes and Pascal Vanier for fruitful discussions and to the anonymous reviewers for truly valuable comments.

References

1 Nathalie Aubrun and Mathieu Sablik. Simulation of effective subshifts by two-dimensional subshifts of finite type. *Acta Applicandae Mathematicae*, 128(1):35–63, 2013.
2 Sergey V. Avgustinovich, Dmitrii G. Fon-Der-Flaass, and Anna E. Frid. Arithmetical complexity of infinite words. In *3rd Int. Colloq. on Words, Languages and Combinatorics*, pages 51–62, 2003.
3 Alexis Ballier and Emmanuel Jeandel. Computing (or not) quasiperiodicity functions of tilings. In *2nd Symposium on Cellular Automata* Journées Automates Cellulaires *(JAC 2010)*, pages 54–64, 2010.
4 Bruno Durand. Tilings and quasiperiodicity. *Theoretical Computer Science*, 221(1):61–75, 1999.
5 Bruno Durand, Leonid Levin, and Alexander Shen. Complex tilings. *The Journal of Symbolic Logic*, 73(2):593–613, 2008.
6 Bruno Durand, Andrei Romashchenko, and Alexander Shen. Fixed-point tile sets and their applications. *Journal of Computer and System Sciences*, 78(3):731–764, 2012.
7 Brunourand Durand and Andrei Romashchenko. Quasiperiodicity and non-computability in tilings. In *Proc. International Symposium on Mathematical Foundations of Computer Science (MFCS 2015)*, pages 218–230, 2015.
8 Peter Gács. Reliable computation with cellular automata. *Journal of Computer and System Sciences*, 32(1):15–78, 1986.
9 Gustav Hedlund and Marston Morse. Symbolic dynamics. *American Journal of Mathematics*, 60(4):815–866, 1938.
10 Michael Hochman. On the dynamics and recursive properties of multidimensional symbolic systems. *Inventiones mathematicae*, 176(1):131–167, 2009.
11 Michael Hochman and Pascal Vanier. A note on turing degree spectra of minimal shifts. In *The 12th International Computer Science Symposium in Russia*, pages 154–161, 2017.
12 Emmanuel Jeandel. Personal communication. private communication, 2015.
13 Emmanuel Jeandel and Pascal Vanier. Turing degrees of multidimensional sfts. *Theoretical Computer Science*, 505:81–92, 2013.
14 Andrey Rumyantsev and Maxim Ushakov. Forbidden substrings, kolmogorov complexity and almost periodic sequences. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 396–407, 2006.
15 Pavel V. Salimov. On uniform recurrence of a direct product. *Discrete Mathematics and Theoretical Computer Science*, 12(4), 2010.
16 Linda Brown Westrick. Seas of squares with sizes from a Π^0_1 set. *arXiv preprint arXiv:1609.07411*, 2016.

Parameterized Algorithms for Partitioning Graphs into Highly Connected Clusters*

Ivan Bliznets[1] and Nikolai Karpov[2]

1 **V.A. Steklov Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg, Russia**
2 **V.A. Steklov Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg, Russia**

Abstract

Clustering is a well-known and important problem with numerous applications. The graph-based model is one of the typical cluster models. In the graph model, clusters are generally defined as cliques. However, such an approach might be too restrictive as in some applications, not all objects from the same cluster must be connected. That is why different types of cliques relaxations often considered as clusters.

In our work, we consider a problem of partitioning graph into clusters and a problem of isolating cluster of a special type where by cluster we mean highly connected subgraph. Initially, such clusterization was proposed by Hartuv and Shamir. And their HCS clustering algorithm was extensively applied in practice. It was used to cluster cDNA fingerprints, to find complexes in protein-protein interaction data, to group protein sequences hierarchically into superfamily and family clusters, to find families of regulatory RNA structures. The HCS algorithm partitions graph in highly connected subgraphs. However, it is achieved by deletion of not necessarily the minimum number of edges. In our work, we try to minimize the number of edge deletions. We consider problems from the parameterized point of view where the main parameter is a number of allowed edge deletions. The presented algorithms significantly improve previous known running times for the Highly Connected Deletion (improved from $O^*\left(81^k\right)$ to $O^*\left(3^k\right)$), Isolated Highly Connected Subgraph (from $O^*(4^k)$ to $O^*\left(k^{O\left(k^{2/3}\right)}\right)$), Seeded Highly Connected Edge Deletion (from $O^*\left(16^{k^{3/4}}\right)$ to $O^*\left(k^{\sqrt{k}}\right)$) problems. Furthermore, we present a subexponential algorithm for Highly Connected Deletion problem if the number of clusters is bounded. Overall our work contains three subexponential algorithms which is unusual as very recently there were known very few problems admitting subexponential algorithms.

1998 ACM Subject Classification G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems, I.5.3 Clustering, H.3.3 Information Search and Retrieval

Keywords and phrases clustering, parameterized complexity, highly connected

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.6

1 Introduction

Clustering is a problem of grouping objects such that objects in one group are more similar to each other than to objects in other groups. Clustering has numerous applications, including: machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics. Graph-based model is one of the typical cluster

* This research was supported by Russian Science Foundation (project 16-11-10123)

models. In a graph-based model most commonly cluster is defined as a clique. However, in many applications, such definition of a cluster is too restrictive [17]. Moreover, clique model generally leads to computationally hard problems. For example clique problem is $W[1] - hard$ while s-club problem, with $s \geq 2$, is fixed-parameter tractable with respect to the parameters solution size and s [19]. Because of the two mentioned reasons researchers consider different clique relaxation models [17, 20]. We mention just some of the possible relaxations: s-club(the diameter is less than of equal to s), s-plex (the smallest degree is at least $|G| - s$), s-defective clique (missing s edges to complete graph), γ-quasi-clique ($|E|/\binom{|V|}{2} \geq \gamma$), highly connected graphs (smallest degree bigger than $|G|/2$) and others. With different degree of details all these relaxations were studied: s-club[19, 20], s-plex [14, 1], s-defective clique [21, 7], γ-quasi-clique [18, 16], highly connected graphs [12, 11, 9].

In this work, we study the clustering problem based on highly connected components model. A graph is *highly connected* if the edge connectivity of a graph(the minimum number of edges whose deletion results in a disconnected graph) is bigger than $\frac{n}{2}$ where n is the number of vertices in a graph. An equivalent characterization is for each vertex has degree bigger than $\frac{n}{2}$, it was proved in [3]. One of the reasons for this choice is a huge success in applications of the Highly Connected Subgraphs(HCS) clustering algorithm proposed by Hartuv and Shamir and the second reason is the lack of research for this model compared with the standard clique model. HCS algorithm was used [11] to cluster cDNA fingerprints [8], to find complexes in protein-protein interaction data [10], to group protein sequences hierarchically into superfamily and family clusters [13], to find families of regulatory RNA structures [15].

Hüffner et al. [11] noted that while Hartuv and Shamir's algorithm partitions a graph into highly connected components, it does not delete the minimum number of edges required for such partitioning. That is why they initiated study of the following problem

Highly Connected Deletion
Instance: Graph $G = (V, E)$.
Task: Find edge subset $E' \subseteq E$ of the minimum size such that each connected component of $G' = (V, E \setminus E')$ is highly connected.

For this problem, Hüffner et al. [11] proposed an algorithm which is based on the dynamic programming technique with the running time bounded by $O^*(3^n)$ where n is the number of vertices. For parameterized version of the problem they proposed an algorithm with the running time $O^*(81^k)$ where k is an upper bound on the size of E'. Additionally, they proved that the problem admits a kernel with the size $O(k^{1.5})$. Moreover, they proved conditional lower bound on the running time of algorithms for Highly Connected Deletion , in particular, the problem cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$, $2^{o(n)} \cdot n^{O(1)}$, or $2^{o(m)} \cdot n^{O(1)}$ unless the exponential-time hypothesis (ETH) fails.

Moreover, in another work Hüffner et al. [12] studied a parameterized complexity of related problem of finding highly connected components in a graph.

Isolated Highly Connected Subgraph
Instance: Graph $G = (V, E)$, integer k, integer s.
Task: Is there a set of vertices S such that $|S| = s$, $G[S]$ is highly connected graph and $|E(S, V \setminus S)| \leq k$.

Seeded Highly Connected Edge Deletion
Instance: Graph $G = (V, E)$, subset $S \subseteq V$, integer a, integer k.
Task: Is there a subset of edges $E' \subseteq E$ of size at most k such that $G - E'$ contains only isolated vertices and one highly connected component C with $S \subseteq V(C)$ and $|V(C)| = |S| + a$.

They proposed algorithms with the running time $O^*(4^k)$ and $O^*(16^{k^{3/4}})$ respectively.

Table 1 Results.

Problem	Previous result	Our result
Highly Connected Deletion (exact)	$O^*(3^n)$	$O^*(2^n)$
Highly Connected Deletion (parameterized)	$O^*\left(81^k\right)$	$O^*\left(3^k\right)$
p-Highly Connected Deletion	-	$O^*\left(2^{O\left(\sqrt{pk}\right)}\right)$
Isolated Highly Connected Subgraph	$O^*(4^k)$	$O^*\left(k^{O\left(k^{2/3}\right)}\right)$
Seeded Highly Connected Edge Deletion	$O^*\left(16^{k^{3/4}}\right)$	$O^*\left(k^{\sqrt{k}}\right)$

Our results. We propose algorithms which significantly improve previous upper bounds. Running times of algorithms may be found in a Table 1. We would like to note that three of the algorithms have subexponential running time which is not common. Until very recently there were very few problems admitting subexponential running time. To our mind in algorithm for Isolated Highly Connected Subgraph problem we have an unusual branching procedure as in one branch parameter is not decreasing. However, the value of subsequent decrementation of parameter in this branch is increasing which leads to subexponential running time. We find the fact interesting as we have not met such behavior of branching procedures before. Presented analysis for this case might be useful in further development of subexponential algorithms.

2 Algorithms for partitioning

2.1 Highly Connected Deletion

In this section we present an algorithm for Highly Connected Deletion problem. Our algorithm is based on the fast subset convolution. Let $f, g : 2^X \to \{0, 1, \dots M\}$ be two functions and $|X| = n$. Björklund et al. in [2] proved that function $f * g : 2^X \to \{0, \dots, 2M\}$, where $(f * g)(S) = \min_{T \subseteq S} (f(T) + g(S \setminus T))$, can be computed on all subsets $S \subseteq X$ in time $O(2^n \text{poly}(n, M))$.

▶ **Theorem 1.** *There is a $O^*(2^n)$ time algorithm for* Highly Connected Deletion *problem.*

Proof. Let define function f in the following way

$$f(S) = \begin{cases} |E(S, V \setminus S)| & \text{if } G[S] \text{ is highly connected} \\ \infty & \text{otherwise} \end{cases}$$

Consider function $f^{*k}(V) = \underbrace{f * \cdots * f}_{\text{k times}}$.

Note that $f^{*k}(V) = \min_{S_1 \sqcup \cdots \sqcup S_k = V} (f(S_1) + \cdots + f(S_k))$. Hence, to solve the problem it is enough to find minimum of $f^{*k}(V)$ over all $1 \le k \le n$. Note that if $f^{*k}(V) = \infty$ then it is not possible to partition V into k highly connected components. So if the minimum value of $f^{*k}(V)$ is ∞ then there is no partitioning of G into highly connected components.

Our algorithm contains the following steps.

1. Compute f, i.e. compute value $f(S)$ for all $S \subset V$. It takes $O(2^n(n + m))$ time.
2. Using Björklund et al.[2] algorithm iteratively compute f^{*i} for all $1 \le i \le n$.
3. Find k such that $f^{*k}(V)$ is minimal.

After we perform above steps we will know values of functions f^{*i} on each subset $S \subseteq X$. Let $S_1 \sqcup S_2 \sqcup \cdots \sqcup S_k$ be an optimum partitioning of X into highly connected components. Knowing values of function f^{*k-1} and f it is straightforward to restore S_k in time 2^n. Moreover, knowing f^{*k-1}, S_k we can find value of S_{k-1}. Proceeding this way we obtain the optimum partitioning. As $k \leq n$, we spent at most $O(n2^n)$ time to find all S_i.

It is left to show how to compute all f^{*i} within $O^*(2^n)$ time. The only obstacle why we cannot straightforwardly apply Björklund's algorithm is that f sometimes takes infinite value. It is easy to fix the problem by replacing infinity value with $2m + 1$. We know that each convolution require $O(2^n \text{poly}(n, M))$ time and above we show that we can put M to be equal $2m + 1$. As we need to perform n subset convolutions. So, the running time of second step is $O^*(2^n)$. Hence, the overall running time is $O^*(2^n)$. ◀

Now we consider parameterized version of Highly Connected Deletion problem (one is asked whether it is possible to delete at most k edges and get a vertex disjoint union of highly connected subgraphs).

▶ **Theorem 2.** *There is an algorithm for* Highly Connected Deletion *problem with running time* $O^*(3^k)$.

Proof. Before we proceed with the proof of the theorem we list several simplification rules and lemmas proved by Hüffner et al. in [11].

▶ **Rule 3.** If G contains a connected component C which is highly connected then replace original instance with instance $(G[V \setminus V(C)], k)$.

▶ **Lemma 4.** *Let G be a highly connected graph and $u, v \in V(G)$ be two different vertices from $V(G)$. If $uv \in E$, then $|N(u) \cap N(v)| \geq 1$. If $uv \notin E$ then $|N(u) \cap N(v)| \geq 3$.*

▶ **Rule 5.** If $u, v \in E$ and $N(u) \cap N(v) = \varnothing$ then delete edge uv and decrease parameter k by 1. The obtained instance is $((V, E \setminus \{uv\}), k - 1)$.

▶ **Definition 6.** Let us call vertices u, v *k-connected* if any cut separating these two vertices has size bigger than k.

▶ **Rule 7.** Let S be an inclusion maximal set of pairwise k-connected vertices and $|S| > 2k$. If the induced graph $G[S]$ is not highly connected then our instance is a NO-instance(it is not possible to delete k edges and obtain vertex disjoint union of higly connected subgraphs). Otherwise, we replace original instance with an instance $(G[V \setminus S], k - |E(S, V \setminus S)|)$.

▶ **Lemma 8.** *If G is highly connected then $diam(G) \leq 2$.*

It was shown in [11] that all of the above rules are applicable in polynomial time.

Without loss of generality assume that G is connected. Otherwise, we consider several independent problems. One problem for each connected component. For each connected component we find minimum number of edges that we have to delete in order to partition this component into highly connected subgraphs. Note that in order to find a minimum number for each subproblem we simply consider all possible values of parameter starting from 0 to k.

From Lemma 8 follows that if $dist(u, v)$ (distance between two vertices u, v) is bigger than 2 then in optimal partitioning u and v belong to different connected components. Hence, if $dist(u, v) \geq 3$ then at least one edge from the shortest path between u and v belongs to E'. If $diam(G) > 2$ then it is possible to find two vertices u, v such that $dist(u, v) = 3$. So given the shortest path u, x, y, v we can branch to three instances $(G \setminus ux, k - 1)$, $(G \setminus xy, k - 1)$,

$(G \setminus yv, k-1)$. We apply such branching exhaustively. Finally, we obtain instance with a graph G' of diameter 2.

Now, for our algorithm it is enough to consider a case when graph G has the following properties: (i) $diam(G) \leq 2$; (ii) there are no subsets S of pairwise k-connected vertices with $|S| > 2k$; (iii) G is not highly connected.

From now on we assume that G has above mentioned properties. Suppose $C_1 \sqcup C_2 \sqcup \cdots \sqcup C_\ell$ is an optimum partitioning of G into highly connected graphs and E' is a subset of removed edges. We call vertex *affected* if it is incident with an edge from E'. Otherwise, it is *unaffected*. Denote by U the set of all unaffected vertices and by T the set of all affected vertices. By $C(v)$ we denote a cluster C_i for which $v \in C_i$. Note that for affected vertex u there is vertex v such that $uv \in E(G)$ and $v \notin C(u)$.

▶ **Lemma 9.** *Let G be a graph with diameter 2 then for any optimum partitioning $C_1 \sqcup C_2 \sqcup \cdots \sqcup C_\ell$ of G into highly connected graphs there is an i such that U is contained in C_i.*

Proof. Assume that there are two unaffected vertices $u, v \in U$ and $C(v) \neq C(u)$. Note that any path between u and v must contain an edge from E' and two different edges contained in $C(u), C(v)$ and incident to u and v correspondingly. So, the shortest path between u and v contains at least three edges which contradict our assumption that $diam(G) \leq 2$. Hence, there is an i such that $U \subseteq C_i$. ◀

▶ **Lemma 10.** *Let G be a graph with diameter 2 and optimum partitioning $C_1 \sqcup C_2 \sqcup \cdots \sqcup C_\ell$ into highly connected graphs. If U is not empty then $|E'| \geq n - |C_i|$ where $U \subseteq C_i$.*

Proof. Consider an arbitrary unaffected vertex u. For any $v \in V$ we have $dist(v, u) \leq 2$. Hence, for any $v \notin C(u)$ there is an edge connecting component $C(u)$ with vertex v as otherwise we have $dist(u, v) > 2$. So we have $|E'| \geq n - |C(u)|$. ◀

For any YES-instance we have $k \geq |E'| \geq \frac{|T|}{2}$, $n = |T| + |U|$, and $|U| \leq 2k$.The inequality $|U| \leq 2k$ follows from the simplification Rule 7 and Lemma 9. As otherwise highly connected component which contains U is bigger than $2k$ and hence simplification Rule 7 can be applied which leads to contradiction. So, it means that $n = |T| + |U| \leq 4k$.

Below we present two algorithms. One of these algorithms solves the problem under assumption that optimum partitioning contains at least one unaffected vertex, the other one solves the problem under assumption that all vertices are affected in optimum partitioning. In order to estimate running time of the algorithms we use the following lemma.

▶ **Lemma 11.** *[5] For any non-negative integer a, b we have $\binom{a+b}{b} \leq 2^{2\sqrt{ab}}$.*

At first, consider a case when there is at least one unaffected vertex in optimum partitioning.

▶ **Lemma 12.** *Let G be a connected graph with diameter at most 2. If there is an optimum partitioning $C_1 \sqcup C_2 \sqcup \cdots \sqcup C_\ell$ of G into highly connected graphs such that set of unaffected vertices is not empty then* Highly Connected Deletion *can be solved in $O^*(2^{\frac{3k}{2}})$ time.*

Proof. Let us fix some unaffected vertex u (in algorithm we simply brute-force all n possible values for unaffected vertex u). By Lemma 10 highly connected graph $C(u)$ contains at least $n-k$ vertices. As u is unaffected then $N(u) \subset C(u)$ and $|N(u)| > \frac{|C(u)|}{2}$. Consider set $V \setminus N[u]$. And partition it into two subsets $W_{1,2} \sqcup W_{\geq 3}$, where $W_{1,2} = \{v | 1 \leq |N(u) \cap N(v)| \leq 2\}$, and $W_{\geq 3} = \{v | 3 \leq |N(u) \cap N(v)|\}$. From lemma 4 follows that $W_{1,2} \cap C(u) = \varnothing$. Note that knowing set $C_{part} = C(u) \cap W_{\geq 3}$ we can find set $C(u) = C_{part} \cup N[u]$ and after this simply run algorithm from Theorem 1 on set $V(G) \setminus C(u)$. We implement this approach.

We know that $N[u] \sqcup C_{part} = C(u)$ and $C(u) \leq 2k$. As $|C_{part}| \leq \frac{C(u)}{2}$ it follows that $|C_{part}| \leq k$. Brute-force over all possible values of $s = |C_{part}|$. Having fixed value of s we enumerate all subsets of $W_{\geq 3}$ of size s. All such subsets are potential candidates for a C_{part} role. It is possible to enumerate candidates with polynomial delay i.e. in $O^*(\binom{|W_{\geq 3}|}{|C_{part}|})$ time.

For each listed candidate we run algorithm from Theorem 1. Let $R = W_{\geq 3} \setminus C_{part}$. Hence, the overall running time for a fixed $|C_{part}|$ is bounded by $O^*(2^{|R \cup W_{1,2}|})\binom{|W_{\geq 3}|}{|C_{part}|} = O^*(2^{|R \cup W_{1,2}|})\binom{|C_{part}|+|R|}{|C_{part}|}$. By Lemma 11 we have:

$O^*(2^{|R \cup W_{1,2}|})\binom{|C_{part}|+|R|}{|C_{part}|} = O^*(2^{2\sqrt{|C_{part}||R|}+|R|+|W_{1,2}|})$.

We know that $|C_{part}| \leq k$, $3|R| + |W_{1,2}| \leq k$, hence $O^*(2^{2\sqrt{|C_{part}||R|}+|R|+|W_{1,2}|}) \leq O^*(2^{2\sqrt{k|R|}-2|R|+k})$. The function $g(t) = 2\sqrt{kt} - 2t + k$ attains it maximum when $t = \frac{k}{4}$. So the running time in the worst case is $O^*(2^{1.5k})$. ◀

It is left to construct an algorithm for a case in which all vertices are affected in optimum partitioning. First of all note that if $n \leq 1.57k \leq k \log_2 3$ we can simply run Algorithm 1 and it finds an answer in $O^*(2^n) = O^*(3^k)$ time. Taking into account that all vertices are affected we have that $n \leq 2k$. So we may assume that $1.57k \leq n \leq 2k$.

▶ **Lemma 13.** *Let G be a graph with diameter 2 and $|V(G)| \geq 1.57k$. Moreover, (G, k)* Highly Connected Deletion *problem admits correct partitioning into highly connected components $C_1 \sqcup C_2 \sqcup \cdots \sqcup C_\ell$ such that all vertices are affected in this partitioning. Then there are two highly connected components C_i, C_j such that $|C_i| + |C_j| \geq n - k$.*

Proof. Let E' be set of deleted edges for partitioning $C_1 \sqcup C_2 \sqcup \cdots \sqcup C_\ell$. From $n \geq 1.57k$ follows that in graph $(V(G), E')$ there is a vertex s of degree 1, let $st \in E'$ be the edge. We prove that $C(s), C(t)$ are desired highly connected components. As $diam(G) \leq 2$ then for any vertex $v \in V(G) \setminus C(s) \setminus C(t)$ there is path of length at most 2 from s to v. Hence, any vertex $v \in V(G) \setminus C(s) \setminus C(t)$ should be connected with $C(s) \cup C(t)$ in graph G. As $|E'| \leq k$ then $V(G) \setminus (C(s) \cup C(t)) \leq k$. So $|C(s)| + |C(t)| \geq n - k$. ◀

Now we brute-force all vertices as candidates for a role of vertex s, i.e. vertex of degree 1 in solution E'. Consider two possibilities either $|C(s)| > 2n - 3.14k$ or $|C(s)| \leq 2n - 3.14k$.

Consider the first case, if $|C(s)| > 2n - 3.14k$, then we find solution in $O^*(2^{n - \frac{|C(s)|}{2}}) = O^*(3^k)$ time. In order to do this we consider $deg_G(s)$ cases. Each case correspond to a different edge st incident with s. Such an edge we treat as the only edge incident with s from E'. Having fixed an edge st being from E' we know that all other edges incident with s belong to $E(C(s))$. Denote the set of endpoints of these edges to be U. So we can identify at least $\frac{|C(s)|}{2}$ vertices from $C(s)$. Now we can apply the same technique as in proof of Theorem 1.

We define three functions f, g, h over subsets of $W = V \setminus U$.

- $f(S) = |E(S, W \setminus S)|$ if $G[S]$ is highly connected, otherwise it is equal to ∞.
- $h(S) = \min_i(f^{*i}(S))$.
- $g(S) = 2|E(W \setminus S, U)| + |E(S, W \setminus S)|$ if $G[U \cup S]$ is highly connected otherwise it is ∞.

Let us provide some intuition standing behind the formulas. Value $f(S)$ indicate number of vertices that we have to delete in order to separate highly connected graph $G[S]$. $h(S)$ is a number of edges needed to be deleted in order to separate $G[S]$ into highly connected components. $g(S)$ in some sense is a number of edge deletion needed to create a highly connected component $U \cup S$ which contains vertex s. We show that to solve the problem it is enough to compute $(g * h)(W)$. In similar way to Theorem 1 $(g * h)(W)/2$ equals to a

number of optimum edge deletions. Note that all deleted edges not having endpoints in $C(s)$ will be calculated two times, one for each of its incident highly connected component, see definition of function h. Each edge of E' having an endpoint in U is counted twice in first term of function g. And finally each edge from E' having endpoint in $C(s) \setminus U$ is counted twice, once in second term of the formula of g, and once in the formula of h. So $(g * h)(W)/2$ is required number of edge deletions.

Second case, if $|C(s)| \leq 2n - 3.14k$ then $n - k \leq |C(s)| + |C(t)| \leq 2n - 3.14k + |C(t)|$.

It follows that $|C(t)| + 2n - 3.14k \geq n - k$. Hence, $C(t) \geq 2.14k - n \geq 0.14k$. It means that in $C(t)$ there is a vertex of degree at most 7 in graph $(V(G), E')$. We brute-force all candidates for such vertex and for such edges from E'. Having fixed the candidates, vertex t' and at most seven edges, we identify more than a half vertices from $C(t') = C(t)$ in the following way. All edges incident to t' except just fixed set of candidates belong to $C(t)$. Denote the endpoints of these edges as U_t. In the same way, all edges incident with s except st belong to $C(s)$. Denote by U_s endpoints of edges incident with s except the edge $st \in E'$. Let $U = U_s \cup U_t$. Below we show how to solve obtained problem in $O^*\left(2^{n-\frac{1}{2}(|C(s)|+|C(t)|)}\right)$ time. As in previous case we apply idea similar to algorithm from Theorem 1. Now we present only functions which convolution give an answer. As the further details are identical to Theorem 1.

Our functions are defined over subsets of a set $W = V \setminus U$.

- $f(S) = |E(S, W \setminus S)|$ if $G[S]$ is highly connected, otherwise ∞.
- $h(S) = \min_i \left(f^{*i}(S)\right)$.
- $g_s(S) = 2|E(S, U_t)| + |E(S, W \setminus S)|$ if $G[S \cup U_s]$ is highly connected, otherwise ∞.
- $g_t(S) = 2|E(S, U_s)| + |E(S, W \setminus S)|$ if $G[S \cup U_t]$ is highly connected, otherwise ∞.

The only difference from previous case is that we constructed two functions g_s, g_t instead of just one function g as now we know two halves of two guessed highly connected components. Minimum number of edge deletions in YES-instance separating clusters $C(s), C(t)$ ($U_s \subseteq C(s), U_t \subseteq C(t)$) is $(h * g_s * g_t)(W)/2$. So in this case we need $O^*(2^{|W|})$ running time which is $O^*\left(2^{n-\frac{(n-k)}{2}}\right) = O^*\left(2^{\frac{3k}{2}}\right)$. ◀

2.2 p-Highly Connected Deletion

p-Highly Connected Deletion
Instance: Graph $G = (V, E)$, integer numbers p and k.
Task: Is there a subset of edges $E' \subset E$ of size at most k such that $G - E'$ contains at most p connected components and each component is highly connected?

Our algorithm for p-Highly Connected Deletion is insipired by algorithm for p-Cluster Editing by Fomin et al. [5].

First of all, we prove an upper bound on the number of small cuts in highly connected graph.

▶ **Lemma 14.** *Let $G = (V, E)$ be highly connected graph, $X = \underset{\substack{S \subset V \\ \frac{|V|}{4} \leq |S| \leq \frac{3|V|}{4}}}{\arg\min} |E(S, V \setminus S)|$, and $Y = V \setminus X$, then*

(i) *If $|E(X, Y)| \geq \frac{|V|^2}{100}$ then for any partition of $V = A \sqcup B$ we have $|E(A, B)| \geq \frac{|A| \cdot |B|}{100}$.*

(ii) *If $|E(X, Y)| < \frac{|V|^2}{100}$ then for any partition of $V = A \sqcup B$ we have:*
$|E(A \cap X, B \cap X)| \geq \frac{|X \cap A| \cdot |X \cap B|}{100}$, $|E(A \cap Y, B \cap Y)| \geq \frac{|Y \cap A| \cdot |Y \cap B|}{100}$,
$|E(A, B)| \geq \frac{|X \cap A| \cdot |X \cap B|}{100} + \frac{|Y \cap A| \cdot |Y \cap B|}{100}$.

Proof. *i*) Let $V = A \sqcup B$. Without loss of generality $|A| < |B|$.

If $\frac{|V|}{4} \leq |A|$ then $|E(X,Y)| \leq |E(A,B)|$. Hence, $|E(A,B)| \geq |E(X,Y)| \geq \frac{|V|^2}{100} \geq \frac{|A|\cdot|B|}{100}$.

If $|A| < \frac{|V|}{4}$ then $|E(A,B)| \geq \sum_{v \in A} (\deg(v) - |A|)$. As $\deg(v) > \frac{|V|}{2}$ for all $v \in V(G)$, we have $|E(A,B)| \geq |A| \left(\frac{|V|}{2} - |A|\right) \geq \frac{|A|\cdot|V|}{4} \geq \frac{|A|\cdot|B|}{4} \geq \frac{|A|\cdot|B|}{100}$.

ii) Note that $|E(A,B)| \geq |E(A \cap X, B \cap X)| + |E(A \cap Y, B \cap Y)|$. So it is enough to prove that $|E(A \cap X, B \cap X)| \geq \frac{|A \cap X|\cdot|B \cap X|}{50}$, as the proof of $|E(A \cap Y, B \cap Y)| \geq \frac{|A \cap Y|\cdot|B \cap Y|}{50}$ is analogous. The sum of these two inequalities gives the proof of the theorem.

Without loss of generality $|B \cap X| \leq |A \cap X|$. Hence, $\frac{|V|}{8} \leq |A \cap X|$ and $|B \cap X| \leq \frac{3|V|}{8}$. Consider two cases: $|A \cap X| \geq \frac{|V|}{4}$ and $|A \cap X| < \frac{|V|}{4}$.

Consider case when $|A \cap X| \geq \frac{|V|}{4}$. At first we prove $|E(A \cap X, B \cap X)| \geq |E(B \cap X, Y)|$. It is known that:

$$|E(A \cap X, V \setminus (A \cap X))| = |E(X,Y)| - |E(B \cap X, Y)| + |E(A \cap X, B \cap X)|, \tag{1}$$

$|A \cap X| \geq \frac{|V|}{4}$, and $|V \setminus (A \cap X)| \geq |Y| \geq \frac{|V|}{4}$, it means $|E(A \cap X, V \setminus (A \cap X))| \geq |E(X,Y)|$. The last inequality and (1) imply $|E(A \cap X, B \cap X)| \geq |E(B \cap X, Y)|$. It follows that $2|E(A \cap X, B \cap X)| \geq |E(B \cap X, A \cap X)| + |E(B \cap X, Y)| = |E(B \cap X, V \setminus (B \cap X)|$.

As $\frac{3|V|}{8} \geq |B \cap X|$ and $|E(B \cap X, V \setminus (B \cap X))| \geq |B \cap X| \left(\frac{|V|}{2} - |B \cap X|\right)$ we have $|E(B \cap X, V \setminus (B \cap X))| \geq \frac{|B \cap X|\cdot|V|}{8}$. Hence, $|E(A \cap X, B \cap X)| \geq \frac{|B \cap X|\cdot|V|}{16} \geq \frac{|B \cap X|\cdot|V|}{100}$.

It is left to consider case $|A \cap X| < \frac{|V|}{4}$. Note that $|E(A \cap X, B \cap X)| = |E(A \cap X, V \setminus (A \cap X))| - |E(A \cap X, Y)|$. As $\frac{|V|}{4} > |A \cap X|$ we have $|E(A \cap X, V \setminus (A \cap X))| \geq |A \cap X| \left(\frac{|V|}{2} - |A \cap X|\right) \geq \frac{|V|}{8} \cdot \frac{|V|}{4} \geq \frac{|V|^2}{32}$. We know that $|E(A \cap X, Y)| \leq |E(X,Y)| \leq \frac{|V|^2}{100}$, hence $|E(A \cap X, B \cap X) \geq \frac{|V|^2}{32} - \frac{|V|^2}{100} > \frac{|V|^2}{50} \geq \frac{|A \cap X|\cdot|B \cap X|}{100}$. ◀

▶ **Definition 15.** A partition of $V = V_1 \sqcup V_2$ is called a k-cut of G if $|E(V_1, V_2)| \leq k$.

The following lemma limits number of k-cuts in a disjoint union of highly connected graphs.

▶ **Lemma 16.** *If $G = (V, E)$ is a union of p disjoint highly connected components and $p \leq k$ then the number of k-cuts in G is bounded by $2^{O\left(\sqrt{pk}\right)}$.*

Proof. Let G be a disjoint union of highly connected components $C_1, \ldots, C_p$. For each C_i we consider sets X_i, Y_i where $E(X_i, Y_i)$ is a minimum cut of C_i and $C_i = X_i \sqcup Y_i$. We construct a new partition $C'_1, \ldots, C'_q$ of $V(G)$. The new partition is obtained from partition $C_1 \sqcup \ldots \sqcup C_p$ in the following way: if $|E(X_i, Y_i)| < |C_i^2|/100$ then we split C_i into two sets X_i, Y_i otherwise we take C_i without splitting. Note that $p \leq q \leq 2p$ as we either split C_i into to parts or leave it as is.

We bound number of k-cuts of graph G in two steps. In first step we bound number of cuts V_1, V_2 such that $|V_1 \cap C'_i| = x_i$ and $|V_2 \cap C'_i| = y_i$ where x_i, y_i are some fixed integers. In second step we bound number of tuples $(x_1, \ldots, x_q, y_1, \ldots, y_q)$ for which there is at least one k-cut V_1, V_2 satisfying conditions $|V_1 \cap C'_i| = x_i$, $|V_2 \cap C'_i| = y_i$.

If x_i, y_i are fixed and $x_i + y_i = |C'_i|$ the number of partitions of C'_i is equal to $\binom{x_i+y_i}{x_i}$. Note that by Lemma 11 we have $\binom{x_i+y_i}{x_i} \leq 2^{\sqrt{x_i y_i}}$. Observe that there are at least $\frac{x_i y_i}{100}$ edges between $V_1 \cap C'_i$ and $V_2 \cap C'_i$ by Lemma 14. So if $V_1 \sqcup V_2$ is partition of V then $\sum_{i=1}^{q} x_i y_i \leq 100k$.

Applying Cauchy–Schwarz inequality we infer that $\sum_{i=1}^{q} \sqrt{x_i y_i} \leq \sqrt{q} \cdot \sqrt{\sum_{i=1}^{q} x_i y_i} \leq \sqrt{200pk}$. Therefore, the number of considered cuts is at most $\prod_{i=1}^{q} \binom{x_i+y_i}{x_i} \leq 2^{2\sum_{i=1}^{q}\sqrt{x_i y_i}} \leq 2^{\sqrt{800pk}}$.

Now we show bound for a second step i.e. number of possible tuples $(x_1, \ldots, x_q, y_1, \ldots, y_q)$ generating at least one k-cut. Note that $\min\{x_i, y_i\} \leq \sqrt{x_i y_i}$. Hence, $\sum_{i=1}^{q} \min(x_i, y_i) \leq \sqrt{100qk}$. Tuple $(x_1, \ldots, x_q, y_1, \ldots, y_q)$ can be generated in the following way: at first we choose which value is smaller x_i or y_i. Then we express $\sqrt{\lfloor 100qk \rfloor}$ as a sum of $q+1$ non-negative numbers: $\min\{x_i, y_i\}$ for $1 \leq i \leq q$ and the rest $\sqrt{\lfloor 100qk \rfloor} - \sum_{i=1}^{q} \min(x_i, y_i)$.

The number of choices in the first step of generation is equal to $2^q \leq 2^{\sqrt{2qk}}$, and number of ways to expreess $\sqrt{100qk}$ as a sum of $q+1$ number is at most $\binom{\sqrt{100qk}+q+1}{q} \leq 2^{\sqrt{100qk}+q+1} \leq 2^{\sqrt{100qk}+\sqrt{2qk}+1}$. Therefore, the total number of partitions is bounded by $2^{c\sqrt{pk}}$ for some constant c. ◀

The last ingredient for our algorithm is the following lemma proved by Fomin et al.[5]

▶ **Lemma 17.** *[5] All cuts (V_1, V_2) such that $|E(V_1, V_2)| \leq k$ of a graph G can be enumerated with polynomial time delay.*

Now we are ready to present a final theorem.

▶ **Theorem 18.** *There is a $O^*(2^{O(\sqrt{pk})})$ time algorithm for p-*Highly Connected Deletion *problem.*

Proof. First of all we solve the problem in case of connected graph. Denote by $\mathcal{N}$ set of all k-cuts in graph G. All elements of set $\mathcal{N}$ can be enumerated with a polynomial time delay. If G is a union of p clusters plus some edges then the size of $\mathcal{N}$ is bounded by $2^{c\sqrt{pk}}$ by Lemma 16 (as additional edges only decrease number of k-cuts). Thus, we enumerate $\mathcal{N}$ in time $O^*(2^{O(\sqrt{pk})})$. If we exceed the bound $2^{c\sqrt{pk}}$ given by Lemma 16 we know that we can terminate our algorithm and return answer `NO`. So we may assume that we enumerate the whole $\mathcal{N}$ and it contains at most $2^{c\sqrt{pk}}$ elements.

We construct a directed graph D, whose vertices are elements of a set $\mathcal{N} \times \{0, 1, \ldots, p\} \times \{0, 1, \ldots, k\}$, note that $|V(D)| = 2^{O(\sqrt{pk})}$. We add arcs going from $((V_1, V_2), j, l)$ to $((V_1', V_2'), j+1, l')$, where $V_1 \subset V_1'$, $G[V_1' \setminus V_1]$ is highly connected graph, $j \in \{0, 1, \ldots, p-1\}$, and $l' = l + |E(V_1, V_1' \setminus V_1)|$. The arcs can be constructed in $2^{O(\sqrt{pk})}$ time. We claim that the answer for an instance (G, p, k) is equivalent to existence of path from a vertex $((V, \varnothing), 0, 0)$ to a vertex $((\varnothing, V), p', k')$ for some $p' \leq p, k' \leq k$.

In one direction, if there is a path from $((\varnothing, V), 0, 0)$ to $((V, \varnothing), p', k')$ for some $k' \leq k$ and $p' \leq p$, then the consecutive sets $V_1' \setminus V_1$ along the path form highly connected components. Moreover, number of deleted edges from G is equal to last coordinate which is smaller than k.

Let us prove the opposite direction. Let assume that we can delete at most k edges and get a graph with highly connected components $C_1, \ldots, C_p$. Let us denote $T_i = \cup_{j<i} V(C_i)$, $l_{i+1} = l_i + |E(T_{i+1} \setminus T_i, T_i)|$ then the vertices $((T_i, V \setminus T_i), i-1, l_i)$ constitute desired path in graph D.

Reachability in a graph can be tested in a linear time with respect to the number of vertices and arcs. To concude the algorithm we simply test the reachability in the graph D.

It is left co consider a case when G is not connected. Let assume that G consist of q connected components $C_1, \ldots, C_q$ then for each connected component C_i we find all $p' \leq p$ and $k' \leq k$ such that (C_i, p', k') is YES-instance. After this we construct auxiliary directed graph Q with a set of vertices $\{0, \ldots, q\} \times \{0, \ldots, p\} \times \{0, \ldots, k\}$. We add arcs going from (i, a, b) to $(i + 1, a + p', b + k')$ if (C_i, p', k') is a YES-instance. Using similar arguments as before it could be shown that reachability of vertex (q, p', k') from vertex $(0, 0, 0)$ is equivalent to possibility delete k' edges and get p' highly connected components. ◀

3 Algorithms for finding a subgraph

3.1 Seeded Highly Connected Edge Deletion

Seeded Highly Connected Edge Deletion
Instance: Graph $G = (V, E)$, subset $S \subseteq V$ and integer numbers a and k.
Task: Is there a subset of edges $E' \subseteq E$ of size at most k such that $G - E'$ contains only isolated vertices and one highly connected component C with $S \subseteq V(C)$ and $\lvert V(C)\rvert = \lvert S\rvert + a$.

Hüffner et al. [12] constructed an algorithm with running time $O(16^{k^{0.75}} + k^2nm)$ for Seeded Highly Connected Edge Deletion problem. We improve the result to $O^*\left(2^{O(\sqrt{k}\log k)}\right)$ time algorithm.

▶ **Theorem 19.** *There is $O^*(2^{O(\sqrt{k}\log k)})$ time algorithm for* Seeded Highly Connected Edge Deletion *problem.*

3.2 Isolated Highly Connected Subgraph

Isolated Highly Connected Subgraph
Instance: Graph $G = (V, E)$, integer k, integer s.
Task: Is there a set of vertices S such that $\lvert S\rvert = s$, $G[S]$ is highly connected graph and $\lvert E(S, V \setminus S)\rvert \leq k$.

Hüffner et al. [12] proposed $O^*(4^k)$ algorithm for Isolated Highly Connected Subgraph problem, in this work we construct subexponential algorithm for the same problem with running time $O^*(k^{O(k^{2/3})})$.

In order to solve Isolated Highly Connected Subgraph problem Hüffner et al. in [12] constructed algorithm for a more general problem:

f-Isolated Highly Connected Subgraph
Instance: Graph $G = (V, E)$, integer k, integer s, function $f : V \to \mathbb{N}$.
Task: Is there a set of vertices S such that $\lvert S\rvert = s$, $G[S]$ is highly connected and $\lvert E(S, V \setminus S)\rvert + \sum_{v \in S} f(v) \leq k$.

Our algorithm uses reduction rules proposed in [12]. Here, we state the reduction rules without proof, as the proofs can be found in [12].

▶ **Rule 20.** If G contains connected component C of size smaller than s then delete C i.e. solve instance $(G \setminus C, f, k)$.

▶ **Rule 21.** Let G contains connected component $C = (V', E')$ with minimal cut bigger than k. If C is highly connected graph, $\lvert V'\rvert = s$ and $\sum_{s \in V'} f(s) \leq k$ then output a trivial

YES-instance otherwise remove C, i.e. consider instance $(G \setminus C, f, k)$ of f-Isolated Highly Connected Subgraph problem.

▶ **Rule 22.** Let G contains connected component C with minimal cut (A, B) of size at most $\frac{s}{2}$. We define function f' in the following way: for each vertex $v \in A$ $f'(v) := f(v) + |N(v) \cap B|$ and for each $v \in B$ we let $f'(v) := f(v) + |N(v) \cap A|$. Replace original instance with an instance $(G \setminus E(A, B), f', k)$.

▶ **Lemma 23.** *Rules 20, 21, 22 can be exhaustively applied in time $O((sn + k)m)$. If rules 20, 21, 22 are not applicable then $k > \frac{s}{2}$.*

We also use following Fomin and Villanger's result.

▶ **Proposition 24.** [6] For each vertex v in graph G and integers $b, f \geq 0$ number of connected induced subgraphs $B \subseteq V(G)$ satisfying the following properties $v \in B$, $|B| = b + 1$, $|N(B)| = f$; is at most $\binom{b+f}{b}$. Moreover, all these sets can be enumerated in time $O\left(\binom{b+f}{b}(n + m)b(b + f)\right)$.

Now we have all ingredients for out algorithm.

▶ **Theorem 25.** f-Isolated Highly Connected Subgraph *can be solved in time $2^{O(k^{2/3} \log k)}$.*

Proof. First of all we exhaustively apply reduction rules 20, 21, 22. From Lemma 23 follows that we may assume $2k > s$. We consider two cases either $k^{2/3} < s$ or $k^{2/3} \geq s$.

Case 1: $s \leq k^{2/3}$. Enumerate all induced connected subgraphs $G' = (V', E')$ such that $|V'| = s$ and $N(V') \leq k$. If desired S exists than it is among enumerated sets. From Proposition 24 follows that number of such sets is at most $nkO^*(\binom{s+k}{s})$. As $s < 2k$ and $s < k^{2/3}$ we have $nkO^*(\binom{s+k}{s}) \leq O^*((s+k)^s) \leq O^*(2^{k^{2/3} \log k})$. Hence, in time $O^*(2^{k^{2/3} \log k})$ we can enumerate all potential candidates S'. For each candidate we check in polynomial time whether $G[S']$ is highly connected and $|E(S', V \setminus S')| + \sum\limits_{v \in S'} f(v) \leq k$.

Case 2: $k^{\frac{2}{3}} < s$. Let set S be a solution. Define edge set $E' = E(S, V \setminus S)$. Consider function $d : S \to \mathbb{N}$ where $d(v) = |N(v) \cap (V \setminus S)|$. As $\sum\limits_{v \in S} d(v) = |E(S, V \setminus S)| \leq k$ then there is a vertex $v \in S$ such that $d(v) \leq \frac{k}{s} < k^{\frac{1}{3}}$. Note that for such v we have $|N(v)| = |N(v) \cap S| + |N(v) \setminus S| \leq s + k^{\frac{1}{3}}$. We branch on possible values of such vertex and a set of its neighbors that do not belong to S. In order to do this we have to consider at most $n \sum\limits_{i \leq k^{1/3}} \binom{s+k^{1/3}}{i} \leq nk^{1/3}2^{2\sqrt{(s+k^{1/3}-i)i}} \leq nk^{1/3}2^{2\sqrt{3k^{4/3}}} = n2^{O(k^{2/3})}$ cases. Knowing vertex $v \in S$ and $N(v) \setminus S$ we find $N(v) \cap S$. So we already identified at least $\frac{s}{2} + 1$ vertices from S, let denote this set by W. Now we start branching procedure that in right branch extend set W into a solution set S. Branching procedure takes as an input tuple (G, k, s', W, B) where W is a set of vertices determined to be in solution S, B is a set of vertices determined to be not in solution, k number of allowed edge deletions, $s' = s - |W|$ number of vertices that is left to add. The procedure pick a vertex $w \notin W \cup B$ and consider two cases either $w \in S, w \notin B$ or $w \notin S, w \in B$. The first call of the procedure is performed on tuple $(G, k - |E(W, N(v) \setminus W)|, s - |W|, W, \varnothing)$.

Consider arbitrary vertex $x \in V \setminus (W \cup B)$. If $x \in S$ then $|N(x) \cap S| \geq \frac{s}{2}$. Hence, $|N(x) \cap W| \geq |N(x) \cap S| - |S \setminus W| \geq \frac{s}{2} - (s - |W|) = |W| - \frac{s}{2}$. So any vertex x such that

$|N(x) \cap W| < |W| - \frac{s}{2}$ cannot belong to solution S and we safely put x to B. Otherwise, we run our procedure on tuples $(G, k - |N(x) \cap B|, s' - 1, W \cup x, B)$ and $(G, k - |N(x) \cap W|, s', W, B \cup x)$. Note that we stop computation in a branch if $k' \leq 0$ or $s' = 0$. It is easy to see that the algorithm is correct.

It is left to determine the running time of the algorithm. Note that procedure contains two parameters k and s'. In one branch we decrease value of s' by one in the other branch we decrease value of k by $E(x, W)$. Note that in first branch we not only decrease value of s' but we also increase a lower bound on $|N(x) \cap W|$ by 1 as $|N(x) \cap W| \geq |W| - \frac{s}{2}$.

Let us consider a path $(x_1, x_2, \dots x_l)$ from root to leaf in our branching tree. To each node we assign a vertex x_i on which we are branching at this node. For each such path we construct unique sequence $a_1, a_2, \dots, a_m$ and a number b. We put b equal to the number of vertices from set $\{x_1, x_2, \dots, x_l\}$ that was assigned to solution S. And $a_i - 1$ is a number of vertices that was assigned to W in a sequence $x_1, x_2, \dots x_j$ where x_j is an $i-$th vertex assigned to B in this sequence. Note that $|N(x_j) \cap W| \geq a_i$, so $\sum_i a_i \leq k$. Note that for any path from root to leaf we can construct a corresponding sequence a_i and number b. Moreover, any sequence $a_1, a_2, \dots a_m$ and number b correspond to at most one path from root to node.

▶ **Proposition 26.** Given number b and non-decreasing sequence $a_1, a_2, \dots, a_m$ we can uniquely determine a corresponding path in a branching tree.

Proof. For a notation convenience we let $a_0 = 1$. For $1 \leq i \leq m$ we perform the following operation: we make $a_i - a_{i-1}$ steps of assigning vertices to a solution set, i.e. to set W and make one step in branch assigning vertex to a set B. After m such iterations we perform $b - m$ steps of assigning vertices to solution. As $a_1, a_2, \dots a_m$ is non-decreasing sequence we have constructed a unique path in branching tree. It is easy to see that the original sequence $a_1, \dots, a_m$ and number b correspond to a constructed path. So for each path from root to leaf there is a corresponding sequence and for each sequence with a number there is at most one corresponding path from root to node in a tree. ◀

▶ **Lemma 27.** *The number of tuples $(a_1, \dots, a_m, b)$ where $0 \leq b \leq s$, $1 \leq a_i \leq a_{i+1}$ for $i < m$, and $\sum_i a_i \leq k$ is bounded by $O^*\left(2^{O(\sqrt{k})}\right)$*

Proof. For fixed l, tuples $(a_1, \dots, a_m)$ such that $\sum_i a_i = l$ are well-known and are called partitions of l. Pribitkin [4] gave a simple upper bound $e^{2.57\sqrt{l}}$ on the number of partitions of l. Hence, number of tuples $(a_1, \dots, a_m)$ is bounded by $\sum_{i=0}^{k} e^{2.57\sqrt{i}} \leq (k+1)e^{2.57\sqrt{k}}$. Moreover, we know that $0 \leq b \leq s$. It means that the number of tuples $(a_1, \dots, a_m, b)$ is bounded by $(s+1)(k+1)2^{O(\sqrt{k})}$. ◀

From Proposition 26 and Lemma 27 follows that the number of nodes in a branching tree is at most $s2^{O(\sqrt{k})}$. Hence, the running time of the procedure is at most $s2^{O(\sqrt{k})}$.

Now, we compute required time for algorithm in this case(case 2). At first, we branch on a vertex and its neighbors from solution set S. We did it by creating at most $O^*\left(2^{O(k^{2/3})}\right)$ subcases. In each subcase we run a procedure with running time $O^*\left(2^{O(\sqrt{k})}\right)$. So, the overall runnning time equals to $O^*\left(2^{O(\sqrt{k})}2^{O(k^{2/3})}\right) = O^*\left(2^{O(k^{2/3})}\right)$.

The worst running time has **Case 1**, so the running time of the whole algorithms is $O^*\left(k^{O(k^{2/3})}\right)$. ◀

References

1 Balabhaskar Balasundaram, Sergiy Butenko, and Illya V Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, 2011.

2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74, 2007. doi:10.1145/1250790.1250801.

3 Gary Chartrand. A graph-theoretic approach to a communications problem. *SIAM Journal on Applied Mathematics*, 14(4):778–781, 1966.

4 Wladimir de Azevedo Pribitkin. Simple upper bounds for partition functions. *The Ramanujan Journal*, 18(1):113–119, 2009. doi:10.1007/s11139-007-9022-z.

5 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014. doi:10.1016/j.jcss.2014.04.015.

6 Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. *Combinatorica*, 32(3):289–308, 2012. doi:10.1007/s00493-012-2536-z.

7 Jiong Guo, Iyad A Kanj, Christian Komusiewicz, and Johannes Uhlmann. Editing graphs into disjoint unions of dense clusters. *Algorithmica*, 61(4):949–970, 2011.

8 Erez Hartuv, Armin O Schmitt, Jörg Lange, Sebastian Meier-Ewert, Hans Lehrach, and Ron Shamir. An algorithm for clustering cdna fingerprints. *Genomics*, 66(3):249–256, 2000.

9 Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Inf. Process. Lett.*, 76(4-6):175–181, 2000. doi:10.1016/S0020-0190(00)00142-3.

10 Wayne Hayes, Kai Sun, and Nataša Pržulj. Graphlet-based measures are suitable for biological network comparison. *Bioinformatics*, 29(4):483–491, 2013.

11 Falk Hüffner, Christian Komusiewicz, Adrian Liebtrau, and Rolf Niedermeier. Partitioning biological networks into connected clusters with maximum edge coverage. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 11(3):455–467, 2014. doi:10.1109/TCBB.2013.177.

12 Falk Hüffner, Christian Komusiewicz, and Manuel Sorge. Finding highly connected subgraphs. In *SOFSEM 2015: Theory and Practice of Computer Science - 41st International Conference on Current Trends in Theory and Practice of Computer Science, Pec pod Sněžkou, Czech Republic, January 24-29, 2015. Proceedings*, pages 254–265, 2015. doi:10.1007/978-3-662-46078-8_21.

13 Antje Krause, Jens Stoye, and Martin Vingron. Large scale hierarchical clustering of protein sequences. *BMC bioinformatics*, 6(1):15, 2005.

14 Hannes Moser, Rolf Niedermeier, and Manuel Sorge. Algorithms and experiments for clique relaxations—finding maximum s-plexes. In *International Symposium on Experimental Algorithms*, pages 233–244. Springer, 2009.

15 Brian J Parker, Ida Moltke, Adam Roth, Stefan Washietl, Jiayu Wen, Manolis Kellis, Ronald Breaker, and Jakob Skou Pedersen. New families of human regulatory rna structures identified by comparative analysis of vertebrate genomes. *Genome research*, 21(11):1929–1943, 2011.

16 Jeffrey Pattillo, Alexander Veremyev, Sergiy Butenko, and Vladimir Boginski. On the maximum quasi-clique problem. *Discrete Applied Mathematics*, 161(1):244–257, 2013.

17 Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013.

18 Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013. doi:10.1016/j.ejor.2012.10.021.

19 Alexander Schäfer. *Exact algorithms for s-club finding and related problems.* PhD thesis, Friedrich-Schiller-University Jena, 2009.

20 Shahram Shahinpour and Sergiy Butenko. Distance-based clique relaxations in networks: s-clique and s-club. In *Models, algorithms, and technologies for network analysis*, pages 149–174. Springer, 2013.

21 Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006.

Hypercube LSH for Approximate near Neighbors

Thijs Laarhoven*

IBM Research, Rüschlikon, Switzerland
mail@thijs.com

Abstract

A celebrated technique for finding near neighbors for the angular distance involves using a set of *random* hyperplanes to partition the space into hash regions [Charikar, STOC 2002]. Experiments later showed that using a set of *orthogonal* hyperplanes, thereby partitioning the space into the Voronoi regions induced by a hypercube, leads to even better results [Terasawa and Tanaka, WADS 2007]. However, no theoretical explanation for this improvement was ever given, and it remained unclear how the resulting hypercube hash method scales in high dimensions.

In this work, we provide explicit asymptotics for the collision probabilities when using hypercubes to partition the space. For instance, two near-orthogonal vectors are expected to collide with probability $(\frac{1}{\pi})^{d+o(d)}$ in dimension d, compared to $(\frac{1}{2})^d$ when using random hyperplanes. Vectors at angle $\frac{\pi}{3}$ collide with probability $(\frac{\sqrt{3}}{\pi})^{d+o(d)}$, compared to $(\frac{2}{3})^d$ for random hyperplanes, and near-parallel vectors collide with similar asymptotic probabilities in both cases.

For c-approximate nearest neighbor searching, this translates to a decrease in the exponent ρ of locality-sensitive hashing (LSH) methods of a factor up to $\log_2(\pi) \approx 1.652$ compared to hyperplane LSH. For $c = 2$, we obtain $\rho \approx 0.302 + o(1)$ for hypercube LSH, improving upon the $\rho \approx 0.377$ for hyperplane LSH. We further describe how to use hypercube LSH in practice, and we consider an example application in the area of lattice algorithms.

1998 ACM Subject Classification F.2 Analysis of algorithms and problem complexity, G.3 Probability and statistics, H.3 Information storage and retrieval

Keywords and phrases (approximate) near neighbors, locality-sensitive hashing, large deviations, dimensionality reduction, lattice algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.7

1 Introduction

Finding (approximate) near neighbors. A key computational problem in various research areas, including machine learning, pattern recognition, data compression, coding theory, and cryptanalysis [34, 11, 15, 16, 29, 23], is finding near neighbors: given a data set $\mathcal{D} \subset \mathbb{R}^d$ of cardinality n, design a data structure and preprocess $\mathcal{D}$ in a way that, when given a query vector $\boldsymbol{q} \in \mathbb{R}^d$, one can efficiently find a near point to $\boldsymbol{q}$ in $\mathcal{D}$. Due to the "curse of dimensionality" [18] this problem is known to be hard to solve exactly (in the worst case) in high dimensions d, so a common relaxation of this problem is the (c, r)-*approximate near neighbor problem* ((c, r)-ANN): given that the nearest neighbor lies at distance at most r from $\boldsymbol{q}$, design an algorithm that finds an element $\boldsymbol{p} \in \mathcal{D}$ at distance at most $c \cdot r$ from $\boldsymbol{q}$.

Locality-sensitive hashing (LSH) and filtering (LSF). A prominent class of algorithms for finding near neighbors in high dimensions is formed by locality-sensitive hashing (LSH) [18] and locality-sensitive filtering (LSF) [9]. These solutions are based on partitioning the space

* The author is supported by the SNSF ERC Transfer Grant CRETP2-166734 FELICITY.

into regions, in a way that nearby vectors have a higher probability of ending up in the same hash region than distant vectors. By carefully tuning (i) the number of hash regions per hash table, and (ii) the number of randomized hash tables, one can then guarantee that with high probability (a) nearby vectors will *collide* in at least one of the hash tables, and (b) distant vectors will not collide in any of the hash tables. For LSH, a simple lookup in all of $\boldsymbol{q}$'s hash buckets then provides a fast way of finding near neighbors to $\boldsymbol{q}$, while for LSF the lookups are slightly more involved. For various metrics, LSH and LSF currently provide the best performance in high dimensions [8, 9, 7, 13].

Near neighbors on the sphere. In this work we will focus on the near neighbor problem under the *angular distance*, where two vectors $\boldsymbol{x}, \boldsymbol{y}$ are considered nearby iff their common angle θ is small [12, 35, 33, 5]. This equivalently corresponds to near neighbor searching for the ℓ_2-norm, where the entire data set is assumed to lie on a sphere. A special case of (c, r)-ANN on the sphere, often considered in the literature, is the *random* case $r = \frac{1}{c}\sqrt{2}$ and $c \cdot r = \sqrt{2}$, in part due to a reduction from near neighbor under the Euclidean metric for general data sets to (c, r)-ANN on the sphere with these parameters [8].

1.1 Related work

Upper bounds. Perhaps the most well-known and widely used solution for ANN for the angular distance is Charikar's hyperplane LSH [12], where a set of *random* hyperplanes is used to partition the space into regions. Due to its low computational complexity and the simple form of the collision probabilities (with no hidden order terms in d), this method is easy to instantiate in practice and commonly achieves the best performance out of all LSH methods when d is not too large. For large d, both spherical cap LSH [6, 8] and cross-polytope LSH [36, 17, 5, 21] are known to perform better than hyperplane LSH. Experiments from [36, 37] showed that using *orthogonal* hyperplanes, partitioning the space into Voronoi regions induced by the vertices of a hypercube, also leads to superior results compared to hyperplane LSH; however, no theoretical guarantees for the resulting hypercube LSH method were given, and it remained unclear whether the improvement persists in high dimensions.

Lower bounds. For the case of *random* data sets, lower bounds have also been found, matching the performance of spherical cap and cross-polytope LSH for large c [30, 32, 5]. These lower bounds are commonly in a model where it is assumed that collision probabilities are "not too small", and in particular not exponentially small in d. Therefore it is not clear whether one can further improve upon cross-polytope LSH when the number of hash regions is exponentially large, which would for instance be the case for hypercube LSH. Together with the experimental results from [36, 37], this naturally begs the question: how efficient is hypercube LSH? Is it better than hyperplane LSH and/or cross-polytope LSH? And how does hypercube LSH compare to other methods in practice?

1.2 Contributions

Hypercube LSH. By carefully analyzing the collision probabilities for hypercube LSH using results from large deviations theory, we show that hypercube LSH is indeed different from, and superior to hyperplane LSH for large d. The following main theorem states the asymptotic form of the collision probabilities when using hypercube LSH, which are also visualized in Figure 1 in comparison with hyperplane LSH.

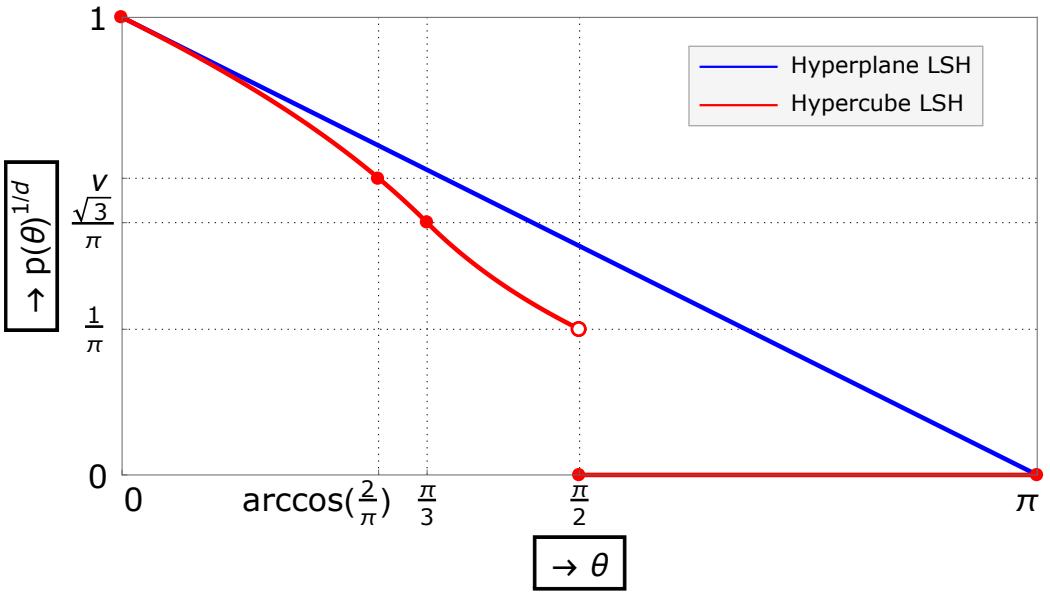


Figure 1 Asymptotics of collision probabilities for hypercube LSH, compared to hyperplane LSH. Here $\nu = \pi/(2\sqrt{\pi^2-4})$, and the dashed vertical lines correspond to boundary points of the piecewise parts of Theorem 1. The blue line indicates hyperplane LSH with d random hyperplanes.

▶ **Theorem 1** (Collision probabilities for hypercube LSH). *Let $\boldsymbol{X}, \boldsymbol{Y} \sim \mathcal{N}(0,1)^d$, let $\theta \in [0,\pi]$ denote the angle between $\boldsymbol{X}$ and $\boldsymbol{Y}$, and let $p(\theta)$ denote the probability that $\boldsymbol{X}$ and $\boldsymbol{Y}$ are mapped to the same hypercube hash region. For $\theta \in (0, \arccos\frac{2}{\pi})$ (respectively $\theta \in (\arccos\frac{2}{\pi}, \frac{\pi}{3})$), let $\beta_0 \in (1,\infty)$ (resp. $\beta_1 \in (1,\infty)$) be the unique solution to:*

$$\arccos\left(\frac{-1}{\beta_0}\right) = \frac{(\beta_0 - \cos\theta)\sqrt{\beta_0^2-1}}{\beta_0(\beta_0\cos\theta - 1)}, \qquad \arccos\left(\frac{1}{\beta_1}\right) = \frac{(\beta_1 + \cos\theta)\sqrt{\beta_1^2-1}}{\beta_1(\beta_1\cos\theta + 1)}. \tag{1}$$

Then, as d tends to infinity, $p(\theta)$ satisfies:

$$p(\theta) = \begin{cases} \left(\frac{(\beta_0-\cos\theta)^2}{\pi\beta_0(\beta_0\cos\theta-1)\sin\theta}\right)^{d+o(d)}, & \text{if } \theta \in [0, \arccos\frac{2}{\pi}]; \\ \left(\frac{(\beta_1+\cos\theta)^2}{\pi\beta_1(\beta_1\cos\theta+1)\sin\theta}\right)^{d+o(d)}, & \text{if } \theta \in [\arccos\frac{2}{\pi}, \frac{\pi}{3}]; \\ \left(\frac{1+\cos\theta}{\pi\sin\theta}\right)^{d+o(d)}, & \text{if } \theta \in [\frac{\pi}{3}, \frac{\pi}{2}); \\ 0, & \text{if } \theta \in [\frac{\pi}{2}, \pi]. \end{cases} \tag{2}$$

Denoting the query complexity of LSH methods by $n^{\rho+o(1)}$, the parameter ρ for hypercube LSH is up to $\log_2(\pi) \approx 1.65$ times smaller than for hyperplane LSH. For large d, hypercube LSH is dominated by cross-polytope LSH (unless $c \cdot r > \sqrt{2}$), but as the convergence to the limit is rather slow, in practice either method might be better, depending on the exact parameter setting. For the random setting, Figure 2 shows limiting values for ρ for hyperplane, hypercube and cross-polytope LSH. We again remark that these are asymptotics for $d \to \infty$, and may not accurately reflect the performance of these methods for moderate d. We further briefly discuss how the hashing for hypercube LSH can be made efficient.

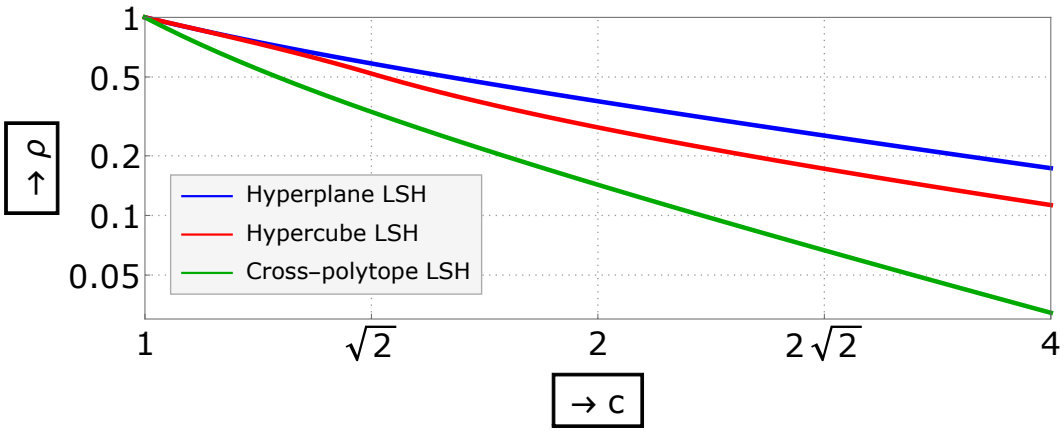


Figure 2 Asymptotics for the LSH exponent ρ when using hyperplane LSH, hypercube LSH, and cross-polytope LSH, for (c, r)-ANN with $c \cdot r = \sqrt{2}$. The curve for hyperplane LSH is exact for arbitrary d, while for the other two curves, order terms vanishing as $d \to \infty$ have been omitted.

Partial hypercube LSH. As the number of hash regions of a full-dimensional hypercube is often prohibitively large, we also consider *partial* hypercube LSH, where a d'-dimensional hypercube is used to partition a data set in dimension d. Building upon a result of Jiang [19], we characterize when hypercube and hyperplane LSH are asymptotically equivalent in terms of the relation between d' and d, and we empirically illustrate the convergence towards either hyperplane or hypercube LSH for larger d'. An important open problem remains to identify how large the ratio d'/d must be for the asymptotics of partial hypercube LSH to be equivalent to those of full-dimensional hypercube LSH.

Application to lattice sieving. Finally, we consider a specific use case of different LSH methods, in the context of lattice cryptanalysis. We show that the heuristic complexity of lattice sieving with hypercube LSH is expected to be slightly better than when using hyperplane LSH, and we discuss how experiments have previously indicated that in this application, hypercube LSH is superior to other dimensions up to dimensions $d \approx 80$.

2 Preliminaries

Notation. We denote probabilities with $\mathbb{P}(\cdot)$ and expectations with $\mathbb{E}(\cdot)$. Capital letters commonly denote random variables, and boldface letters denote vectors. We informally write $\mathbb{P}(X = x)$ for continuous X to denote the density of X at x. For probability distributions $\mathcal{D}$, we write $X \sim \mathcal{D}$ to denote that X is distributed according to $\mathcal{D}$. For sets S, with abuse of notation we further write $X \sim S$ to denote X is drawn uniformly at random from S. We write $\mathcal{N}(\mu, \sigma^2)$ for the normal distribution with mean μ and variance σ^2, and $\mathcal{H}(\mu, \sigma^2)$ for the distribution of $|X|$ when $X \sim \mathcal{N}(\mu, \sigma^2)$. For $\mu = 0$ the latter corresponds to the *half-normal* distribution. We write $\boldsymbol{X} \sim \mathcal{D}^d$ to denote a d-dimensional vector where each entry is independently distributed according to $\mathcal{D}$. In what follows, $\|\boldsymbol{x}\| = \sqrt{\sum_i x_i^2}$ denotes the Euclidean norm, and $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_i x_i y_i$ denotes the standard inner product. We denote the angle between two vectors by $\phi(\boldsymbol{x}, \boldsymbol{y}) = \arccos\langle \boldsymbol{x}/\|\boldsymbol{x}\|, \boldsymbol{y}/\|\boldsymbol{y}\| \rangle$.

▶ **Lemma 2** (Distribution of angles between random vectors [9, Lemma 2]). *Let $\boldsymbol{X}, \boldsymbol{Y} \sim \mathcal{N}(0,1)^d$ be two independent standard normal vectors. Then $\mathbb{P}(\phi(\boldsymbol{X}, \boldsymbol{Y}) = \theta) = (\sin\theta)^{d+o(d)}$.*

Locality-sensitive hashing. Locality-sensitive hash functions [18] are functions h mapping a d-dimensional vector $\boldsymbol{x}$ to a low-dimensional *sketch* $h(\boldsymbol{x})$, such that vectors which are nearby in $\mathbb{R}^d$ are more likely to be mapped to the same sketch than distant vectors. For the angular distance[1] $\phi(\boldsymbol{x}, \boldsymbol{y})$, we quantify a set of hash functions $\mathcal{H}$ as follows (see [18]):

▶ **Definition 3.** A hash family $\mathcal{H}$ is called $(\theta_1, \theta_2, p_1, p_2)$-sensitive if for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ we have:
- If $\phi(\boldsymbol{x}, \boldsymbol{y}) \leq \theta_1$ then $\mathbb{P}_{h\sim\mathcal{H}}(h(\boldsymbol{x}) = h(\boldsymbol{y})) \geq p_1$;
- If $\phi(\boldsymbol{x}, \boldsymbol{y}) \geq \theta_2$ then $\mathbb{P}_{h\sim\mathcal{H}}(h(\boldsymbol{x}) = h(\boldsymbol{y})) \leq p_2$.

The existence of locality-sensitive hash families implies the existence of fast algorithms for (approximate) near neighbors, as the following lemma describes[2]. For more details on the general principles of LSH, we refer the reader to e.g. [18, 4].

▶ **Lemma 4** (Locality-sensitive hashing [18]). *Suppose there exists a $(\theta_1, \theta_2, p_1, p_2)$-sensitive family $\mathcal{H}$. Let $\rho = \frac{\log(p_1)}{\log(p_2)}$. Then w.h.p. we can either find an element $\boldsymbol{p} \in L$ at angle at most θ_2 from $\boldsymbol{q}$, or conclude that no elements $\boldsymbol{p} \in L$ at angle at most θ_1 from $\boldsymbol{q}$ exist, in time $n^{\rho+o(1)}$ with space and preprocessing costs $n^{1+\rho+o(1)}$.*

Hyperplane LSH. For the angular distance, Charikar [12] introduced the hash family $\mathcal{H} = \{h_{\boldsymbol{a}} : \boldsymbol{a} \sim \mathcal{D}\}$ where $\mathcal{D}$ is any spherically symmetric distribution on $\mathbb{R}^d$, and $h_{\boldsymbol{a}}$ satisfies:

$$h_{\boldsymbol{a}}(\boldsymbol{x}) = \begin{cases} +1, & \text{if } \langle \boldsymbol{a}, \boldsymbol{x} \rangle \geq 0; \\ -1, & \text{if } \langle \boldsymbol{a}, \boldsymbol{x} \rangle < 0. \end{cases} \tag{3}$$

The vector $\boldsymbol{a}$ can be interpreted as the normal vector of a random hyperplane, and the hash value depends on which side of the hyperplane $\boldsymbol{x}$ lies on. For this hash function, the probability of a collision is directly proportional to the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$\mathbb{P}_{h\sim\mathcal{H}}\big(h(\boldsymbol{x}) = h(\boldsymbol{y})\big) = 1 - \frac{\phi(\boldsymbol{x}, \boldsymbol{y})}{\pi}. \tag{4}$$

For any two angles $\theta_1 < \theta_2$, the above family $\mathcal{H}$ is $(\theta_1, \theta_2, 1 - \frac{\theta_1}{\pi}, 1 - \frac{\theta_2}{\pi})$-sensitive.

Large deviations theory. Let $\{\boldsymbol{Z}_d\}_{d\in\mathbb{N}} \subset \mathbb{R}^k$ be a sequence of random vectors corresponding to an empirical mean, i.e. $\boldsymbol{Z}_d = \frac{1}{d}\sum_{i=1}^{d} \boldsymbol{U}_i$ with $\boldsymbol{U}_i$ i.i.d. We define the logarithmic moment generating function Λ of $\boldsymbol{Z}_d$ as:

$$\Lambda(\boldsymbol{\lambda}) = \ln \mathbb{E}_{\boldsymbol{U}_1}\left[\exp\langle \boldsymbol{\lambda}, \boldsymbol{U}_1 \rangle\right]. \tag{5}$$

Define $\mathcal{D}_\Lambda = \{\boldsymbol{\lambda} \in \mathbb{R}^k : \Lambda(\boldsymbol{\lambda}) < \infty\}$. The *Fenchel-Legendre* transform of Λ is defined as:

$$\Lambda^*(\boldsymbol{z}) = \sup_{\boldsymbol{\lambda}\in\mathbb{R}^k} \{\langle \boldsymbol{\lambda}, \boldsymbol{z} \rangle - \Lambda(\boldsymbol{\lambda})\}. \tag{6}$$

The following result describes that under certain conditions on $\{\boldsymbol{Z}'_d\}$, the asymptotics of the probability measure on a set F are related to the function Λ^*.

▶ **Lemma 5** (Gärtner-Ellis theorem [14, Theorem 2.3.6 and Corollary 6.1.6]). *Let $\mathbf{0}$ be contained in the interior of $\mathcal{D}_\Lambda$, and let $\boldsymbol{Z}_d$ be an empirical mean. Then for arbitrary sets F,*

$$\lim_{d\to\infty} \tfrac{1}{d} \ln \mathbb{P}(\boldsymbol{z} \in F) = -\inf_{\boldsymbol{z}\in F} \Lambda^*(\boldsymbol{z}). \tag{7}$$

The latter statement can be read as $\mathbb{P}(\boldsymbol{z} \in F) = \exp(-d \inf_{\boldsymbol{z}\in F} \Lambda^*(\boldsymbol{z}) + o(d))$, and thus tells us exactly how $\mathbb{P}(\boldsymbol{z} \in F)$ scales as d tends to infinity, up to order terms.

[1] Formally speaking, the angular distance is only a similarity measure, and not a metric.
[2] Various conditions and order terms (which are commonly $n^{o(1)}$) are omitted here for brevity.

3 Hypercube LSH

In this section, we will analyze full-dimensional hypercube hashing, with hash family $\mathcal{H} = \{h_A : A \in SO(d)\}$ where $SO(d) \subset \mathbb{R}^{d\times d}$ denotes the rotation group, and h_A satisfies:

$$h_A(\boldsymbol{x}) = (h_1(A\boldsymbol{x}), \dots, h_d(A\boldsymbol{x})), \qquad h_i(\boldsymbol{x}) = \begin{cases} +1, & \text{if } x_i \geq 0; \\ -1, & \text{if } x_i < 0. \end{cases} \tag{8}$$

In other words, a hypercube hash function first applies a uniformly random rotation, and then maps the resulting vector to the orthant it lies in. This equivalently corresponds to a concatenation of d hyperplane hash functions, where all hyperplanes are orthogonal. Collision probabilities for prescribed angles θ between $\boldsymbol{x}$ and $\boldsymbol{y}$ are denoted by:

$$p(\theta) = \mathbb{P}(h_A(\boldsymbol{x}) = h_A(\boldsymbol{y}) \mid \phi(\boldsymbol{x}, \boldsymbol{y}) = \theta). \tag{9}$$

Above, the randomness is over $h_A \sim \mathcal{H}$, with $\boldsymbol{x}$ and $\boldsymbol{y}$ arbitrary vectors at angle θ (e.g. $\boldsymbol{x} = \boldsymbol{e}_1$ and $\boldsymbol{y} = \boldsymbol{e}_1 \cos\theta + \boldsymbol{e}_2 \sin\theta$). Alternatively, the random rotation A inside h_A may be omitted, and the probability can be computed over $\boldsymbol{X}, \boldsymbol{Y}$ drawn uniformly at random from a spherically symmetric distribution, *conditioned* on their common angle being θ.

3.1 Outline of the proof of Theorem 1

Although Theorem 1 is a key result, due to space restrictions we have decided to defer the full proof (approximately 5.5 pages) to the appendix. The approach of the proof can be summarized by the following four steps:

- Rewrite the collision probabilities in terms of (normalized) half-normal vectors $\boldsymbol{X}, \boldsymbol{Y}$;
- Introduce dummy variables x, y for the norms of these half-normal vectors, so that the probability can be rewritten in terms of unnormalized half-normal vectors;
- Apply the Gärtner-Ellis theorem (Lemma 5) to the three-dimensional vector $\boldsymbol{Z} = \frac{1}{d}(\sum_i X_i Y_i, \sum_i X_i^2, \sum_i Y_i^2)$ to compute the resulting probabilities for arbitrary x, y;
- Maximize the resulting expressions over $x, y > 0$ to get the final result.

The majority of the technical part of the proof lies in computing $\Lambda^*(\boldsymbol{z})$, which involves a somewhat tedious optimization of a multivariate function through a case-by-case analysis.

A note on Gaussian approximations. From the (above outline of the) proof, and the observation that the final optimization over x, y yields $x = y = 1$ as the optimum, one might wonder whether a simpler analysis might be possible by assuming (half-)normal vectors are already normalized. Such a computation however would only lead to an approximate solution, which is perhaps easiest to see by computing collision probabilities for $\theta = 0$. In the exact computation, where vectors are normalized, $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle = 1$ implies $\boldsymbol{X} = \boldsymbol{Y}$. If however we do not take into account the norms of $\boldsymbol{X}$ and $\boldsymbol{Y}$, and do not condition on the norms being equal to 1, then $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle = 1$ could also mean that $\boldsymbol{X}, \boldsymbol{Y}$ are slightly longer than 1 and have a small, non-zero angle. In fact, such a computation would indeed yield $p(\theta)^{1/d} \not\to 0$ as $\theta \to 0$.

3.2 Consequences of Theorem 1

From Theorem 1, we can draw several conclusions. Substituting values for θ, we can find asymptotics for $p(\theta)$, such as $p(\frac{\pi}{3})^{1/d} = \frac{\sqrt{3}}{\pi} + o(1)$ and $p(\frac{\pi}{2})^{1/d} = \frac{1}{\pi} + o(1)$. We observe that the limiting function of Theorem 1 (without the order terms) is continuous everywhere except at $\theta = \frac{\pi}{2}$. To understand the boundary $\theta = \arccos\frac{2}{\pi}$ of the piece-wise limit function, note that two (normalized) half-normal vectors $\boldsymbol{X}, \boldsymbol{Y}$ have expected inner product $\mathbb{E}\langle \boldsymbol{X}, \boldsymbol{Y} \rangle = \frac{2}{\pi}$.

LSH exponents ρ for random settings. Using Theorem 1, we can explicitly compute LSH exponents ρ for given angles θ_1 and θ_2 for large d. As an example, consider the random setting[3] with $c = \sqrt{2}$, corresponding to $\theta_2 = \frac{\pi}{2}$ and $\theta_1 = \frac{\pi}{3}$. Substituting the collision probabilities from Theorem 1, we get $\rho \to 1 - \frac{1}{2}\log_\pi(3) \approx 0.520$ as $d \to \infty$. To compare, if we had used random hyperplanes, we would have gotten a limiting value $\rho \to \log_2(\frac{3}{2}) \approx 0.585$. For the random case, Figure 2 compares limiting values ρ using random and orthogonal hyperplanes, and using the asymptotically superior cross-polytope LSH.

Scaling at $\theta \to 0$ and asymptotics of ρ for large c. For θ close to 0, by Theorem 1 we are in the regime defined by β_0. For $\cos\theta = 1 - \varepsilon$ with $\varepsilon > 0$ small, observe that $\beta_0 \approx 1$ satisfies $\beta_0 > 1/\cos\theta$. Computing a Taylor expansion around $\varepsilon = 0$, we eventually find $\beta_0 = 1 + \varepsilon + \frac{2\sqrt{2}}{\pi}\varepsilon^{3/2} + O(\varepsilon^2)$. Substituting this value β_0 into $p(\theta)$ with $\cos\theta = 1 - \varepsilon$, we find:

$$p(\theta) = \left(1 - \frac{\sqrt{2}}{\pi}\sqrt{\varepsilon} + O(\varepsilon)\right)^{d+o(d)}. \tag{10}$$

To compare this with hyperplane LSH, recall that the collision probability for d random hyperplanes is equal to $(1 - \frac{\theta}{\pi})^d$. Since $\cos\theta = 1 - \varepsilon$ translates to $\theta = \sqrt{2\varepsilon}(1 + O(\varepsilon))$, the collision probabilities for hyperplane hashing in this regime are also $(1 - \frac{\sqrt{2}}{\pi}\sqrt{\varepsilon} + O(\varepsilon))^d$. In other words, for angles $\theta \to 0$, the collision probabilities for hyperplane hashing and hypercube hashing are similar. This can also be observed in Figure 1. Based on this result, we further deduce that in random settings with large c, for hypercube LSH we have:

$$\rho \to \frac{\ln\left(1 - \frac{\sqrt{2}}{\pi c} + O\left(\frac{1}{c^2}\right)\right)}{\ln(1/\pi)} = \frac{\sqrt{2}}{\pi c \ln \pi} + O\left(\frac{1}{c^2}\right) \approx \frac{0.393}{c} + O\left(\frac{1}{c^2}\right). \tag{11}$$

For hyperplane LSH, the numerator is the same, while the denominator is $\ln(\frac{1}{2})$ instead of $\ln(\frac{1}{\pi})$, leading to values ρ which are a factor $\log_2 \pi + o(1) \approx 1.652 + o(1)$ larger. Both methods are inferior to cross-polytope LSH for large d, as there $\rho = O(1/c^2)$ for large c [5].

3.3 Convergence to the limit

To get an idea how hypercube LSH compares to other methods when d is not too large, we start by giving explicit collision probabilities for the first non-trivial case, namely $d = 2$.

▶ **Proposition 6** (Square LSH). *For $d = 2$, $p(\theta) = 1 - \frac{2\theta}{\pi}$ for $\theta \leq \frac{\pi}{2}$ and $p(\theta) = 0$ otherwise.*

Proof. In two dimensions, two randomly rotated vectors $\boldsymbol{X}, \boldsymbol{Y}$ at angle θ can be modeled as $\boldsymbol{X} = (\cos\psi, \sin\psi)$ and $\boldsymbol{Y} = (\cos(\psi+\theta), \sin(\psi+\theta))$ for $\psi \sim [0, 2\pi)$. The conditions $\boldsymbol{X}, \boldsymbol{Y} > 0$ are then equivalent to $\psi \in (0, \frac{\pi}{2}) \cap (-\theta, \frac{\pi}{2} - \theta)$, which for $\theta < \frac{\pi}{2}$ occurs with probability $\frac{\pi/2-\theta}{2\pi}$ over the randomness of ψ. As a collision can occur in any of the four quadrants, we finally multiply this probability by 4 to obtain the stated result. ◀

Figure 3 depicts $p(\theta)^{1/2}$ in green, along with hyperplane LSH (blue) and the asymptotics for hypercube LSH (red). For larger d, computing $p(\theta)$ exactly becomes more complicated,

[3] Here we assume that $c \cdot r \to (\sqrt{2})^-$, i.e. $c \cdot r$ approaches $\sqrt{2}$ from below. Alternatively, one might interpret this as that if distant points lie at distance $\sqrt{2} \pm o(1)$, then we might expect approximately half of them to lie at distance less than $\sqrt{2}$, with query complexity $O(n/2)^{\rho+o(1)} = n^{\rho+o(1)}$. If however $c \cdot r \geq \sqrt{2}$ then clearly $\rho = 0$, regardless of d and c.

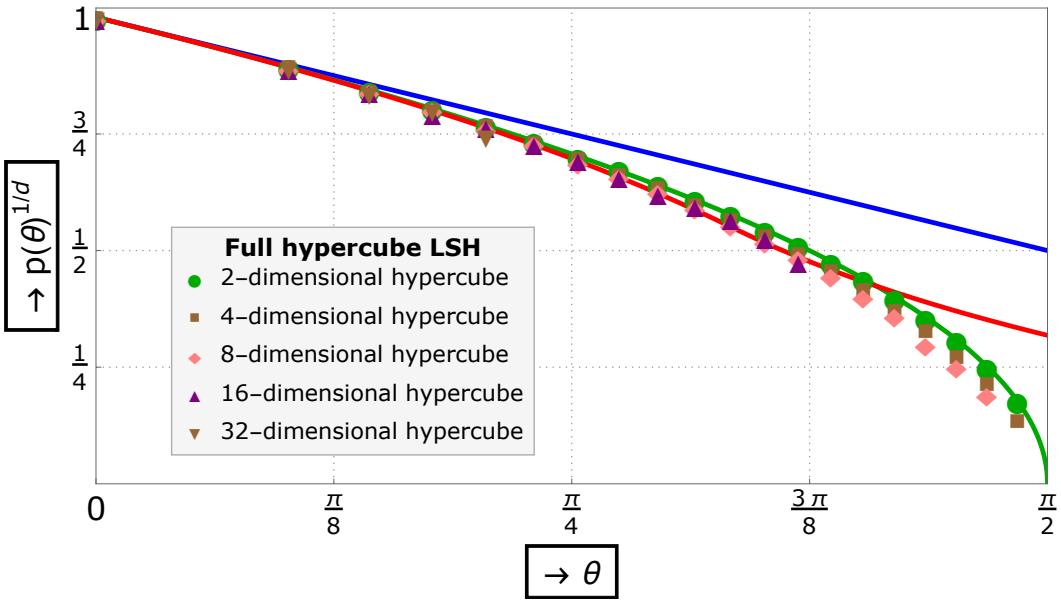


Figure 3 Empirical collision probabilities for hypercube LSH for small d. The green curve denotes the exact collision probabilities for $d = 2$ from Proposition 6.

and so instead we performed experiments to empirically obtain estimates for $p(\theta)$ as d increases. These estimates are also shown in Figure 3, and are based on 10^5 trials for each θ and d. Observe that as $\theta \to \frac{\pi}{2}$ and/or d grows larger, $p(\theta)$ decreases and the empirical estimates become less reliable. Points are omitted for cases where no successes occurred.

Based on these estimates and our intuition, we conjecture that (1) for $\theta \approx 0$, the scaling of $p(\theta)^{1/d}$ is similar for all d, and similar to the asymptotic behavior of Theorem 1; (2) the normalized collision probabilities for $\theta \approx \frac{\pi}{2}$ approach their limiting value from below; and (3) $p(\theta)$ is likely to be continuous for arbitrary d, implying that for $\theta \to \frac{\pi}{2}$, the collision probabilities tend to 0 for each d. These together suggest that values for ρ are actually *smaller* when d is small than when d is large, and the asymptotic estimate from Figure 2 might be pessimistic in practice. For the random setting, this would suggest that $\rho \approx 0$ regardless of c, as $p(\theta) \to 0$ as $\theta \to \frac{\pi}{2}$ for arbitrary d.

Comparison with hyperplane/cross-polytope LSH. Finally, [36, Figures 1 and 2] previously illustrated that among several LSH methods, the smallest values ρ (for their parameter sets) are obtained with hypercube LSH with $d = 16$, achieving smaller values ρ than e.g. cross-polytope LSH with $d = 256$. An explanation for this can be found in:

- The (conjectured) convergence of ρ to its limit from *below*, for hypercube LSH;
- The slow convergence of ρ to its limit (from above) for cross-polytope LSH[4].

This suggests that the actual values ρ for moderate dimensions d may well be smaller for hypercube LSH (and hyperplane LSH) than for cross-polytope LSH. Based on the limiting cases $d = 2$ and $d \to \infty$, we further conjecture that compared to hyperplane LSH, hypercube LSH achieves smaller values ρ for arbitrary d.

[4] [5, Theorem 1] shows that the leading term in the asymptotics for ρ scales as $\Theta(\ln d)$, with a first order term scaling as $O(\ln \ln d)$, i.e. a relative order term of the order $O(\ln \ln d / \ln d)$.

3.4 Fast hashing in practice

To further assess the practicality of hypercube LSH, recall that hashing is done as follows:

- Apply a uniformly random rotation A to $\boldsymbol{x}$;
- Look at the signs of $(A\boldsymbol{x})_i$.

Theoretically, a uniformly random rotation will be rather expensive to compute, with A being a real, dense matrix. As previously discussed in e.g. [3], it may suffice to only consider a sparse subset of all rotation matrices with a large enough amount of randomness, and as described in [5, 21] pseudo-random rotations may also be help speed up the computations in practice. As described in [21], this can even be made provable, to obtain a reduced $O(d \log d)$ computational complexity for applying a random rotation.

Finally, to compare this with cross-polytope LSH, note that cross-polytope LSH in dimension d partitions the space in $2d$ regions, as opposed to 2^d for hypercube hashing. To obtain a similar fine-grained partition of the space with cross-polytopes, one would have to concatenate $\Theta(d/\log d)$ random cross-polytope hashes, which corresponds to computing $\Theta(d/\log d)$ (pseudo-)random rotations, compared to only one rotation for hypercube LSH. We therefore expect hashing to be up to a factor $\Theta(d/\log d)$ less costly.

4 Partial hypercube LSH

Since a high-dimensional hypercube partitions the space in a large number of regions, for various applications one may only want to use hypercubes in a lower dimension $d' < d$. In those cases, one would first apply a random rotation to the data set, and then compute the hash based on the signs of the first d' coordinates of the rotated data set. This corresponds to the hash family $\mathcal{H} = \{h_{A,d'} : A \in SO(d)\}$, with $h_{A,d'}$ satisfying:

$$h_{A,d'}(\boldsymbol{x}) = (h_1(A\boldsymbol{x}), \ldots, h_{d'}(A\boldsymbol{x})), \qquad h_i(\boldsymbol{x}) = \begin{cases} +1, & \text{if } x_i \geq 0; \\ -1, & \text{if } x_i < 0. \end{cases} \tag{12}$$

When "projecting" down onto the first d' coordinates, observe that distances and angles are distorted: the angle between the vectors formed by the first d' coordinates of $\boldsymbol{x}$ and $\boldsymbol{y}$ may not be the same as $\phi(\boldsymbol{x}, \boldsymbol{y})$. The amount of distortion depends on the relation between d' and d. Below, we will investigate how the collision probabilities $p_{d',d}(\theta)$ for partial hypercube LSH scale with d' and d, where $p_{d',d}(\theta) = \mathbb{P}(h(\boldsymbol{x}) = h(\boldsymbol{y}) \mid \phi(\boldsymbol{x}, \boldsymbol{y}) = \theta)$.

4.1 Convergence to hyperplane LSH

First, observe that for $d' = 1$, partial hypercube LSH is *equal* to hyperplane LSH, i.e. $p_{1,d}(\theta) = 1 - \frac{\theta}{\pi}$. For $1 < d' \ll d$, we first observe that both (partial) hypercube LSH and hyperplane LSH can be modeled by a projection onto d' dimensions:

- Hyperplane LSH: $\boldsymbol{x} \mapsto A\boldsymbol{x}$ with $A \sim \mathcal{N}(0,1)^{d' \times d}$;
- Hypercube LSH: $\boldsymbol{x} \mapsto (A^*)\boldsymbol{x}$ with $A \sim \mathcal{N}(0,1)^{d' \times d}$.

Here A^* denotes the matrix obtained from A after applying Gram-Schmidt orthogonalization to the rows of A. In both cases, hashing is done after the projection by looking at the signs of the projected vector. Therefore, the only difference lies in the projection, and one could ask: for which d', as a function of d, are these projections equivalent? When is a set of random hyperplanes already (almost) orthogonal?

This question was answered in [19]: if $d' = o(d/\log d)$, then $\max_{i,j} |A_{i,j} - A^*_{i,j}| \to 0$ in probability as $d \to \infty$ (implying $A^* = (1 + o(1))A$), while for $d' = \Omega(d/\log d)$ this

maximum does not converge to 0 in probability. In other words, for large d a set of d' random hyperplanes in d dimensions is (approximately) orthogonal iff $d' = o(d/\log d)$.

▶ **Proposition 7** (Convergence to hyperplane LSH). *Let $p_{d',d}(\theta)$ denote the collision probabilities for partial hypercube LSH, and let $d' = o(d/\log d)$. Then $p_{d',d}(\theta)^{1/d'} \to 1 - \frac{\theta}{\pi}$.*

As $d' = \Omega(d/\log d)$ random vectors in d dimensions are asymptotically *not* orthogonal, in that case one might expect either convergence to full-dimensional hypercube LSH, or to something in between hyperplane and hypercube LSH.

4.2 Convergence to hypercube LSH

To characterize when partial hypercube LSH is equivalent to full hypercube LSH, we first observe that if d' is large compared to $\ln n$, then convergence to the hypercube LSH asymptotics follows from the Johnson-Lindenstrauss lemma.

▶ **Proposition 8** (Sparse data sets). *Let $d' = \omega(\ln n)$. Then the same asymptotics for the collision probabilities as those of full-dimensional hypercube LSH apply.*

Proof. Let $\theta \in (0, \frac{\pi}{2})$. By the Johnson-Lindenstrauss lemma [20], we can construct a projection $\boldsymbol{x} \mapsto A\boldsymbol{x}$ from d onto d' dimensions, preserving all pairwise distances up to a factor $1 \pm \varepsilon$ for $\varepsilon = \Theta((\ln n)/d') = o(1)$. For fixed $\theta \in (0, \frac{\pi}{2})$, this implies the angle ϕ between $A\boldsymbol{x}$ and $A\boldsymbol{y}$ will be in the interval $\theta \pm o(1)$, and so the collision probability lies in the interval $p(\theta \pm o(1))$. For large d, this means that the asymptotics of $p(\theta)$ are the same. ◀

To analyze collision probabilities for partial hypercube LSH when neither of the previous two propositions applies, note that through a series of transformations similar to those for full-dimensional hypercube LSH, it is possible to eventually end up with the following probability to compute, where $d_1 = d'$ and $d_2 = d - d'$:

$$\max_{x,y,u,v,\phi} \mathbb{P}\left(\frac{1}{d_1}\sum_{i=1}^{d_1} X_i Y_i = xy\cos\phi, \quad \frac{1}{d_1}\sum_{i=1}^{d_1} X_i^2 = x^2, \quad \frac{1}{d_1}\sum_{i=1}^{d_1} Y_i^2 = y^2, \right. \tag{13}$$

$$\left. \frac{1}{d_2}\sum_{i=1}^{d_2} U_i V_i = uvf(\phi,\theta), \quad \frac{1}{d_2}\sum_{i=1}^{d_2} U_i^2 = u^2, \quad \frac{1}{d_2}\sum_{i=1}^{d_2} V_i^2 = v^2 \right). \tag{14}$$

Here f is some function of ϕ and θ. The approach is comparable to how we ended up with a similar probability to compute in the proof of Theorem 1, except that we split the summation indices $I = [d]$ into two sets $I_1 = \{1, \ldots, d'\}$ of size d_1 and $I_2 = \{d'+1, \ldots, d\}$ of size d_2. We then substitute $U_i = X_{d'+i}$ and $V_i = Y_{d'+i}$, and add dummy variables x, y, u, v for the norms of the four partial vectors, and a dummy angle ϕ for the angle between the d_1-dimensional vectors, given the angle θ between the d-dimensional vectors.

Although the vector $\boldsymbol{Z}$ formed by the six random variables in (14) is not an empirical mean over a fixed number d of random vectors (the first three are over d_1 terms, the last three over d_2 terms), one may expect a similar large deviations result such as Lemma 5 to apply here. In that case, the function $\Lambda^*(\boldsymbol{z}) = \Lambda^*(z_1, \ldots, z_6)$ would be a function of six variables, which we would like to evaluate at $(xy\cos\phi, x^2, y^2, uvf(\phi,\theta), u^2, v^2)$. The function Λ^* itself involves an optimization (finding a supremum) over another six variables $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_6)$, so to compute collision probabilities for given d, d', θ exactly, using large deviations theory, one would have to compute an expression of the following form:

$$\min_{x,y,u,v,\phi} \left\{ \sup_{\lambda_1,\lambda_2,\lambda_3,\lambda_4,\lambda_5,\lambda_6} F_{d,d',\theta}(x,y,u,v,\phi,\lambda_1,\lambda_2,\lambda_3,\lambda_4,\lambda_5,\lambda_6) \right\}. \tag{15}$$

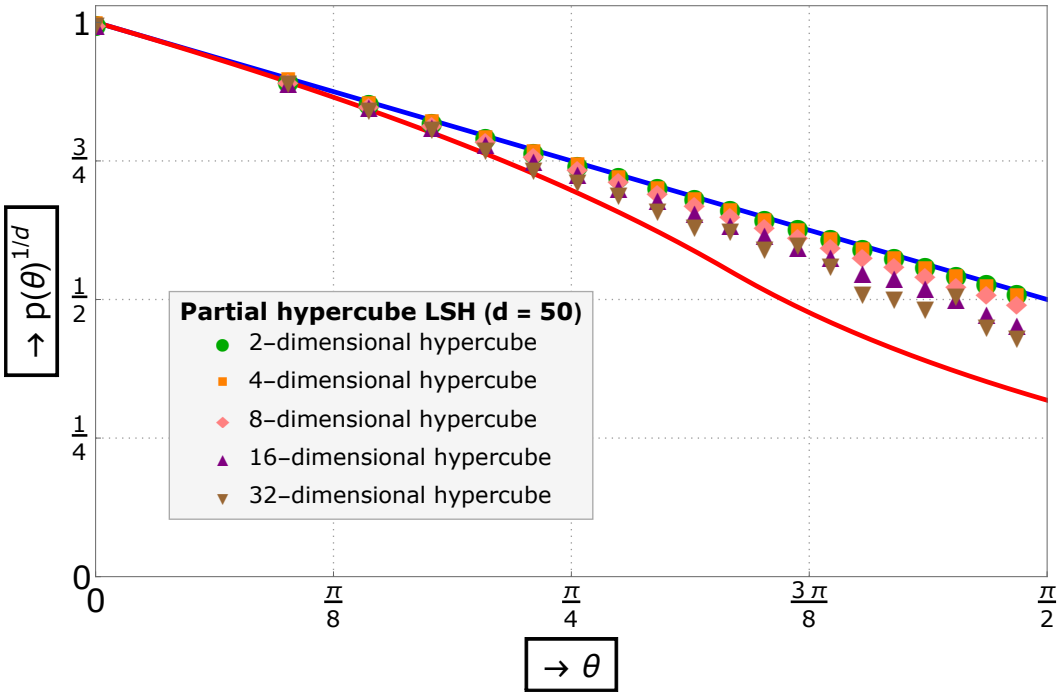


Figure 4 Experimental values of $p_{d',50}(\theta)^{1/d'}$, for different values d', compared with the asymptotics for hypercube LSH (red) and hyperplane LSH (blue).

As this is a very complex task, and the optimization will depend heavily on the parameters d, d', θ defined by the problem setting, we leave this optimization as an open problem. We only mention that intuitively, from the limiting cases of small and large d' we expect that depending on how d' scales with d (or n), we obtain a curve somewhere in between the two curves depicted in Figure 1.

4.3 Empirical collision probabilities

To get an idea of how $p_{d',d}(\theta)$ scales with d' in practice, we empirically computed several values for fixed $d = 50$. For fixed θ we then applied a least-squares fit of the form $e^{c_1 d + c_2}$ to the resulting data, and plotted e^{c_1} in Figure 4. These data points are again based on at least 10^5 experiments for each d' and θ. We expect that as d' increases, the collision probabilities slowly move from hyperplane hashing towards hypercube hashing, this can also be seen in the graph – for $d' = 2$, the least-squares fit is almost equal to the curve for hyperplane LSH, while as d' increases the curve slowly moves down towards the asymptotics for full hypercube LSH. Again, we stress that as d' becomes larger, the empirical estimates become less reliable, and so we did not consider even larger values for d'.

Compared to full hypercube LSH and Figure 3, we observe that we now approach the limit from above (although the fitted collision probabilities never seem to be smaller than those of hyperplane LSH), and therefore the values ρ for partial hypercube LSH are likely to lie in between those of hyperplane and (the asymptotics of) hypercube LSH.

5 Application: Lattice sieving for the shortest vector problem

We finally consider an explicit application for hypercube LSH, namely lattice sieving algorithms for the shortest vector problem. Given a basis $\boldsymbol{B} = \{\boldsymbol{b}_1, \dots, \boldsymbol{b}_d\} \subset \mathbb{R}^d$ of a lattice $\mathcal{L}(\boldsymbol{B}) = \{\sum_i \lambda_i \boldsymbol{b}_i : \lambda_i \in \mathbb{Z}\}$, the shortest vector problem (SVP) asks to find a shortest non-zero vector in this lattice. Various different methods for solving SVP in high dimensions are known, and currently the algorithm with the best heuristic time complexity in high dimensions is based on lattice sieving, combined with nearest neighbor searching [9].

In short, lattice sieving works by generating a long list L of pairwise reduced lattice vectors, where $\boldsymbol{x}, \boldsymbol{y}$ are reduced iff $\|\boldsymbol{x} - \boldsymbol{y}\| \geq \min\{\|\boldsymbol{x}\|, \|\boldsymbol{y}\|\}$. The previous condition is equivalent to $\phi(\boldsymbol{x}, \boldsymbol{y}) \leq \frac{\pi}{3}$, and so the length of L can be bounded by the kissing constant in dimension d, which is conjectured to scale as $(4/3)^{d/2+o(d)}$. Therefore, if we have a list of size $n = (4/3)^{d/2+o(d)}$, any newly sampled lattice vector can be reduced against the list many times to obtain a very short lattice vector. The time complexity of this method is dominated by doing $\mathrm{poly}(d) \cdot n$ reductions (searches for nearby vectors) with a list of size n. A linear search trivially leads to a heuristic complexity of $n^{2+o(1)} = (4/3)^{d+o(d)}$ (with space $n^{1+o(1)}$), while nearest neighbor techniques can reduce the time complexity to $n^{1+\rho+o(1)}$ for $\rho < 1$ (increasing the space to $n^{1+\rho+o(1)}$). For more details, see e.g. [31, 23, 9].

Based on the collision probabilities for hypercube LSH, and assuming the asymptotics for partial hypercube LSH (with $d' = O(d)$) are similar to those of full-dimensional hypercube LSH, we obtain the following result. An outline of the proof is given in the appendix.

▶ **Proposition 9** (Complexity of lattice sieving with hypercube LSH)**.** *Suppose the asymptotics for full hypercube LSH also hold for partial hypercube LSH with $d' \approx 0.1335d$. Then lattice sieving with hypercube LSH heuristically solves SVP in time and space $2^{0.3222d+o(d)}$.*

As expected, the conjectured asymptotic performance of (sieving with) hypercube LSH lies in between those of hyperplane LSH and cross-polytope LSH.

- Linear search [31]: $2^{0.4150d+o(d)}$.
- Hyperplane LSH [23]: $2^{0.3366d+o(d)}$.
- **Hypercube LSH**: $2^{0.3222d+o(d)}$.
- Spherical cap LSH [24]: $2^{0.2972d+o(d)}$.
- Cross-polytope LSH [10]: $2^{0.2972d+o(d)}$.
- Spherical LSF [9]: $2^{0.2925d+o(d)}$.

In practice however, the picture is almost entirely reversed [1]. The lattice sieving method used to solve SVP in the highest dimension to date ($d = 116$) used a very optimized linear search [22]. The furthest that any nearest neighbor-based sieve has been able to go to date is $d = 107$, using hypercube LSH [27, 26][5]. Experiments further indicated that spherical LSF only becomes competitive with hypercube LSH as $d \gtrsim 80$ [9, 28], while sieving with cross-polytope LSH turned out to be rather slow compared to other methods [10, 25]. Although it remains unclear which nearest neighbor method is the "most practical" in the application of lattice sieving, hypercube LSH is one of the main contenders.

Acknowledgments. The author is indebted to Ofer Zeitouni for his suggestion to use results from large deviations theory, and for his many helpful comments regarding this application. The author further thanks Brendan McKay and Carlo Beenakker for their comments.

[5] Although phrased as hyperplane LSH, the implementations from [23, 27, 26] are using hypercube LSH.

References

1. SVP challenge, 2015. URL: http://latticechallenge.org/svp-challenge/.
2. Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Formulas*. Dover Publications, 1972. URL: http://people.math.sfu.ca/~cbm/aands/toc.htm.
3. Dimitris Achlioptas. Database-friendly random projections. In *PODS*, pages 274–281, 2001. doi:10.1145/375551.375608.
4. Alexandr Andoni. *Nearest Neighbor Search: the Old, the New, and the Impossible*. PhD thesis, Massachusetts Institute of Technology, 2009. URL: http://hdl.handle.net/1721.1/55090.
5. Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In *NIPS*, pages 1225–1233, 2015. URL: https://papers.nips.cc/paper/5893-practical-and-optimal-lsh-for-angular-distance.
6. Alexandr Andoni, Piotr Indyk, Huy Lê Nguyên, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *SODA*, pages 1018–1028, 2014. doi:10.1137/1.9781611973402.76.
7. Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *SODA*, pages 47–66, 2017. doi:10.1137/1.9781611974782.4.
8. Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *STOC*, pages 793–801, 2015. doi:10.1145/2746539.2746553.
9. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA*, pages 10–24, 2016. doi:10.1137/1.9781611974331.ch2.
10. Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. In *AFRICACRYPT*, pages 3–23, 2016. doi:10.1007/978-3-319-31517-1_1.
11. Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.
12. Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002. doi:10.1145/509907.509965.
13. Tobias Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In *SODA*, pages 31–46, 2017. doi:10.1137/1.9781611974782.3.
14. Amir Dembo and Ofer Zeitouni. *Large deviations techniques and applications (2nd edition)*. Springer, 2010. doi:10.1007/978-3-642-03311-7.
15. Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Transactions on Information Theory*, 56(8):4166–4179, Aug 2010. doi:10.1109/TIT.2010.2050814.
16. Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley, 2000.
17. Kave Eshghi and Shyamsundar Rajaram. Locality sensitive hash functions based on concomitant rank order statistics. In *KDD*, pages 221–229, 2008. doi:10.1145/1401890.1401921.
18. Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998. doi:10.1145/276698.276876.
19. Tiefeng Jiang. How many entries of a typical orthogonal matrix can be approximated by independent normals? *The Annals of Probability*, 34(4):1497–1529, 2006. doi:10.1214/009117906000000205.
20. William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26(1):189–206, 1984. doi:10.1090/conm/026/737400.

21 Christopher Kennedy and Rachel Ward. Fast cross-polytope locality-sensitive hashing. In *ITCS*, 2017. URL: https://arxiv.org/abs/1602.06922.

22 Thorsten Kleinjung. Private communication, 2014.

23 Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *CRYPTO*, pages 3–22, 2015. doi:10.1007/978-3-662-47989-6_1.

24 Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In *LATINCRYPT*, pages 101–118, 2015. doi:10.1007/978-3-319-22174-8_6.

25 Artur Mariano. Private communication., 2016.

26 Artur Mariano and Christian Bischof. Enhancing the scalability and memory usage of HashSieve on multi-core CPUs. In *PDP*, pages 545–552, 2016. doi:10.1109/PDP.2016.31.

27 Artur Mariano, Thijs Laarhoven, and Christian Bischof. Parallel (probable) lock-free HashSieve: a practical sieving algorithm for the SVP. In *ICPP*, pages 590–599, 2015. URL: https://eprint.iacr.org/2015/041.

28 Artur Mariano, Thijs Laarhoven, and Christian Bischof. A parallel variant of LDSieve for the SVP on lattices. *PDP*, 2017.

29 Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015. doi:10.1007/978-3-662-46800-5_9.

30 Rajeev Motwani, Assaf Naor, and Rina Panigrahy. Lower bounds on locality sensitive hashing. *SIAM Journal of Discrete Mathematics*, 21(4):930–935, 2007. doi:10.1137/050646858.

31 Phong Q. Nguyên and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008. doi:10.1515/JMC.2008.009.

32 Ryan O'Donnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality sensitive hashing (except when q is tiny). In *ICS*, pages 276–283, 2011. URL: http://conference.itcs.tsinghua.edu.cn/ICS2011/content/papers/2.html.

33 Ludwig Schmidt, Matthew Sharifi, and Ignacio Lopez-Moreno. Large-scale speaker identification. In *ICASSP*, pages 1650–1654, 2014. doi:10.1109/ICASSP.2014.6853878.

34 Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2005. URL: http://ttic.uchicago.edu/~gregory/annbook/book.html.

35 Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *VLDB*, 6(14):1930–1941, 2013. doi:10.14778/2556549.2556574.

36 Kengo Terasawa and Yuzuru Tanaka. Spherical LSH for approximate nearest neighbor search on unit hypersphere. In *WADS*, pages 27–38, 2007. doi:10.1007/978-3-540-73951-7_4.

37 Kengo Terasawa and Yuzuru Tanaka. Approximate nearest neighbor search for a dataset of normalized vectors. In *IEICE Transactions on Information and Systems*, volume 92, pages 1609–1619, 2009. URL: http://search.ieice.org/bin/summary.php?id=e92-d_9_1609.

A Proof of Theorem 1

Theorem 1 will be proved through a series of lemmas, each making partial progress towards a final solution. Reading only the claims made in the lemmas may give the reader an idea how the proof is built up. Before starting the proof, we begin with a useful lemma regarding integrals of (exponentials of) quadratic forms.

▶ **Lemma 10** (Integrating an exponential of a quadratic form in the positive quadrant)**.** *Let $a, b, c \in \mathbb{R}$ with $a, c < 0$ and $D = b^2 - 4ac < 0$. Then:*

$$\int_0^\infty \int_0^\infty \exp(ax^2 + bxy + cy^2)\, dx\, dy = \frac{\pi + 2\arctan\left(\frac{b}{\sqrt{-D}}\right)}{2\sqrt{-D}}. \tag{16}$$

Proof. The proof below is based on substituting $y = xs$ (and $dy = x\, ds$) before computing the integral over x. An integral over $1/(a + bs + cs^2)$ then remains, which leads to the arctangent solution in case $b^2 < 4ac$.

$$I = \int_{y=0}^\infty \int_0^\infty \exp(ax^2 + bxy + cy^2)\, dx\, dy \tag{17}$$

$$= \int_{s=0}^\infty \left(\int_0^\infty x \exp\left((a + bs + cs^2)x^2\right)\, dx \right) ds \tag{18}$$

$$= \int_0^\infty \left[\frac{\exp\left((a + bs + cs^2)x^2\right)}{2(a + bs + cs^2)} \right]_{x=0}^\infty ds \tag{19}$$

$$= \int_0^\infty \left[0 - \frac{1}{2(a + bs + cs^2)} \right] ds \tag{20}$$

$$= \frac{-1}{2} \int_0^\infty \frac{1}{a + bs + cs^2}\, ds. \tag{21}$$

The last equality used the assumptions $a, c < 0$ and $b^2 < 4ac$ so that $a + bs + cs^2 < 0$ for all $s > 0$. We then solve the last remaining integral (see e.g. [2, Equation (3.3.16)]) to obtain:

$$I = \frac{-1}{2} \left[\frac{2}{\sqrt{4ac - b^2}} \arctan\left(\frac{b + 2cs}{\sqrt{4ac - b^2}} \right) \right]_{s=0}^\infty \tag{22}$$

$$= \frac{-1}{2\sqrt{4ac - b^2}} \left(-\pi - 2\arctan\left(\frac{b}{\sqrt{4ac - b^2}} \right) \right). \tag{23}$$

Eliminating minus signs and substituting $D = b^2 - 4ac$, we obtain the stated result. ◀

Next, we begin by restating the collision probability between two vectors in terms of half-normal vectors.

▶ **Lemma 11** (Towards three-dimensional large deviations)**.** *Let $\mathcal{H}$ denote the hypercube hash family in d dimensions, and as before, let p be defined as:*

$$p(\theta) = \mathbb{P}_{h \sim \mathcal{H}}(h(\boldsymbol{x}) = h(\boldsymbol{y}) \mid \phi(\boldsymbol{x}, \boldsymbol{y}) = \theta). \tag{24}$$

Let $\hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}} \sim \mathcal{H}(0,1)^d$ and let the sequence $\{\boldsymbol{Z}_d\}_{d \in \mathbb{N}} \subset \mathbb{R}^3$ be defined as:

$$\boldsymbol{Z}_d = \frac{1}{d} \left(\sum_{i=1}^d \hat{X}_i \hat{Y}_i, \sum_{i=1}^d \hat{X}_i^2, \sum_{i=1}^d \hat{Y}_i^2 \right). \tag{25}$$

Then:

$$p(\theta) = \left(\frac{1}{2\sin\theta} \right)^{d + o(d)} \max_{x,y>0} \mathbb{P}(\boldsymbol{Z}_d = (xy\cos\theta,\, x^2,\, y^2)). \tag{26}$$

Proof. First, we write out the definition of the conditional probability in p, and use the fact that each of the 2^d hash regions (orthants) has the same probability mass. Here

$\boldsymbol{X}, \boldsymbol{Y} \sim \mathcal{N}(0,1)^d$ denote random Gaussian vectors, and subscripts denoting what probabilities are computed over are omitted when implicit.

$$p(\theta) = \mathbb{P}_{h\sim\mathcal{H}}(h(\boldsymbol{x}) = h(\boldsymbol{y}) \mid \phi(\boldsymbol{x},\boldsymbol{y}) = \theta) \tag{27}$$

$$= 2^d \cdot \mathbb{P}_{\boldsymbol{X},\boldsymbol{Y}\sim\mathcal{N}(0,1)^d}(\boldsymbol{X} > 0,\, \boldsymbol{Y} > 0 \mid \phi(\boldsymbol{X},\boldsymbol{Y}) = \theta) \tag{28}$$

$$= \frac{2^d \cdot \mathbb{P}(\boldsymbol{X} > 0,\, \boldsymbol{Y} > 0,\, \phi(\boldsymbol{X},\boldsymbol{Y}) = \theta)}{\mathbb{P}(\phi(\boldsymbol{X},\boldsymbol{Y}) = \theta)}\,. \tag{29}$$

By Lemma 2, the denominator is equal to $(\sin\theta)^{d+o(d)}$. The numerator of (29) can further be rewritten as a conditional probability on $\{\boldsymbol{X} > 0, \boldsymbol{Y} > 0\}$, multiplied with $\mathbb{P}(\boldsymbol{X} > 0, \boldsymbol{Y} > 0) = 2^{-2d}$. To incorporate the conditionals $\boldsymbol{X}, \boldsymbol{Y} > 0$, we replace $\boldsymbol{X}, \boldsymbol{Y} \sim \mathcal{N}(0,1)^d$ by half-normal vectors $\hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}} \sim \mathcal{H}(0,1)^d$, resulting in:

$$p(\theta) = \frac{\mathbb{P}_{\hat{\boldsymbol{X}},\hat{\boldsymbol{Y}}\sim\mathcal{H}(0,1)^d}(\phi(\hat{\boldsymbol{X}},\hat{\boldsymbol{Y}}) = \theta)}{(2\sin\theta)^{d+o(d)}} = \frac{q(\theta)}{(2\sin\theta)^{d+o(d)}}\,. \tag{30}$$

To incorporate the normalization over the (half-normal) vectors $\hat{\boldsymbol{X}}$ and $\hat{\boldsymbol{Y}}$, we introduce dummy variables x, y corresponding to the norms of $\hat{\boldsymbol{X}}/\sqrt{d}$ and $\hat{\boldsymbol{Y}}/\sqrt{d}$, and observe that as the probabilities are exponential in d, the integrals will be dominated by the maximum value of the integrand in the given range:

$$q(\theta) = \int_0^\infty \int_0^\infty \mathbb{P}(\langle\hat{\boldsymbol{X}},\hat{\boldsymbol{Y}}\rangle = x\,y\,d\,\cos\theta, \|\hat{\boldsymbol{X}}\|^2 = x^2 d, \|\hat{\boldsymbol{Y}}\|^2 = y^2 d)\,dx\,dy \tag{31}$$

$$= 2^{o(d)} \max_{x,y>0} \mathbb{P}\Big(\langle\hat{\boldsymbol{X}},\hat{\boldsymbol{Y}}\rangle = x\,y\,d\,\cos\theta, \|\hat{\boldsymbol{X}}\|^2 = x^2 d, \|\hat{\boldsymbol{Y}}\|^2 = y^2 d\Big)\,. \tag{32}$$

Substituting $\boldsymbol{Z}_d = \frac{1}{d}(\langle\hat{\boldsymbol{X}},\hat{\boldsymbol{Y}}\rangle, \|\hat{\boldsymbol{X}}\|^2, \|\hat{\boldsymbol{Y}}\|^2)$, we obtain the claimed result. ◀

Note that Z_1, Z_2, Z_3 are pairwise but not jointly independent. To compute the density of $\boldsymbol{Z}_d$ at $(xy\cos\theta, x^2, y^2)$ for $d \to \infty$, we use the Gärtner-Ellis theorem stated in Lemma 5.

▶ **Lemma 12** (Applying the Gärtner-Ellis theorem to $\boldsymbol{Z}_d$)**.** *Let $\{\boldsymbol{Z}_d\}_{d\in\mathbb{N}} \subset \mathbb{R}^3$ as in Lemma 11, and let Λ and Λ^* as in Section 2. Then $\mathbf{0}$ lies in the interior of $\mathcal{D}_\Lambda$, and therefore*

$$\mathbb{P}(\boldsymbol{Z}_d = (xy\cos\theta,\, x^2,\, y^2)) = \exp\left(-\Lambda^*(xy\cos\theta, x^2, y^2)d + o(d)\right). \tag{33}$$

Essentially, all that remains now is computing Λ^* at the appropriate point $\boldsymbol{z}$. To continue, we first compute the logarithmic moment generating function $\Lambda = \Lambda_d$ of $\boldsymbol{Z}_d$:

▶ **Lemma 13** (Computing Λ)**.** *Let $\boldsymbol{Z}_d$ as before, and let $D = D(\lambda_1,\lambda_2,\lambda_3) = \lambda_1^2 - (1 - 2\lambda_2)(1-2\lambda_3)$. Then for $\boldsymbol{\lambda} \in \mathcal{D}_\Lambda = \{\boldsymbol{\lambda} \in \mathbb{R}^3 : \lambda_2, \lambda_3 < \frac{1}{2}, D < 0\}$ we have:*

$$\Lambda(\boldsymbol{\lambda}) = \ln\left(\pi + 2\arctan\left(\frac{\lambda_1}{\sqrt{-D}}\right)\right) - \ln\pi - \tfrac{1}{2}\ln(-D). \tag{34}$$

Proof. By the definition of the LMGF, we have:

$$\Lambda(\boldsymbol{\lambda}) = \ln \mathbb{E}_{\hat{X}_1,\hat{Y}_1\sim\mathcal{H}(0,1)}\left[\exp\left(\lambda_1\hat{X}_1\hat{Y}_1 + \lambda_2\hat{X}_1^2 + \lambda_3\hat{Y}_1^2\right)\right]. \tag{35}$$

We next compute the inner expectation over the random variables $\hat{X}_1, \hat{Y}_1$, by writing out the double integral over the product of the argument with the densities of $\hat{X}_1$ and $\hat{Y}_1$.

$$\mathbb{E}_{X_1,Y_1}\left[\exp\left(\lambda_1 X_1 Y_1 + \lambda_2 X_1^2 + \lambda_3 Y_1^2\right)\right] \tag{36}$$

$$= \int_0^\infty \sqrt{\frac{2}{\pi}}\exp\left(-\frac{x^2}{2}\right)dx \int_0^\infty \sqrt{\frac{2}{\pi}}\exp\left(-\frac{y^2}{2}\right)dy\,\exp\left(\lambda_1 xy + \lambda_2 x^2 + \lambda_3 y^2\right) \tag{37}$$

$$= \frac{2}{\pi}\int_0^\infty\int_0^\infty \exp\left(\lambda_1 xy + \left(\lambda_2 - \tfrac{1}{2}\right)x^2 + \left(\lambda_3 - \tfrac{1}{2}\right)y^2\right)\,dx\,dy\,. \tag{38}$$

Applying Lemma 10 with $(a,b,c) = (\lambda_2 - \frac{1}{2}, \lambda_1, \lambda_3 - \frac{1}{2})$ yields the claimed expression for Λ, as well as the bounds stated in $\mathcal{D}_\Lambda$ which are necessary for the expectation to be finite. ◀

We now continue with computing the Fenchel-Legendre transform of Λ, which involves a rather complicated maximization (supremum) over $\boldsymbol{\lambda} \in \mathbb{R}^3$. The following lemma makes a first step towards computing this supremum.

▶ **Lemma 14** (Computing $\Lambda^*(\boldsymbol{z})$ – General form). *Let $\boldsymbol{z} \in \mathbb{R}^3$ such that $z_2, z_3 > 0$. Then the Fenchel-Legendre transform Λ^* of Λ at $\boldsymbol{z}$ satisfies*

$$\Lambda^*(\boldsymbol{z}) = \ln \pi + \sup_{\substack{\lambda_1,\beta \\ \beta>1}} \left\{ \frac{z_2}{2} + \frac{z_3}{2} + \lambda_1 z_1 - |\lambda_1|\beta\sqrt{z_2 z_3} + \frac{1}{2}\ln(\beta^2-1) + \ln|\lambda_1| \right. \tag{39}$$

$$\left. - \ln\left(\pi + 2\arctan\left(\frac{\lambda_1}{|\lambda_1|\sqrt{\beta^2-1}}\right)\right)\right\}. \tag{40}$$

Proof. First, we recall the definition of Λ^* and substitute the previous expression for Λ:

$$\Lambda^*(\boldsymbol{z}) = \sup_{\boldsymbol{\lambda}\in\mathbb{R}^3} \{\langle \boldsymbol{\lambda}, \boldsymbol{z}\rangle - \Lambda(\boldsymbol{\lambda})\} \tag{41}$$

$$= \ln\pi + \sup_{\boldsymbol{\lambda}\in\mathbb{R}^3} \left\{ \langle \boldsymbol{\lambda}, \boldsymbol{z}\rangle + \ln\sqrt{-D} - \ln\left(\pi + 2\arctan\left(\frac{\lambda_1}{\sqrt{-D}}\right)\right)\right\}. \tag{42}$$

Here as before $D = \lambda_1^2 - (1-2\lambda_2)(1-2\lambda_3) < 0$. Let the argument of the supremum above be denoted by $f(\boldsymbol{z}, \boldsymbol{\lambda})$. We make a change of variables by setting $t_2 = 1 - 2\lambda_2 > 0$ and $t_3 = 1 - 2\lambda_3 > 0$, so that D becomes $D = \lambda_1^2 - t_2 t_3 < 0$:

$$f(\boldsymbol{z}, \lambda_1, t_2, t_3) = \frac{z_2}{2} + \frac{z_3}{2} + \lambda_1 z_1 - \frac{t_2 z_2}{2} - \frac{t_3 z_3}{2} \tag{43}$$

$$+ \tfrac{1}{2}\ln(t_2 t_3 - \lambda_1^2) - \ln\left(\pi + 2\arctan\left(\tfrac{\lambda_1}{\sqrt{t_2 t_3 - \lambda_1^2}}\right)\right). \tag{44}$$

We continue by making a further change of variables $u = t_2 t_3 > \lambda_1^2$ so that $t_2 = u/t_3$. As a result the dependence of f on t_3 is only through the fourth and fifth terms above, from which one can easily deduce that the supremum over t_3 occurs at $t_3 = \sqrt{u z_2/z_3}$. This also implies that $t_2 = \sqrt{u z_3/z_2}$. Substituting these values for t_2, t_3, we obtain:

$$f(\boldsymbol{z}, \lambda_1, u) = \tfrac{z_2}{2} + \tfrac{z_3}{2} + \lambda_1 z_1 - \sqrt{u z_2 z_3} + \tfrac{1}{2}\ln(u - \lambda_1^2) - \ln\left(\pi + 2\arctan\left(\tfrac{\lambda_1}{\sqrt{u-\lambda_1^2}}\right)\right).$$

Finally, we use the substitution $u = \beta^2 \cdot \lambda_1^2$. From $D < 0$ it follows that $u/\lambda_1^2 = \beta > 1$. This substitution and some rewriting of f leads to the claimed result. ◀

The previous simplifications were regardless of z_1, z_2, z_3, where the only assumption that was made during the optimization of t_3 was that $z_2, z_3 > 0$. In our application, we want to compute Λ^* at $\boldsymbol{z} = (xy\cos\theta, x^2, y^2)$ for certain $x, y > 0$ and $\theta \in (0, \frac{\pi}{2})$. Substituting these values for $\boldsymbol{z}$, the expression from Lemma 11 becomes:

$$\Lambda^*(xy\cos\theta, x^2, y^2) = \ln\pi + \frac{x^2}{2} + \frac{y^2}{2} + \sup_{\substack{\lambda_1,\beta \\ \beta>1}} \left\{ (\lambda_1\cos\theta - |\lambda_1|\beta)xy + \frac{1}{2}\ln(\beta^2-1) \right. \tag{45}$$

$$\left. + \ln|\lambda_1| - \ln\left(\pi + 2\arctan\left(\frac{\lambda_1}{|\lambda_1|\sqrt{\beta^2-1}}\right)\right)\right\}. \tag{46}$$

The remaining optimization over λ_1, β now takes slightly different forms depending on whether $\lambda_1 < 0$ or $\lambda_1 > 0$. We will tackle these two cases separately, based on the identity:

$$\Lambda^*(\boldsymbol{z}) = \max\left\{ \sup_{\substack{\boldsymbol{\lambda}\in\mathbb{R}^3 \\ \lambda_1>0}} \{\langle \boldsymbol{\lambda}, \boldsymbol{z}\rangle - \Lambda(\boldsymbol{\lambda})\}, \sup_{\substack{\boldsymbol{\lambda}\in\mathbb{R}^3 \\ \lambda_1<0}} \{\langle \boldsymbol{\lambda}, \boldsymbol{z}\rangle - \Lambda(\boldsymbol{\lambda})\} \right\} = \max\{\Lambda^*_+(\boldsymbol{z}), \Lambda^*_-(\boldsymbol{z})\}.$$

▶ **Lemma 15** (Computing $\Lambda^*(\boldsymbol{z})$ for positive λ_1)**.** *Let $\boldsymbol{z} = (xy\cos\theta, x^2, y^2)$ with $x, y > 0$ and $\theta \in (0, \frac{\pi}{2})$. For $\theta \in (0, \arccos\frac{2}{\pi})$, let $\beta_0 = \beta_0(\theta) \in (1, \infty)$ be the unique solution to (1). Then the Fenchel-Legendre transform Λ^* at $\boldsymbol{z}$, restricted to $\lambda_1 > 0$, satisfies*

$$\Lambda^*_+(\boldsymbol{z}) = \frac{x^2}{2} + \frac{y^2}{2} - 1 - \ln(xy) + \begin{cases} \ln\left(\frac{\pi\beta_0(\beta_0\cos\theta - 1)}{2(\beta_0 - \cos\theta)^2}\right), & \text{if } \theta \in (0, \arccos\frac{2}{\pi}); \\ 0, & \text{if } \theta \in [\arccos\frac{2}{\pi}, \frac{\pi}{2}). \end{cases} \tag{47}$$

Proof. Substituting $\lambda_1 > 0$ into (46), we obtain:

$$\Lambda^*_+(xy\cos\theta, x^2, y^2) = \ln\pi + \frac{x^2}{2} + \frac{y^2}{2} + \sup_{\substack{\lambda_1>0 \\ \beta>1}} \Big\{ g_+(\lambda_1, \beta) \Big\}, \tag{48}$$

$$g_+(\lambda_1, \beta) = (\cos\theta - \beta)\lambda_1 xy + \frac{\ln(\beta^2 - 1)}{2} + \ln\lambda_1 - \ln\left(\pi + 2\arctan\left(\frac{1}{\sqrt{\beta^2 - 1}}\right)\right). \tag{49}$$

Differentiating w.r.t. λ_1 gives $(\cos\theta - \beta)xy + \frac{1}{\lambda_1}$. Recall that $\beta > 1 > \cos\theta$. For $\lambda_1 \to 0^+$ the derivative is therefore positive, for $\lambda_1 \to \infty$ it is negative, and there is a global maximum at the only root $\lambda_1 = 1/((\beta - \cos\theta)xy)$. In that case, the expression further simplifies and we can pull out more terms that do not depend on β, to obtain:

$$\Lambda^*_+(xy\cos\theta, x^2, y^2) = \ln\pi + \frac{x^2}{2} + \frac{y^2}{2} - 1 - \ln(xy) + \sup_{\beta>1} \Big\{ g_+(\beta) \Big\}, \tag{50}$$

$$g_+(\beta) = \ln\left(\frac{\sqrt{\beta^2 - 1}}{(\beta - \cos\theta)\left(\pi + 2\arcsin\frac{1}{\beta}\right)}\right) = \ln h_+(\beta). \tag{51}$$

Here we used the identity $\arctan(1/\sqrt{\beta^2 - 1}) = \arcsin(1/\beta)$. Now, for $\beta \to 1^+$ we have $h_+(\beta) \to 0^+$, while for $\beta \to \infty$, we have

$$h_+(\beta) = \frac{1}{\pi} + \frac{1}{\pi\beta}\left(\cos\theta - \frac{2}{\pi}\right) + O\left(\frac{1}{\beta^2}\right). \tag{52}$$

In other words, if $\cos\theta \le \frac{2}{\pi}$ or $\theta \ge \arccos\frac{2}{\pi}$, we have $h_+(\beta) \to (\frac{1}{\pi})^-$ (the second order term is negative for $\cos\theta = \frac{2}{\pi}$), while for $\theta < \arccos\frac{2}{\pi}$ we approach the same limit from above as $h_+(\beta) \to (\frac{1}{\pi})^+$. For $\theta < \arccos\frac{2}{\pi}$ there is a non-trivial maximum at some value $\beta = \beta_0 \in (1, \infty)$, while for $\theta \ge \arccos\frac{2}{\pi}$, we can see from the derivative $h'_+(\beta)$ that $h_+(\beta)$ is strictly increasing on $(1, \infty)$, and the supremum is attained at $\beta \to \infty$. We therefore obtain two different results, depending on whether $\theta < \arccos\frac{2}{\pi}$ or $\theta \ge \arccos\frac{2}{\pi}$.

Case 1: $\arccos\frac{2}{\pi} \le \theta < \frac{\pi}{2}$. The supremum is attained in the limit of $\beta \to \infty$, which leads to $h_+(\beta) \to \frac{1}{\pi}$ and the stated expression for $\Lambda^*_+(xy\cos\theta, x^2, y^2)$.

Case 2: $0 < \theta < \arccos\frac{2}{\pi}$. In this case there is a non-trivial maximum at some value $\beta = \beta_0$, namely there where the derivative $h'_+(\beta_0) = 0$. After computing the derivative, eliminating the (positive) denominator and rewriting, this condition is equivalent to (1). This allows us to rewrite g and Λ^* in terms of β_0, by substituting the given expression for $\arcsin\left(\frac{1}{\beta_0}\right)$, which ultimately leads to the stated formula for Λ^*_+. ◀

▶ **Lemma 16** (Computing $\Lambda^*(\boldsymbol{z})$ for negative λ_1)**.** *Let $\boldsymbol{z} = (xy\cos\theta, x^2, y^2)$ with $x, y > 0$ and $\theta \in (0, \frac{\pi}{2})$. For $\theta \in (\arccos\frac{2}{\pi}, \frac{\pi}{3})$, let $\beta_1 \in (1, \infty)$ be the unique solution to (1). Then the Fenchel-Legendre transform Λ^* at $\boldsymbol{z}$, restricted to $\lambda_1 < 0$, satisfies*

$$\Lambda^*_-(\boldsymbol{z}) = \frac{x^2}{2} + \frac{y^2}{2} - 1 - \ln(xy) + \begin{cases} 0, & \text{if } \theta \in (0, \arccos\frac{2}{\pi}]; \\ \ln\left(\frac{\pi\beta_1(\beta_1\cos\theta+1)}{2(\cos\theta+\beta_1)^2}\right), & \text{if } \theta \in (\arccos\frac{2}{\pi}, \frac{\pi}{3}); \\ \ln\left(\frac{\pi}{2(1+\cos\theta)}\right), & \text{if } \theta \in [\frac{\pi}{3}, \frac{\pi}{2}). \end{cases} \tag{53}$$

Proof. We again start by substituting $\lambda_1 < 0$ into (46):

$$\Lambda^*_-(xy\cos\theta, x^2, y^2) = \ln\pi + \frac{x^2}{2} + \frac{y^2}{2} + \sup_{\substack{\lambda_1<0 \\ \beta>1}}\Big\{g_-(\lambda_1,\beta)\Big\}, \tag{54}$$

$$g_-(\lambda_1,\beta) = (\cos\theta+\beta)\lambda_1 xy + \frac{\ln(\beta^2-1)}{2} + \ln(-\lambda_1) - \ln\left(\pi + 2\arctan\left(\frac{-1}{\sqrt{\beta^2-1}}\right)\right).$$

Differentiating w.r.t. λ_1 gives $(\cos\theta+\beta)xy + \frac{1}{\lambda_1}$. For $\lambda_1 \to -\infty$ this is positive, for $\lambda_1 \to 0^-$ this is negative, and so the maximum is at $\lambda_1 = -1/((\cos\theta+\beta)xy)$. Substituting this value for λ_1, and pulling out terms which do not depend on β yields:

$$\Lambda^*_-(xy\cos\theta, x^2, y^2) = \ln\left(\frac{\pi}{2}\right) + \frac{x^2}{2} + \frac{y^2}{2} - 1 - \ln(xy) + \sup_{\beta>1}\Big\{g_-(\beta)\Big\}, \tag{55}$$

$$g_-(\beta) = \ln\left(\frac{\sqrt{\beta^2-1}}{(\cos\theta+\beta)\arccos\frac{1}{\beta}}\right) = \ln h_-(\beta).$$

Above we used the identity $\pi + 2\arctan(-1/\sqrt{\beta^2-1}) = 2\arccos\frac{1}{\beta}$, where the factor 2 has been pulled outside the supremum. Now, differentiating h_- w.r.t. β results in:

$$h'_-(\beta) = \frac{\beta\sqrt{\beta^2-1}(\beta\cos\theta+1)\arccos\frac{1}{\beta} - \left(\beta^2-1\right)(\cos\theta+\beta)}{\beta\left(\beta^2-1\right)(\cos\theta+\beta)^2\arccos\frac{1}{\beta}}. \tag{56}$$

Clearly the denominator is positive, while for $\beta \to 1^+$ the limit is negative iff $\cos\theta < \frac{1}{2}$. For $\beta \to \infty$ we further have $h'_-(\beta) \to 0^-$ for $\cos\theta \le \frac{2}{\pi}$ and $h'_-(\beta) \to 0^+$ for $\cos\theta > \frac{2}{\pi}$. We therefore analyze three cases separately below.

Case 1: $\frac{\pi}{3} \le \theta < \frac{\pi}{2}$. In this parameter range, $h'_-(\beta)$ is negative for all $\beta > 1$, and the supremum lies at $\beta \to 1^+$ with limiting value $h_-(\beta) \to \frac{1}{1+\cos\theta}$. This yields the given expression for Λ^*_-.

Case 2: $\arccos\frac{2}{\pi} < \theta < \frac{\pi}{3}$. For θ in this range, $h'_-(\beta)$ is positive for $\beta \to 1^+$ and negative for $\beta \to \infty$, and changes sign exactly once, where it attains its maximum. After some rewriting, we find that this is at the value $\beta = \beta_1(\theta) \in (1, \infty)$ satisfying the relation from (1). Substituting this expression for $\arccos\frac{1}{\beta_1}$ into h_-, we obtain the result for Λ^*_-.

Case 3: $0 < \theta \le \arccos\frac{2}{\pi}$. In this case h'_- is positive for all $\beta > 1$, and the supremum lies at $\beta \to \infty$. For $\beta \to \infty$ we have $h_-(\beta) \to \frac{2}{\pi}$ (regardless of θ) and we therefore get the final claimed result. ◀

Proof of Theorem 1. Combining the previous two results with Lemma 12 and Equation 47, we obtain explicit asymptotics for $\mathbb{P}(\boldsymbol{Z}_d \approx (xy\cos\theta,\, x^2,\, y^2))$. What remains is a maximization over $x, y > 0$ of p, which translates to a minimization of Λ^*. As $\frac{x^2}{2} + \frac{y^2}{2} - 1 - \ln(xy)$ attains its minimum at $x = y = 1$ with value 0, we obtain Theorem 1. ◀

B Proof of Proposition 9

We will assume the reader is familiar with (the notation from) [23]. Let $t = 2^{c_t d + o(d)}$ denote the number of hash tables, and $n = (4/3)^{d/2+o(d)}$. Going through the proofs of [23, Appendix A] and replacing the explicit instantiation of the collision probabilities $(1 - \theta/\pi)$ by an arbitrary function $p(\theta)$, we get that the optimal number of hash functions concatenated into one function for each hash table, denoted k, satisfies

$$k = \frac{\ln t}{-\ln p(\theta_1)} = \frac{c_t d}{d' \log_2(\pi/\sqrt{3})}. \tag{57}$$

The latter equality follows when substituting $\theta_1 = \pi/3$ and substituting the collision probabilities for partial hypercube LSH in some dimension $d' \leq d$. As we need $k \geq 1$, the previous relation translates to a condition on d' as $d' \leq \frac{c_t}{\log_2(\pi/\sqrt{3})} d$. As we expect the collision probabilities to be closer to those of full-dimensional hypercube LSH when d' is closer to d, we replace the above inequality by an equality, and what remains is finding the minimum value c_t satisfying the given constraints.

By carefully checking the proofs of [23, Appendix A.2-A.3], the exact condition on c_t to obtain the minimum asymptotic time complexity is the following:

$$-c_n = \max_{\theta_2 \in (0,\pi)} \left\{ \log_2 \sin\theta_2 + \frac{c_t}{\rho(\frac{\pi}{3}, \theta_2)} \right\}. \tag{58}$$

Here $c_n = \frac{1}{2}\log_2(\frac{4}{3}) \approx 0.20752$, and $\rho(\theta_1, \theta_2) = \ln p(\theta_1)/\ln p(\theta_2)$ corresponds to the exponent ρ for given angles θ_1, θ_2. Note that in the above equation, only c_t is an unknown. Substituting the asymptotic collision probabilities from Theorem 1, we find a solution at $c_t \approx 0.11464$, with maximizing angle $\theta_2 \approx 0.45739\pi$. This corresponds to a time and space complexity of $(n \cdot t)^{1+o(1)} = 2^{(c_n + c_t)d + o(d)} \approx 2^{0.32216d + o(d)}$ as claimed.

Generalized Predecessor Existence Problems for Boolean Finite Dynamical Systems

Akinori Kawachi[1], Mitsunori Ogihara[2], and Kei Uchizawa[3]

1 **Graduate School of Engineering, Osaka University, Osaka, Japan**
2 **Department of Computer Science, University of Miami, Coral Gables, FL, USA**
3 **Graduate School of Science and Engineering, Yamagata University, Yamagata, Japan**

Abstract

A Boolean Finite Synchronous Dynamical System (BFDS, for short) consists of a finite number of objects that each maintains a boolean state, where after individually receiving state assignments, the objects update their state with respect to object-specific time-independent boolean functions synchronously in discrete time steps. The present paper studies the computational complexity of determining, given a boolean finite synchronous dynamical system, a configuration, which is a boolean vector representing the states of the objects, and a positive integer t, whether there exists another configuration from which the given configuration can be reached in t steps. It was previously shown that this problem, which we call the t-Predecessor Problem, is NP-complete even for $t = 1$ if the update function of an object is either the conjunction of arbitrary fan-in or the disjunction of arbitrary fan-in.

This paper studies the computational complexity of the t-Predecessor Problem for a variety of sets of permissible update functions as well as for polynomially bounded t. It also studies the t-Garden-Of-Eden Problem, a variant of the t-Predecessor Problem that asks whether a configuration has a t-predecessor, which itself has no predecessor. The paper obtains complexity theoretical characterizations of all but one of these problems.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Computational complexity, dynamical systems, Garden of Eden, predecessor

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.8

1 Introduction

A *dynamical system* is a time-dependent network of objects that models evolution, where each object holds a state value that is an element of a state set. The configuration of the system is the collective state of the objects and is the vector that assembles the states of all the objects in a certain order. Given an initial configuration, the system evolves over time through state update, where the state of an object is updated by a function that takes as input state values of some nodes, possibly its own state value. Variants of dynamical systems can be defined by considering the state set (binary, discrete, countably infinite, and uncountable), the types of permissible update functions, whether the number of states is fixed, and the order in which the updates are performed (either in a fixed order or all at the same time).

The simplest dynamical systems are those with the boolean state set, a fixed finite number of objects, and synchronous updates, where the state update function does not depend on the time distance from the start. These systems are called *boolean finite dynamical systems*

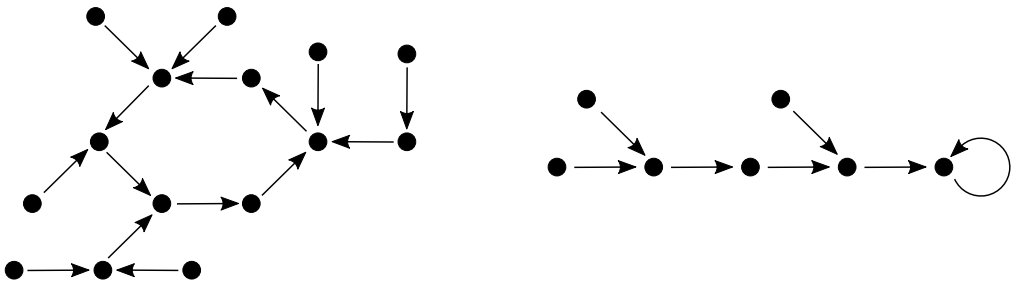


Figure 1 Configuration space with a loop, a fixed point, and flows into them. Dots represent configurations, and arrows do transitions. A node of in-degree 0 represents a Garden of Eden.

(BFDS) [1]. Quite often, the BFDS model is further simplified by assuming that the update functions are chosen from a collection of templates, such as the exclusive-or, the negation, the conjunction, and the disjunction.

Given a BFDS $\mathcal{F}$ of n objects, the number of possible configurations of the system is 2^n. Due to the assumptions that the updates are synchronized and that the update function does not change over time, the imposed finiteness of the system configuration space leads to important facts: for each initial configuration $\mathbf{a}$, the system starting from $\mathbf{a}$ either converges to a fixed point or enters some loop having length ≥ 2 and that the convergence or the entrance to a loop takes place within 2^n steps from departure (see Figure 1). These facts mean that the dichonomical fate of an initial configuration in a BFDS can be tested in the linear space. In fact, the fate-determination problem (or the convergence problem) as well as its variant, the reachability problem (whether a configuration be reached from another configuration with respect to a given BFDS), is PSPACE-complete if a complete boolean basis is available for building update functions and the complexity is lower for both problems otherwise [3]. The BDFS offers a rich theory not only in terms of fixed points, reachability, and cycles, but also in terms of the reversal, that is, the action of going back in time starting from a given configuration. Since the system changes two distinct configurations the same configuration with a single update, so a given configuration may have multiple predecessors. Also, there may exist configurations without predecessors. We call such a configuration *Garden of Eden (GOE)* (see Figure 1).

If the update functions are each polynomial time computable, which is indeed the case where the functions are chosen from a predetermined set of templates, testing whether a given configuration of a BFDS has a predecessor can be answered in NP, and thus, whether the configuration is a GOE can be answered in coNP. In fact, it is NP-complete to decide whether a configuration is *not* a GOE [2]. This paper makes a deeper investigation into this problem by asking how much simplification can be given the template set for update functions to retain this completeness. We show: if the templates are either only conjunction or only disjunction the GOE problem is in AC^0; if the templates are the two-fan-in conjunction and the two-fan-in disjunction, then the GOE problem is NL-complete; if the templates are either the combination of the two-fan-in conjunction and the three-fan-in disjunction or the combination of the two-fan-in disjunction and the three-fan-in conjunction, then the GOE problem is coNP-complete.

We generalize the GOE Problem further in two ways. First, for an integer $t \geq 1$, we ask whether we can go back from a given configuration successively t times, by cleverly choosing at each time one of the possible predecessors, if any at all. We call this problem the t-Predecessor Existence Problem (the t-PRED Problem, for short). Second, for an integer

$t \geq 0$, we ask whether we can go back from a given configuration successively t times and arrive at a GOE, by cleverly choosing at each time one of the possible predecessors, if any at all. We call this problem the t-Garden Of Eden Existence Problem (the t-GOE Problem, for short). The GOE Problem we mentioned earlier is indeed the 0-GOE Problem in this extension. It is easy to see that the 1-PRED Problem is exactly complementary to the 0-GOE Problem; but, for $t \geq 2$, the t-PRED is not necessarily complementary to the $(t-1)$-GOE Problem.

In this paper we ask the complexity of these two extensions with the functions restricted to be disjunction and conjunction. Except for the 1-GOE Problem with 2-fan-in disjunction and 2-fan-in conjunction, we obtain complete characterization of the constant-bounded as well as the polynomial-bounded t-PRED Problem and t-GOE Problem for all the templates consisting of conjunction and disjunction (see Table 1 in Section 2).

We note here that the papers [5, 6] show that the problem of computing fixed points in a BFDS exhibits a curious dichotomy between P and #P-complete and that [8] studies the problem of calculating the length of a cycle that an initial configuration is eventually taken to, and shows that the problem is polynomial-time solvable for some template sets, computable in UP for some, and PSPACE-complete for some. Along with these prior papers, our paper shows BFDS offers a rich theory of computational complexity.

This paper is organized as follows: In the next section we go over the definitions and prove some useful lemmas and propositions. We then show the results on the predecessor existence problems in Section 3 and the results on the Garden of Eden problems in Section 4. We will conclude the paper in Section 5.

2 Preliminaries

For an integer $n \geq 1$, a *synchronous boolean finite dynamical system* (synchronous BFDS, for short) $\mathcal{F}$ of n objects consists of n variables $x_1, \ldots, x_n$ and n boolean functions $(f_1, f_2, \ldots, f_n)$ such that for each i, $1 \leq i \leq n$, f_i is a boolean function that takes input from $x_1, \ldots, x_n$. A *state configuration* $\mathbf{a}$ (or simply a *configuration*) of $\mathcal{F}$ is an n-dimensional boolean vector and for each variable x, $\mathbf{a}[x] \in \{0, 1\}$ represents the component of $\mathbf{a}$ corresponding to x. The action of $\mathcal{F}$ on an state configuration $\mathbf{x}$ is defined by: $\mathcal{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x}))$. In other words, the elements of $\mathcal{F}(\mathbf{x})$ are obtained by applying the n boolean functions $f_1, \ldots, f_n$ concurrently on the variables $x_1, \ldots, x_n$.

In the remainder of the paper, boolean dynamical systems are defined without giving an explicit ordering among the objects. We will use the notation $\mathcal{F}[x]$ to mean the function of the dynamical system $\mathcal{F}$ for the object x.

Given an initial state configuration $\mathbf{x}^0$, the synchronous BFDS generates a sequence of state configurations by iterative applications of $\mathcal{F}$: For all $t \geq 0$, $\mathbf{x}^{t+1} = \mathcal{F}(\mathbf{x}^t)$, where $\mathbf{x}^t = (x_1^t, x_2^t, \ldots, x_n^t)$. In other words, for all $t \geq 0$, $\mathbf{x}^t = \mathcal{F}^t(\mathbf{x}^0)$.

Although we use the notation to mean that all the n variables are fed to each f_i, in reality some f_i may depend on a proper subset of the variables. Let g be a boolean function possibly with arity less than n. We say that f_i has template g to mean that f_i is equivalent to g with input variables properly chosen from $x_1, \ldots, x_n$. Given a collection of boolean functions, we say that $\mathcal{F}$ *has template set* $\mathcal{B}$ if each function in $\mathcal{F}$ has template in $\mathcal{B}$. We are interested in the following functions as template:

- id: this is the identity function with only one input that outputs the value of its input without changing it;
- AND_k, $k \geq 1$: this is the conjunction of arity k;
- OR_k, $k \geq 1$: this is the disjunction of arity k.

Note that for all $k \geq 2$ and $m < k$ AND_k (OR_k, respectively) can be used as a template for AND_m (OR_m, respectively) by repeating some of the inputs. Note also that both AND_1 and OR_1 are identical to id.

We are interested in the following template sets:

- $\mathcal{B}_{\mathrm{id}} = \{\mathrm{id}\}$,
- $\mathcal{B}_{2\mathrm{OR}} = \{\mathrm{OR}_2\}$ and $\mathcal{B}_{2\mathrm{AND}} = \{\mathrm{AND}_2\}$,
- $\mathcal{B}_{\mathrm{OR}} = \{\mathrm{OR}_k \mid k \geq 1\}$ and $\mathcal{B}_{\mathrm{AND}} = \{\mathrm{AND}_k \mid k \geq 1\}$,
- $\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}} = \{\mathrm{OR}_2, \mathrm{AND}_2\}$,
- $\mathcal{B}_{3\mathrm{OR},2\mathrm{AND}} = \{\mathrm{OR}_3, \mathrm{AND}_2\}$ and $\mathcal{B}_{2\mathrm{OR},3\mathrm{AND}} = \{\mathrm{OR}_2, \mathrm{AND}_3\}$.

Given a synchronous BFDS $\mathcal{F}$ and a configuration $\mathbf{a}$ of $\mathcal{F}$, we say that another configuration $\mathbf{b}$ is the *t-th predecessor of* $\mathbf{a}$, $t \geq 1$, if $\mathcal{F}^t(\mathbf{b}) = \mathbf{a}$. We will omit the word *first* in the case where $t = 1$. By convention, we define the 0-th predecessor of $\mathbf{a}$ to be $\mathbf{a}$ itself. We say that $\mathbf{a}$ is a Garden of Eden if $\mathbf{a}$ has no predecessor. We then consider the following problems.

- Let t, $t \geq 1$, be a fixed constant. Given a synchronous BFDS $\mathcal{F}$ with template set $\mathcal{B}$ and a configuration $\mathbf{a}$, the t-PRED Problem for $\mathcal{B}$ asks whether $\mathbf{a}$ has a t-th predecessor.
- Let t, $t \geq 0$, be a fixed constant. Given a synchronous BFDS $\mathcal{F}$ with template set $\mathcal{B}$, a configuration $\mathbf{a}$, the t-GOE Problem for $\mathcal{B}$ asks whether $\mathbf{a}$ has a t-th Garden of Eden, i.e., a t-th predecessor that is a Garden of Eden.

We also consider the polynomial version of the two problems.

- Given a synchronous BFDS $\mathcal{F}$ with template set $\mathcal{B}$, a configuration $\mathbf{a}$, and p presented in unary as 1^p, the Poly-PRED Problem for $\mathcal{B}$ asks whether $\mathbf{a}$ has a p-th predecessor.
- Given a synchronous BFDS $\mathcal{F}$ with template set $\mathcal{B}$, a configuration $\mathbf{a}$, and p presented in unary as 1^p, the Poly-GOE Problem for $\mathcal{B}$ asks whether $\mathbf{a}$ has a p-th Garden of Eden.

Note that as long as the predecessor existence and Garden of Eden problems go, by exchanging simultaneously between AND and OR and between true and false, any complexity result with respect to $\mathcal{B}_{\mathrm{AND}}$ holds with respect to $\mathcal{B}_{\mathrm{OR}}$. The same relation holds between $\mathcal{B}_{2\mathrm{AND}}$ and $\mathcal{B}_{2\mathrm{OR}}$ and between $\mathcal{B}_{2\mathrm{OR},3\mathrm{AND}}$ and $\mathcal{B}_{3\mathrm{OR},2\mathrm{AND}}$.

The 1-PRED and 0-GOE are generally called the Predecessor Existence Problem and the Garden-of-Eden Problem, respectively, and have been well studied. We note here that the synchronous BFDS often assumes that for each object its update function takes its state as part of the input; that is, x_i is one of the inputs to f_i. In this paper we remove that restriction, since if for all i it holds that f_i is either disjunction or conjunction whose inputs include x_i, the system $\mathcal{F}$ is monotone and converges within n steps, which gives little room for exploration.

For GOE Problem, in [2] it is shown that for the sequential dynamical systems (that is, the systems in which updates are performed one variable at a time with respect to a predetermined order), t-PRED is NP-complete even if the template set consists of AND's and OR's of any arity. The proof of this result does not directly imply the same result for the synchronous dynamical system.

In the case of sequential and synchronous dynamical systems, we have the following:

▶ **Proposition 1** ([2]). *The Poly-PRED Problem is solvable in polynomial time if $\mathcal{B}$ is one of the following: (i) ANDs of any fan-in and their negation, (ii) ORs of any fan-in and their negation, and (iii) XORs of any fan-in and their negation.*

For GOE Problem, the following is known:

▶ **Proposition 2** ([2]). 0-*GOE Problem is* coNP-*complete in general, but is solvable in polynomial time if* 1-*PRED Problem is solvable in polynomial time.*

Table 1 The results from this paper. The left panel is for the PRED Problems and the right panel is for the GOE problems. For "?" it is only known that the problem is NL-hard and in NP. The "-C" stands for "-complete".

PRED	t		
	1	≥ 2	poly
$\mathcal{B}_{\mathrm{id}}$	AC^0		DL-C
$\mathcal{B}_{2\mathrm{OR}}$, $\mathcal{B}_{2\mathrm{AND}}$			NL-C
$\mathcal{B}_{\mathrm{OR}}$, $\mathcal{B}_{\mathrm{AND}}$			
$\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}}$	NL-C	NP-C	
$\mathcal{B}_{2\mathrm{OR},3\mathrm{AND}}$ $\mathcal{B}_{3\mathrm{OR},2\mathrm{AND}}$	NP-C		

GOE	t			
	0	1	≥ 2	poly
$\mathcal{B}_{\mathrm{id}}$	AC^0			DL-C
$\mathcal{B}_{2\mathrm{OR}}$, $\mathcal{B}_{2\mathrm{AND}}$				NL-C
$\mathcal{B}_{\mathrm{OR}}$, $\mathcal{B}_{\mathrm{AND}}$		NP-C		
$\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}}$	NL-C	?		
$\mathcal{B}_{2\mathrm{OR},3\mathrm{AND}}$ $\mathcal{B}_{3\mathrm{OR},2\mathrm{AND}}$	coNP-C	Σ_2^p-C		

Table 1 summarizes the results shown in this paper.

We prove the following lemma, which will be useful in proving results on $\mathcal{B}_{\mathrm{OR}}$ and $\mathcal{B}_{2\mathrm{OR}}$.

▶ **Definition 3.** Let $\mathcal{F}$ be an n-variable synchronous BFDS, let $\mathbf{a}$ be a configuration of $\mathcal{F}$, and let $t \geq 1$ be an integer. Define a directed graph $G[\mathcal{F}, \mathbf{a}, t] = (V, E)$ with V partitioned into two groups K and L as follows:

- $V = V_0 \cup \ldots \cup V_t$ and for each i, $0 \leq i \leq t$, $V_i = \{v_{i,1}, \cdots, v_{i,n}\}$;
- $E = \{(v_{i,p}, v_{i+1,q}) \mid 0 \leq i \leq t-1$ and the function for x_p takes input from x_q $\}$.
- $L = L_0 \cup \ldots \cup L_t$, where $L_0 = \{v_{0,j} \mid$ the value of x_j in $\mathbf{a}$ is false $\}$ and for each i, $1 \leq i \leq t$, L_i is the set of all nodes in V_i that are reachable from L_0.
- $K = K_0 \cup \ldots \cup K_t$ and for each $i, 0 \leq i \leq t$, $K_i = V_i - L_i$; specifically, $K_0 = \{v_{0,j} \mid$ the value of x_j in $\mathbf{a}$ is true $\}$.

▶ **Lemma 4.** *Let $\mathcal{F}$ be an n-variable synchronous BFDS whose template set is the OR function. Let $t \geq 1$ and $\mathbf{a}$ a configuration of $\mathcal{F}$. Let $G = (V, E) = G[\mathcal{F}, \mathbf{a}, t]$ be as defined in the above and (K, L) be the partition of V. Then*

1. *$\mathbf{a}$ has a t-th predecessor if and only if for each $u \in K_0$ there is a path to a node in K_t that does not visit any node in L.*
2. *$\mathbf{a}$ has a t-th Garden-of-Eden predecessor if and only if there exist some $M = \{v_{t,j_1}, \ldots, v_{t,j_m}\} \subseteq V_t$ and $v_{t,j_0} \in K_t \backslash M (= V_t - (L_t \cup M))$ such that:*
   1. *for each $u \in K_0$ there is a path to a node in $K_t \setminus M$ that does not visit any node in $L \cup M$,*
   2. *each input of the function for x_{j_0} is an input of the function for one of $x_{j_1}, \cdots, x_{j_m}$, and*
   3. *the cardinality of M is no greater than the arity of the function for x_{j_0}.*

Proof. (The Predecessor Case) Suppose the condition in the lemma is satisfied. For each $u \in K_0$ choose one L-free path to some node in K_t. For each $i, 0 \leq i \leq t$, define $B_i = \{v_{i,j} \mid v_{i,j}$ appears on one of the chosen paths $\}$. Then $B_i \subseteq K_i$ for all i. For each $i, 0 \leq i \leq t$, define configuration $\mathbf{b}_i$ by setting the value of a variable x_j true if $v_{i,j} \in B_i$ and false otherwise. Then $\mathbf{b}_0 = \mathbf{a}$ and for all $i, 0 \leq i \leq t-1$, $\mathcal{F}(\mathbf{b}_{i+1}) = \mathbf{b}_i$. Thus, $\mathbf{b}_t$ is a t-th predecessor. On the other hand, suppose $\mathbf{a}$ has a t-th predecessor. Let $\mathbf{b}_0 = \mathbf{a}$. Select configurations $\mathbf{b}_1, \ldots, \mathbf{b}_t$ so that for each $i, 0 \leq i \leq t-1$, $\mathbf{b}_{i+1}$ is a predecessor of $\mathbf{b}_i$. For each $i, 0 \leq i \leq t$, let $S_i = \{v_{i,j} \mid \mathbf{b}_i$ assigns true to $x_{i,j}$ $\}$. Then $S_0 = K_0$. Because of the predecessor relations, for each $i, 1 \leq i \leq t$, $S_i \subseteq K_i$ and each node $u \in S_i$ has at least one incoming edge from S_{i-1}. Thus, the property holds for $\mathcal{F}$, $\mathbf{a}$, and t.

(The Garden Of Eden Case) Suppose that the conditions stated in the lemma hold. As before for each $u \in K_0$ choose a path that is free on $M \cup L$ but ensure that one of the paths arrive at some node in $K_t \setminus M$. Then, following the argument as before, the configurations induced by the path nodes form a series of t configurations arriving at $\mathbf{a}$. For $\mathbf{b}_t$, the value of x_{j_0} is true and the values of $x_{j_1}, \ldots, x_{j_m}$ are false. Each input of x_{j_0} is also an input of one of $x_{j_1}, \ldots, x_{j_m}$. Any predecessor of $\mathbf{b}_t$ must assign true to one of the inputs of x_{j_0} and must assign false to all of the inputs of $x_{j_1}, \ldots, x_{j_m}$, but that is not possible. Thus, there is no predecessor of $\mathbf{b}_t$.

On the other hand, suppose that there is a t-th predecessor of $\mathbf{a}$ that is a Garden of Eden. Select such one and define $\mathbf{b}_0, \ldots, \mathbf{b}_t$ and $S_0, \ldots, S_t$ as before. We have, as we have observed previously, for all $i, 0 \leq i \leq t$, $S_i \subseteq K_i$, and each node in $S_0 \cup \cdots \cup S_t$ is along an L-free path from K_0 (which is equal to S_0) to S_t. Let R be the variables that supply input to the variables corresponding to the nodes in $V_t - S_t$. In any predecessor of $\mathbf{b}_t$, the value of each variable in R must be false while each variable in $X - R$ can be set to true if needed. Then, that $\mathbf{b}_t$ is a Garden of Eden implies that there is some variable $u \in S_t$ all of whose input belong to R. Select one such u and for each input h of u, select a variable R that takes input from h. Construct M by placing the chosen variables from R. Then u and M satisfy the property in question. ◀

We have straightforward upper bounds on the predecessor and Garden-Of-Eden problems.

▶ **Proposition 5** ([2]). *Suppose the template set $\mathcal{B}$ consists only of polynomial-time computable boolean functions. Then the Poly-PRED Problem is in* NP.

▶ **Proposition 6.** *Suppose the template set $\mathcal{B}$ consists only of polynomial-time computable boolean functions. Then the Poly-GOE Problem is in Σ_2^p.*

The following proposition is useful to reduce the predecessor problems to Garden-Of-Eden problems.

▶ **Proposition 7.** *Given a synchronous BFDS $\mathcal{F}$, its configuration $\mathbf{a}$, and $t \geq 1$, we can add $t+2$ variables to $\mathcal{F}$ and $\mathbf{a}$ to create a new BFDS $\mathcal{F}'$ and $\mathbf{a}'$ so that:*

- *if $\mathbf{a}$ has a t-th predecessor in $\mathcal{F}$, then $\mathbf{a}'$ has a t-th predecessor in $\mathcal{F}'$ and none of its t-th predecessors have a predecessor; and*
- *if $\mathbf{a}$ does not have a t-th predecessor in $\mathcal{F}$, then $\mathbf{a}'$ does not have a t-th predecessor in $\mathcal{F}'$.*

Proof. Let $\mathcal{F}$, $\mathbf{a}$, and t be given. Introduce $t+2$ variables $e_0, \ldots, e_{t+1}$. We define the function of e_0 to be $\mathrm{id}(e_0)$ and the function for $e_i, 1 \leq i \leq t+1$, to be $\mathrm{id}(e_{i-1})$. This is $\mathcal{F}'$. We then add to $\mathbf{a}$ the values of e_i as all false except e_{t+1}. Then, for each i, $1 \leq i \leq t$, the i-th predecessor of the additional part has false for $e_0, \ldots, e_{t-i}$ and true for e_{t-i+1}. Specifically we have that the t-th predecessor on this part has false for e_0 and true for e_1. Since e_0 and e_1 take input from e_0 and is the identify function, clearly, such a t-th predecessor cannot have a predecessor. Since the new variables and functions are disjoint with those in the original, the new part does not affect the invertibility of the original part. This proves the proposition. ◀

The last general result we present in this section shows that if a predecessor problem with a certain template set (respectively, a GOE problem with a certain template set) for some t is hard a problem H, then the problem is hard for $t+1$. We omit the proof of this lemma.

▶ **Lemma 8.** *Let $\mathcal{B}$ be a template set containing the two-fan-in OR. Suppose there is a many-one reduction g from a problem H to the t-predecessor problem with $t \geq 1$ (respectively,*

the t-GOE problem with $t \geq 0$). Then there is a many-one reduction g' from H to the $(t+1)$-predecessor problem (respectively, the $(t+1)$-GOE problem). Furthermore, if g is logspace computable (polynomial-time computable) using $\mathcal{B}$ as the oracle, then so is g'.

3 The Complexity of Predecessor Problems

We first consider the case of $\mathcal{B}_{2\mathrm{OR},3\mathrm{AND}}$ and $\mathcal{B}_{3\mathrm{OR},2\mathrm{AND}}$, and prove that the problems are NP-complete.

▶ **Theorem 9.** *For $\mathcal{B}_{2\mathrm{OR},3\mathrm{AND}}$ and $\mathcal{B}_{3\mathrm{OR},2\mathrm{AND}}$, the t-PRED Problem is* NP*-complete for all constants $t \geq 1$.*

Proof. The inclusion in NP follows from Proposition 5.

We consider $\mathcal{B}_{3\mathrm{OR},2\mathrm{AND}}$ for NP-hardness and provide a polynomial-time many-one reduction from 3SAT to the 1-PRED Problem. Let φ be a 3CNF formula with n variables and m clauses. We introduce variables w, $c_1, \ldots, c_m$, $y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1}$, and $z_{1,0}, z_{1,1}, \ldots, z_{n,0}, z_{n,1}$. We associate $y_{i,1}$ with the positive literal of the i-th variable of y and $y_{i,0}$ with the negative literal of the i-th variable. We define the functions for these variables as follows:

- w, $y_{i,0}$, $y_{i,1}$: the function is $\mathrm{id}(w)$.
- $z_{i,0}$: $\mathrm{OR}(y_{i,0}, y_{i,1})$.
- $z_{i,1}$: $\mathrm{AND}(y_{i,0}, y_{i,1})$.
- c_j: Let the three literals of C_j be $y_{p,b}$, $y_{q,c}$, and $y_{r,d}$. Then the function is $\mathrm{OR}(y_{p,b}, y_{q,c}, y_{r,d})$.

We set the values of the variables in $\mathbf{a}$ to true for w, $y_{i,0}$, $y_{i,1}$, and $z_{i,0}$ and false for $z_{i,1}$.

Suppose $\mathbf{a}$ has a predecessor $\mathbf{b}$. Then for all i, since $z_{i,0}$ is true and $z_{i,1}$ is false in $\mathbf{a}$, in $\mathbf{b}$ exactly one of $y_{i,0}$ and $y_{i,1}$ is true and the values of y's in $\mathbf{b}$ can be viewed as a truth-assignment to the n variables of φ. Then, for all j, c_j is true in $\mathbf{a}$ and the inputs to the function for c_j correspond to the literals of C_j, it must be the case that the truth-assignment as represented by the y's in $\mathbf{b}$ form a satisfying assignment of φ. This means that φ is satisfiable.

On the other hand, suppose that φ is satisfiable. We take one satisfying assignment α of φ and build $\mathbf{b}$ by setting the values to y's according to α, true to w, and an arbitrary value to z's. Then $\mathcal{F}(\mathbf{b}) = \mathbf{a}$ and so $\mathbf{a}$ has a predecessor.

Clearly, this reduction can be computed in polynomial time, and so the theorem holds for $t = 1$. By combining this with Lemma 8, we obtain the proof for $t \geq 2$. ◀

We then consider the case of $\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}}$. In this case, the problem is tractable only when $t = 1$.

▶ **Theorem 10.** *For $\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}}$, the* 1*-PRED Problem is* NL*-complete and for all constants $t \geq 2$ the problem is* NP*-complete.*

Proof. It is easy to see that the above proof can be carried out for 2SAT with a logspace computable many-one reduction and so the 1-PRED Problem with $\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}}$ is NL-hard. To show that the problem is in NL, note that in the case of $\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}}$, given a configuration $\mathbf{a}$, the value assignments to its predecessor can be written as a 2CNF formula with possible single-literal clauses; e.g., $\mathrm{OR}(x, y) = \text{true}$ can be expressed as $x \vee y$.

For the 2-PRED Problem, the main idea is to break the computation of three-literal ORs into the OR to two two-variable ORs. We introduce alternating variables w_1 and w_2 whose functions are $\mathrm{id}(w_2)$ and $\mathrm{id}(w_1)$, respectively, and set their values in $\mathbf{a}$ to be true

and false, respectively. Then for any predecessor of $\mathbf{a}$, their values should be false and true, respectively, and for any second predecessor of $\mathbf{a}$, their values should be true and false, respectively. We use variables $y_{i,0}, y_{i,1}, z_{i,0}, z_{i,1}$ as in the case of $\mathcal{B}_{3\mathrm{OR},2\mathrm{AND}}$ and add $u_{i,0}, u_{i,1}$. For each j, we introduce three variables c_j, d_j, and e_j and define their functions to be $\mathrm{OR}(d_j, e_j)$, $\mathrm{OR}(y_{p,c}, y_{q,d})$, and $\mathrm{OR}(y_{p,c}, y_{r,e})$ where $y_{p,c}, y_{q,d}$, and $y_{r,e}$ are the three literals of C_j. We define the functions for $y_{i,b}$ to be $\mathrm{OR}(y_{i,b}, w_1)$ for each $b \in \{0,1\}$, the functions for $u_{i,b}$ to be $\mathrm{id}(z_{i,b})$, the functions for $z_{i,0}$ to be $\mathrm{OR}(y_{i,0}, y_{i,1})$, and the functions for $z_{i,1}$ to be $\mathrm{AND}(y_{i,0}, y_{i,1})$.

In $\mathbf{a}$ we set the value of w_2 and all $u_{i,1}$ to false and set everything else to true. Assume $\mathbf{a}$ has a predecessor $\mathbf{a}'$ and $\mathbf{a}'$ has a predecessor $\mathbf{a}''$. The values of $y_{i,0}$ and $y_{i,1}$ in $\mathbf{a}''$ are OR-ed and AND-ed and stored in $z_{i,0}$ and $z_{i,1}$, respectively, in $\mathbf{a}'$ and then preserved in $u_{i,0}$ and $u_{i,1}$, respectively, in $\mathbf{a}$. So, it must be the case that exactly one of $y_{i,0}$ and $y_{i,1}$ is true in $\mathbf{a}''$ and thus we can view these as truth-assignments to the variables of φ. For each clause C_j, the first and the second literals of C_j as appearing in $\mathbf{a}''$ are OR-ed and stored in $\mathbf{a}'$ as d_j as well as the first and the second literals as e_j. These two are then joined by an OR in $\mathbf{a}$ as c_j. Thus, for $\mathbf{a}''$ to exist the y's in $\mathbf{a}''$ must represent a satisfying assignment of φ. Because y's are OR-ed with w_1 and $\mathbf{a}''$ should have true for w_1, y's in $\mathbf{a}'$ are all true. This means that z's, d's, and e's become all true in $\mathbf{a}$. Thus, $\mathbf{a}$ has a second predecessor if and only if φ is satisfiable.

The hardness for the case $t \geq 3$ follows from Lemma 8. ◀

If $\mathcal{B}$ contains only conjunction or only disjunction, the problems are significantly easy.

▶ **Theorem 11.** *For $\mathcal{B}_{\mathrm{OR}}$ and $\mathcal{B}_{\mathrm{AND}}$, the t-PRED Problem is in AC^0 for all constants $t \geq 1$.*

Proof. Suppose that $\mathcal{B}$ is $\mathcal{B}_{\mathrm{OR}}$, and we are given $\mathcal{F}$ and $\mathbf{a}$. Let $G = (V, E) = G[\mathcal{F}, \mathbf{a}, t]$, and (K, L) be the partition of V as given in Definition 3. By Lemma 4, we have only to, for each $u \in K_0$, enumerate all the paths from u to V_t and then check if the reachable nodes in K_t can be reachable from L_0. Since the number of all the paths in G is $O(n^t)$, this can be carried out using an AC^0 circuit. Thus, we have done. ◀

We now consider Poly-PRED Problems where t is part of the input. By combining Proposition 5 and Theorem 9, we obtain the following corollary for the case where $\mathcal{B}$ contains both conjunction and disjunction.

▶ **Corollary 12.** *Suppose $\mathcal{B}$ is one of $\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}}$, $\mathcal{B}_{2\mathrm{OR},3\mathrm{AND}}$, and $\mathcal{B}_{3\mathrm{OR},2\mathrm{AND}}$. Then the Poly-PRED Problem with template set $\mathcal{B}$ is* NP*-complete.*

In the case of $\mathcal{B}_{\mathrm{id}}$, the problem is L-complete, and in the case of $\mathcal{B}_{\mathrm{OR}}$ and $\mathcal{B}_{\mathrm{AND}}$, the problem is NL-complete. The proofs are omitted due to the page limitation.

▶ **Theorem 13.** *The Poly-PRED Problem with template set $\mathcal{B}_{\mathrm{id}}$ is* L*-complete under logspace-uniform AC^0-reductions.*

▶ **Theorem 14.** *Suppose $\mathcal{B}$ is either $\mathcal{B}_{\mathrm{OR}}$ or $\mathcal{B}_{\mathrm{AND}}$. Then the Poly-PRED Problem with template set $\mathcal{B}$ is* NL*-complete under logspace-uniform AC^0-reductions.*

4 The Complexity of Garden-of-Eden Problems

In this section we prove our results on the Garden of Eden problems. In the previous section we prove our hardness results by producing a many-one reduction from a language L to a synchronous BFDS and $\mathbf{a}$ so that there is a t-th predecessor if the input is a member of L and so that there is a $(t-1)$-st predecessor but no t-th predecessor if the input is not a

member of L. One may think that this construction can be used to produce a many-one reduction from the complement of L to the $(t-1)$-st GOE problem with respect to the same BFDS. Unfortunately, this is not the case because according to the construction there may be multiple first predecessors of $\mathbf{a}$ and so even in the case where the input is a member of L, not every $(t-1)$-st predecessor of $\mathbf{a}$ has a predecessor. Thus, to prove the hardness of a Garden of Eden problem for a language L the construction must be such that if the input is in L, there is a $(t-1)$-st predecessor that is a Garden of Eden and such that if the input is not in L, *every* $(t-1)$-st predecessor has a predecessor.

As mentioned earlier, unlike other $t \geq 1$, 0-GOE Problem is complementary to 1-PRED Problem. Noting that NL is closed under complementation [4, 7], we have the following result from Theorems 9 and 10.

▶ **Theorem 15.** *The* 0*-GOE Problem is in* AC^0 *if the template set is one of* $\mathcal{B}_{\mathrm{id}}$*,* $\mathcal{B}_{2\mathrm{OR}}$*,* $\mathcal{B}_{2\mathrm{AND}}$*,* $\mathcal{B}_{\mathrm{OR}}$*, and* $\mathcal{B}_{\mathrm{AND}}$*,* NL*-complete if the template set is* $\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}}$*, and* coNP*-complete if the template set is either* $\mathcal{B}_{2\mathrm{OR},3\mathrm{AND}}$ *or* $\mathcal{B}_{3\mathrm{OR},2\mathrm{AND}}$*.*

Consider then the case where $t \geq 1$. We first observe an upper bound on the problems for the case of $\mathcal{B}_{\mathrm{OR}}$ and $\mathcal{B}_{\mathrm{AND}}$.

▶ **Lemma 16.** *The Poly-GOE Problem with template* $\mathcal{B}_{\mathrm{OR}}$ *or* $\mathcal{B}_{\mathrm{AND}}$ *is in* NP*.*

Proof. Let $\mathcal{F}$ be a synchronous BFDS with template $\mathcal{B}_{\mathrm{OR}}$ with n variables $x_1, \ldots, x_n$ and let $\mathbf{a}$ be a configuration of $\mathcal{F}$. Suppose we are testing whether $\mathbf{a}$ is a t-th predecessor and is a GOE in the system $\mathcal{F}$ and t is given as 1^t as part of input. Consider a layered digraph $G = (V, E)$ whose variable set V has $t+2$ layers, $X_0, \ldots, X_t, X_{t+1}$, where for each j, $0 \leq j \leq t+1$, X_j consists of n variables $x_{1,j}, \ldots, x_{n,j}$. We draw a directed edge $(x_{i,j}, x_{i',j'})$ if $0 \leq j \leq t$, $j' = j+1$, and $x_{i'}$ is an input to the OR function for x_i.

Let S be the set of all $x_{i,0}$ such that the value of x_i is true in $\mathbf{a}$ and Let S' be the set of all $x_{i,0}$ such that the value of x_i is false in $\mathbf{a}$. Let R be the set of all nodes reachable from S'. Let $T' = R \cap X_t$ and $T = X_t - T'$.

We claim that $\mathbf{a}$ has a t-th predecessor that is a GOE if and only if there exists a set $D \subseteq T$ such that (i) from each node in S there is a path to a node in D that does not visit R and (ii) there is a node u in D such that every node v in X_{t+1} is reached from u is reached from some node in $X_t - D$.

Suppose there is such a D. Define a configuration $\mathbf{b}$ by setting the value of u to true if and only if it belongs to D. Since D is reachable from S without going through the nodes in R and $D \cap R = \emptyset$, we have $\mathcal{F}^t(\mathbf{b}) = \mathbf{a}$ and so $\mathbf{a}$ is a t-th predecessor of $\mathbf{a}$. Each predecessor $\mathbf{b}$ of must have a value true for at least one of the variables v that is reachable from u, but since $\mathbf{b}$ has value false for all the nodes in $X_t - D$, all of them must have value false. These two requirements are conflicting with each other and cannot be satisfied at the same time. Thus, $\mathbf{b}$ is a GOE.

On the other hand, suppose there is no such a D. Suppose there is no D satisfying (i), it means that $\mathbf{a}$ does not have a t-th predecessor, and so, $\mathbf{a}$ does not have a t-th predecessor that is a GOE. Suppose there is a D satisfying (i) but each such D fails to satisfy (ii). For each such D we define $\mathbf{b}$ to be configuration that assigns true to all variables in D and false to the rest. Also, for each such D we can select for each $u \in D$ select one node v in $X_{t+1} - R$ that is reachable from u. By setting the value to true for all v's thus selected to false for the remainder, we obtain a configuration $\mathbf{c}$. Then $\mathbf{b}$ is a t-th predecessor of $\mathbf{a}$ and $\mathbf{c}$ is a predecessor of $\mathbf{b}$ and so every t-th predecessor of $\mathbf{a}$ has a predecessor.

To test the existence of a D in question, we calculate S, S', R, T', T and then try all possible combinations of $D \subseteq T$ and $u \in D$, and so the test can be carried out in NP.

This proves the lemma. ◀

Unlike predecessor problems, Garden-of-Eden problem is NP-complete even if $\mathcal{B}$ is either $\mathcal{B}_{\mathrm{OR}}$ or $\mathcal{B}_{\mathrm{AND}}$ and t is part of the input.

▶ **Theorem 17.** *The* 1*-GOE Problem with template* $\mathcal{B}_{\mathrm{OR}}$ *or* $\mathcal{B}_{\mathrm{AND}}$ *is* NP*-complete.*

Proof. Because we have Lemma 16, we have only to show that the 1-GOE Problem is NP-hard. We will reduce 3SAT to the problem. Let φ be a 3CNF formula of n variables and m clauses. Our synchronous BFDS uses variables, among other, $x_{i,0}, x_{i,1}, y_{i,0}, y_{i,1}, u_i, z_i, 1 \leq i \leq n$, $c_j, 1 \leq j \leq m$. The idea is to use $y_{i,0}$ and $y_{i,1}$ to encode purported value assignments to the negative and the positive literals of x_i. These assignments are not necessarily opposite to each other. We use the variable u_i to generate $y_{i,0}$ as the OR of $x_{i,0}$ and u_i and generate $y_{i,1}$ as the OR of $x_{i,1}$ and u_i. We generate z_i as the OR of $y_{i,0}$ and $y_{i,1}$. We use the value assignments to the literal $y_{i,b}$'s to evaluate the variables c_j's, which are corresponding to the clauses. If φ is satisfiable, it is possible to choose the values for $y_{i,b}$'s so that for each i, $y_{i,0}$ and $y_{i,1}$ are opposite to each other and so u_i is false. If φ is not satisfiable, to make the value of c_j true for all i, we need to assign true to both $y_{i,0}$ and $y_{i,1}$ and so for such an i, u_i can be set to true. So, assuming that z_i's and c_j's are all true, φ is satisfiable if and only if we can choose the values for $y_{i,b}$'s so that u_i's is all false.

We formalize this idea using additional variables p_0, p_1, p_2, u_0, and v_0. The functions for p_0, p_1, and p_2 are respectively $\mathrm{id}(p_2)$, $\mathrm{id}(p_0)$, and $\mathrm{id}(p_1)$; that is, they rotate the values among them, from p_0 to p_1, p_1 to p_2, and then p_2 to p_0. We set the values for them in $\mathbf{a}$ to be false, false, and true, and so, in each predecessor of $\mathbf{a}$, if any, their values should be false, true, false, and in each second predecessor of $\mathbf{a}$, if any, their values should be true, false, false. The function for u_0 is the OR of $u_1, \ldots, u_n$ and the function for v_0 is the OR of u_0 and p_0.

The functions for the other variables are as follows:

- u_i: the OR of u_i and p_0;
- z_i: the OR of $y_{i,0}, y_{i,1}$, and p_0;
- $y_{i,0}$: the OR of $x_{i,0}$ and u_i;
- $y_{i,1}$: the OR of $x_{i,1}$ and u_i;
- $x_{i,0}$: the OR of $x_{i,0}$ and p_0;
- $x_{i,1}$: the OR of $x_{i,1}$ and p_0;
- c_j: the OR of $y_{k,b}$, $y_{l,c}$, $y_{m,d}$, and p_0 where $y_{k,b}$, $y_{l,c}$, $y_{m,d}$ are the variables corresponding to the three literals of the jth clause in φ.

Figure 2 shows how these variables interact.

The configuration $\mathbf{a}$ is all true except for p_0 and p_1. Let us speak of a hypothetical predecessor $\mathbf{b}$ of $\mathbf{a}$ and a hypothetical predecessor $\mathbf{c}$ of $\mathbf{b}$. As mentioned earlier, the value of p_0 is true in $\mathbf{c}$ and false in $\mathbf{b}$. Because of this, all the variables that take p_0 as part of input must be true in $\mathbf{b}$; they are p_1, v_0, all u_i's, all z_i's, all c_j's, and all $x_{i,b}$'s. Since these are all true, in $\mathcal{F}(\mathbf{b})$ the value should be true for u_0, all u_i's, all $y_{i,b}$'s, and $x_{i,b}$'s, which is consistent with their value assignments in $\mathbf{a}$. Also, since v_0 has value true in $\mathbf{a}$, in $\mathbf{b}$ the value of u_0 should be true. The only values that are not determined in $\mathbf{b}$ are those of $y_{i,b}$'s. The constraints are that for each i, either $y_{i,0}$ or $y_{i,1}$ must be true and that these assignments satisfy all the clauses.

Suppose φ is satisfiable. We pick one satisfying assignment of φ and select the values of $y_{i,b}$ accordingly. Then $y_{i,0}$ and $y_{i,1}$ are opposite to each other for all i, and so in $\mathbf{c}$ u_i's must be all false. However, since u_0 has value true in $\mathbf{b}$ and it is the OR of all u_i's, to make it happen the value of at least one u_i must be true in $\mathbf{c}$. These two requirements are conflicting and so $\mathbf{c}$ cannot exist and thus $\mathbf{b}$ is a GOE.

On the other hand, suppose φ is not satisfiable. We determine the values of $y_{i,b}$'s in $\mathbf{b}$ so that they turn c_j's all true in $\mathbf{a}$ and for each i, at least one of $y_{i,0}$ and $y_{i,1}$ is true. Because φ

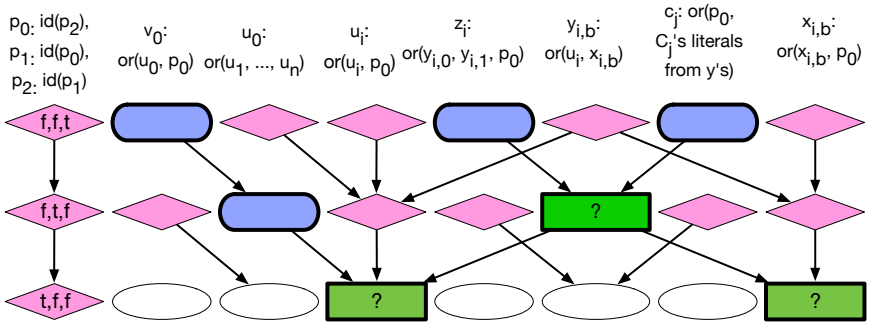


Figure 2 The construction for the 1-GOE Problem for $\mathcal{B}_{\mathrm{OR}}$. The top layer is **a**, the middle layer is a predecessor of **a**, and the bottom layer is a second predecessor of **a**. The arrows show where the inputs come from but inputs from p_i's are not shown. The variables in each diamond shaped are fixed because of the use of p_0. Unless specified, they are all true. The rectangles with rounded sides show blocks of variables whose values are expected to be true. They take input from the rectangles marked with the question mark. They are variables corresponding to the value assignments to the literals and the u_i's.

is not satisfiable, there must be at least one i such that both $y_{i,0}$ and $y_{i,1}$ are true. Select s to be such an i. In **c** set u_s to true and other u_i's to false and set the value of $x_{i,b}$ in **c** equal to the value of $y_{i,b}$ in **b**. Also, set all the remaining variables except the p_i's false. Then this **c** is indeed in a predecessor of **b**. This means that each predecessor **b** of **a** has a predecessor and thus there is no predecessor that is a GOE.

This proves the theorem. ◀

From the above and Lemmas 8 and 16, we have the following corollary.

▶ **Corollary 18.** *Let $\mathcal{B}$ be either $\mathcal{B}_{\mathrm{OR}}$ or $\mathcal{B}_{\mathrm{AND}}$. The Poly-GOE Problem with template $\mathcal{B}$ and t-GOE Problem for $t \geq 1$ with template $\mathcal{B}$ are* NP*-complete.*

If the arities of conjunction and disjunction are bounded by two, the problems are tractable.

▶ **Theorem 19.** *Let $\mathcal{B}$ be one of $\mathcal{B}_{\mathrm{id}}$, $\mathcal{B}_{2\mathrm{OR}}$, and $\mathcal{B}_{2\mathrm{AND}}$. For all constants $t \geq 0$, the t-GOE Problem with template $\mathcal{B}$ is in* AC^0.

Proof. We have only to test the conditions as stated in Lemma 4 part 2. Specifically, we need to compute $G = (V, E)$, the partition (K, L) of V, and then try all possible combinations for u and M to see whether the three properties hold. The conditions can be tested in AC^0 because M has cardinality at most 2. ◀

However, we show that the problem for $\mathcal{B}_{\mathrm{id}}$ and that for $\mathcal{B}_{2\mathrm{OR}}$ or $\mathcal{B}_{2\mathrm{AND}}$ belong to different complexity classes if t is part of the input.

▶ **Theorem 20.** *The Poly-GOE Problem with template $\mathcal{B}_{\mathrm{id}}$ is* L*-complete.*

Proof. Theorem 13 shows that the Poly-PRED Problem with template $\mathcal{B}_{\mathrm{id}}$ is L-complete. By Proposition 7, this implies that Theorem 13 shows that the Poly-GOE Problem with template $\mathcal{B}_{\mathrm{id}}$ is L-hard.

To show that the problem is in L, we use Lemma 4. Since the available function template is id, the set M in the proposition has cardinality 1. As we have seen in Theorem 13, the reachability part of the test can be carried out in L. As for the remaining properties, we have only to try all possible combinations of u and M. ◀

▶ **Theorem 21.** *The Poly-GOE Problem with template $\mathcal{B}_{2\mathrm{OR}}$ or $\mathcal{B}_{2\mathrm{AND}}$ is* NL*-complete.*

Proof. The NL-hardness follows from Theorem 14 and Proposition 7. To show that the problem is in NL, we use Lemma 4. Since the arity of the functions is at most 2, there are only polynomially many possibilities for the choice of z and M and so the test can be done in NL. ◀

In the case where $\mathcal{B}$ contains both conjunction and disjunction, the complexity of problems depends on whether $\mathcal{B}$ contains a function of arity more than two or not for any $t \geq 2$.

▶ **Theorem 22.** *The t-GOE Problem with template $\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}}$ for $t \geq 2$ and the Poly-GOE Problem with template $\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}}$ are* NP*-complete.*

Proof. The hardness follows from Theorem 10, Proposition 7, and Lemma 8. The membership in NP comes from the fact that in the case where the functions have arity 2, whether a configuration has a predecessor can be expressed as a 2CNF formula. We have only to guess a series of t configurations, verify the series flows into $\mathbf{a}$, and the t-th predecessor in the series does not have a predecessor. ◀

▶ **Theorem 23.** *Let $\mathcal{B}$ be either $\mathcal{B}_{2\mathrm{OR},3\mathrm{AND}}$ or $\mathcal{B}_{3\mathrm{OR},2\mathrm{AND}}$. The t-GOE Problem with template $\mathcal{B}$ for $t \geq 1$ and the Poly-GOE Problem with template $\mathcal{B}$ are Σ_2^p-complete.*

We omit the proof of Thorem 23.

5 Conclusions

In this paper we studied the complexity of the Predecessor Existence Problem and the Garden of Eden Problem for various orders of predecessor and for various template sets. Other than those stating containment in AC^0 and the 1-Garden of Eden Problem for $\mathcal{B}_{2\mathrm{OR},2\mathrm{AND}}$, the problems are shown to be complete for standard complexity classes and the classes that appear are diverse: $\mathrm{L}, \mathrm{NL}, \mathrm{NP}, \mathrm{coNP}$, and Σ_2^p. An obvious next step for the paper is to pinpoint the complexity for the remaining case. Also, it will be interesting to look at other template sets.

References

1 C. L. Barrett, H. S. Mortveit, C. M. Reidys. Elements of a theory of simulation II: Sequential dynamical systems. *Applied Mathematics and Computation*, 107(2-3):121–136, 2000.

2 C. L. Barrett, H. B.Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns and P. T. Tosic. Gardens of Eden and Fixed Points in Sequential Dynamical Systems. In *Proceedings of Discrete Mathematics and Theoretical Computer Science*, 95–110, 2001.

3 C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. Complexity of reachability problems for finite discrete dynamical systems. *Journal of Computer and System Sciences*, 340(3):496–513, 2005.

4 N. Immerman. Nondeterministic space is closed under complementation. SIAM Journal on Computing, 17:935–938, 1988.

5 S. Kosub. Dichotomy results for fixed-point existence problems for boolean dynamical systems. *Mathematics in Computer Science*, 1(3):487–505, 2008.

6 S. Kosub and C. M. Homan. Dichotomy results for fixed point counting in boolean dynamical systems. In *Proceedings of the Tenth Italian Conference on Theoretical Computer Science*, pages 163–174, 2007.

7 R. Szelepcsényi. The method of forcing for nondeterministic automata. Bulletin of the EATCS, 33:96–100, 1987.

8 M. Ogihara and K. Uchizawa. Computational complexity studies of synchronous boolean finite dynamical systems. In *Proceedings of the 12th Conference on Theory and Applications of Models of Computation*, pages, 87–98, 2015.

Dividing Splittable Goods Evenly and With Limited Fragmentation

Peter Damaschke

Department of Computer Science and Engineering, Chalmers University, Göteborg, Sweden
ptr@chalmers.se

Abstract

A splittable good provided in n pieces shall be divided as evenly as possible among m agents, where every agent can take shares of at most F pieces. We call F the fragmentation. For $F = 1$ we can solve the max-min and min-max problems in linear time. The case $F = 2$ has neat formulations and structural characterizations in terms of weighted graphs. Here we focus on perfectly balanced solutions. While the problem is strongly NP-hard in general, it can be solved in linear time if $m \geq n - 1$, and a solution always exists in this case. Moreover, case $F = 2$ is fixed-parameter tractable in the parameter $2m - n$. The results also give rise to various open problems.

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

Keywords and phrases packing, load balancing, weighted graph, linear-time algorithm, parameterized algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.9

1 Introduction

Suppose that we are given n pieces of a good, and these pieces have sizes $x_1, \ldots, x_n$. The good shall be divided among m agents, thereby respecting certain criteria and restrictions. In a solution, let $y_1, \ldots, y_m$ denote the amounts that the agents receive, and let z_{ij} be the amount that agent j receives from piece i. Clearly, we have $y_j = \sum_{i=1}^{n} z_{ij}$ for all agents j, and $\sum_{j=1}^{m} z_{ij} \leq x_i$ for all pieces i. All mentioned variables are non-negative. The agents are identical, and so are the pieces of the good (apart from their different sizes).

Ideally we would like to divide the good evenly and completely, that is: $y_1 = \ldots = y_m$ and $\sum_{i=1}^{n} x_i = \sum_{j=1}^{m} y_j$. Without further restrictions it is a trivial problem to find mn numbers z_{ij} that satisfy these demands. But suppose that we also want to limit the fragmentation, in the following sense. Let F be some fixed positive integer. Every agent shall get parts of at most F distinct pieces. Formally, for every j we allow $z_{ij} > 0$ for at most F indices i. (These indices can be chosen by the solution, but their number is limited by F.) However, every piece may be divided among an unlimited number of agents.

One possible motivation is that pieces of land, at n different locations and with areas $x_1, \ldots, x_n$, shall be assigned to farmers in a fair way. Besides fairness, it would be desirable for every single farmer to get only a few different fields, rather than several scattered ones, such that the farmer does not have to divide activities between many different locations. One may also think of applications in scheduling, where the x_i are durations of n jobs that shall be divided among m workers, in such a way that they get equal workloads, and every worker is concerned with only a few different jobs, in order to limit context switching. Of

course, in such a scenario we have to assume that the jobs can be arbitrarily split and also parallelized. An example where these assumptions are realistic is grading of the n exercises of an exam by m graders.

Prominent problems in Discrete Optimization, in various application domains, deal with cutting or connecting items from raw materials, e.g., the Cutting Stock and Skiving Stock problems; see [6]. An action equivalent to cutting is packing items into containers, as in the Bin Packing and Knapsack problems [5]. (In fact, Skiving Stock is also known as Dual Bin Packing). However, the problems we consider here differ from all the mentioned problems in one way or another, and to our best knowledge their complexities have not been studied before. For instance, in the "Stock" problems we are given large amounts of items of only a few different sizes, and in the Bin Packing and Knapsack problems the sizes of items to be packed (or cut out) are prescribed.

The paper is organized as follows. In Section 2 we define splitting problems where the good shall be divided completely and the minimum share shall be maximized, or vice versa. We call a solution perfect if all shares are equal. In Section 3 we solve the case of fragmentation $F = 1$ in linear time. However, we first derive weaker $O(n \log n)$ time bounds, because this makes the presentation easier, and the $O(n)$ solution relies on the (barely practical) linear-time selection but should still be of theoretical interest. In Section 4 we are concerned with the case of fragmentation $F = 2$. Here we focus on structural properties of the problem of getting perfect solutions, but this may also serve as a basis for, e.g., approximation algorithms for the more general optimization versions. We describe solutions in a natural way by weighted graphs. Then we present a linear-time elimination algorithm that finds a tree-like perfect solution whenever $m \geq n - 1$. Based on the graph-theoretic structure we also prove strong NP-completeness of the general problem, and NP-completeness already for the special case when m is slightly smaller than $n - 1$. For m being close to the smallest possible value $n/2$ we show fixed-parameter tractability. In Section 5 we point out possible directions of further research.

2 Preliminaries

For clarity we provide formal definitions of the problems. We always silently presume that F is a fixed positive integer called the *fragmentation*. Common to all problems are the input and some of the constraints:

Splitting:
Given are n positive numbers $x_1, \ldots, x_n$.
Find non-negative numbers z_{ij} and y_j (for $i = 1, \ldots, n$ and $j = 1, \ldots, m$) subject to:
$\forall j: \ y_j = \sum_{i=1}^{n} z_{ij}$,
$\forall i: \ \sum_{j=1}^{m} z_{ij} \leq x_i$,
$\forall j: \ |\{i \mid z_{ij} > 0\}| \leq F$,
and further constraints and objectives specified below.

To fully specify the actual problems we only mention these additional constraints, in order to avoid repetitions.

Perfect Splitting: $\forall i: \ \sum_{j=1}^{m} z_{ij} = x_i$, and all shares y_j are equal.

We refer to a solution of Perfect Splitting as a *perfect solution*. In such a solution, all agents get the same amount, and no goods are held back.

Min-Max Splitting: $\forall i: \ \sum_{j=1}^{m} z_{ij} = x_i$, and the largest share, $\max_j y_j$, is minimized.

Max-Min Splitting: $\forall i: \ \sum_{j=1}^{m} z_{ij} = x_i$, and the smallest share, $\min_j y_j$, is maximized.

In the last two problems, still all goods are distributed, but in general the agents get different shares. If a perfect solution exists, then this is also the optimal solution to both Min-Max and Max-Min Splitting. The Min-Max problem is more appropriate for applications where some work must be divided completely, and the goal is not to load any individual agents too much. The Max-Min problem aims at giving everyone a guaranteed amount of a good, as large as possible. However, a solution may be perceived as unfair, in the sense that other agents get significantly more because the entire good must be divided. Also, the size of the smallest piece is a trivial upper bound on the objective value. To circumvent these issues one may define modified Max-Min problems which aim at giving maximal but equal amounts to all agents, possibly leaving the remainder of goods unused. One could also relax the condition on fragmentation F and allow some outliers. However, we cannot study all variants in this paper, and we focus on the basic problems.

3 The Case Without Fragmentation (F=1)

First we study Min-Max Splitting and Max-Min Splitting, which also subsumes the special case of Perfect Splitting, for fragmentation $F = 1$.

As a trivial observation, either of the first two problems always permits an optimal solution where, for every piece i, all $z_{ij} > 0$ are equal. Otherwise we can balance these values without making the solution worse. Thus, such a solution is fully characterized by a vector $(p_1, \ldots, p_n)$ with $\sum_{i=1}^{n} p_i = m$, where p_i is the number of indices j with $z_{ij} > 0$. For every such i, j we obviously have $y_j = z_{ij} = x_i/p_i$. Due to this observation on the possible y_j-values we define the following sequence Y.

▶ **Definition 1.** Y is the sequence of all numbers x_i/p, where $i = 1, \ldots, n$ and $p = 1, \ldots, m$, sorted in decreasing order. For $k = 1, \ldots, mn$ let $Y[k]$ denote the value at position k in Y.

The same value x_i/p may come from different i and p, therefore a value may occur multiple times in Y. However, we can break ties arbitrarily and sort equal values in any order. Formally this can also be achieved by random and infinitesimal perturbations of the values x_i. We will henceforth assume that all values in Y are distinct, hence Y is strictly monotone decreasing. This avoids circumstantial treatment of specific cases.

Intuitively, the p_i should be roughly proportional to the given x_i. But the efficient computation of exact optimal solutions is a bit less obvious. We discuss this matter in the following subsections.

3.1 Maximizing the minimum

The following lemma deals with the simple problem being inverse to Max-Min Splitting.

▶ **Lemma 2.** *For any fixed y with $0 < y \leq \min_i x_i$, let k be the maximum number of agents*[1] *such that every agent can obtain an amount at least y. This number k satisfies:*

(a) $k = \sum_{i=1}^{n} p_i$, *where* $p_i := \lfloor x_i/y \rfloor > 0$.
(b) k *is the maximum index with* $Y[k] \geq y$.
(c) $Y[k] = \min_i x_i/p_i$.

Proof. (a): For every i we can split the ith piece among at most $\lfloor x_i/y \rfloor$ agents. Summation over all i yields the assertion. Due to the assumption on y we have $p_i > 0$ for all i.

[1] As y is fixed, for notational convenience we do not mention the argument y and call this number only k.

(b): Note that $x_i/s_i \geq y$ holds for all i and all $s_i \leq p_i$, and there exist k such pairs (i, s_i). Hence at least k members of Y are greater than or equal to y, that is, $Y[k] \geq y$. Since $x_i/(p_i + 1) < y$ holds for all i, we also see that no further members of Y have this property.

(c): As seen above, the k values x_i/s_i ($s_i \leq p_i$) are exactly those members of Y being greater than or equal to y, that is, they are the first k items in Y, and clearly $\min_i x_i/p_i$ is the last of them. ◀

Lemma 2 yields a characterization of the optimal solution.

▶ **Lemma 3.** *The optimal objective value for* Max-Min Splitting *with* $F = 1$ *is*

$$\max \min_j y_j = \min\{Y[m], \min_i x_i\},$$

where the maximum is taken over all solutions.

Proof. First suppose that $Y[m] < \min_i x_i$.

Then we can apply Lemma 2 to $y := Y[m]$. The maximum number k of agents that can be served is, due to (b), equal to the maximum index k with $Y[k] \geq Y[m]$. This implies $k = m$. In other words, m agents can obtain an amount of at least $Y[m]$ each.

Assume that m agents can obtain more, say an amount of $y' > y$ each, where still $y' < \min_i x_i$. We apply Lemma 2 to y'. The maximum number k' of agents that can be served is, due to (b), equal to the maximum index k' with $Y[k'] \geq y' > y = Y[m]$. This implies $k' < m$. Hence we have shown by contradiction that m agents cannot obtain any amount larger than y.

The other case to consider is $Y[m] \geq \min_i x_i$. Now we can apply Lemma 2 to $\min_i x_i$. Part (c) implies that $Y[k] = \min_i x_i/p_i \leq \min_i x_i \leq Y[m]$, thus $k \geq m$. That is, m agents can be served with an amount at least $\min_i x_i$. But due to the problem specification, $\min_i x_i$ is also a trivial upper bound on the objective value, which finally proves the assertion also in this case. ◀

The previous lemmas yield already an algorithm for Max-Min Splitting: First, it is trivial to figure out whether m agents can get a share of at least $\min_i x_i$ each. If so, then this solution is optimal, and we are done. If not, then $Y[m] < \min_i x_i$. In this case, find the value $y := Y[m]$ and then determine the p_i as in Lemma 2 (a), using this y.

However, in order to save time we want to get the value $Y[m]$ without naively following Definition 1 and generating all $m - 1$ preceding elements of Y. The intuition for a faster approach is to search the optimal value near the average. In the following we can always suppose $Y[m] < \min_i x_i$.

▶ **Lemma 4.** *Let* $\bar{y} := \sum_{i=1}^{n} x_i/m$ *be the average amount given to the* m *agents, and let* k *be the maximum number of agents such that every agent can actually obtain an amount at least* $\bar{y}$*. Then* $Y[k] \geq Y[m]$ *and* $m - k \leq n$*.*

Proof. Clearly, $\bar{y} \geq \min_j y_j$ holds in any solution, hence $\bar{y} \geq \max \min_j y_j$ (where the maximum is taken over all solutions). Now Lemma 3 yields $\bar{y} \geq Y[m]$, and Lemma 2 applied to $\bar{y}$ gives $Y[k] \geq \bar{y} \geq Y[m]$.

For $i = 1, \ldots, n$ we define $q_i := x_i/\bar{y}$, with integer and fractional part $p_i := \lfloor q_i \rfloor$ and $r_i := q_i - \lfloor q_i \rfloor$, respectively. Lemma 2 (a) states that $k = \sum_{i=1}^{n} p_i$. Thus we observe:

$$k = \sum_{i=1}^{n} p_i = \sum_{i=1}^{n} (q_i - r_i) \geq \sum_{i=1}^{n} q_i - n = m - n.$$

From this chain of inequalities we get $m - k \leq n$. ◀

Based on these inequalities we will now give an efficient algorithm for Max-Min Splitting. We adopt the unit cost measure where comparisons and algebraic operations with real numbers take constant time.

▶ **Theorem 5.** Max-Min Splitting *with* $F = 1$ *can be solved in* $O(n \log n)$ *time.*

Proof. First we compute $\bar{y}$ and all p_i (as defined in Lemma 2 with $y := \bar{y}$ and x_i/p_i), as well as k and $Y[k] = \min_i x_i/p_i$. The latter equation holds due to Lemma 2 (c). These calculations cost $O(n)$ time.

By Lemma 3, the optimal value is $Y[m]$. In the following we determine the value $Y[m]$. Note that k and $Y[k]$ are known from the calculations above, and $Y[k] \geq Y[m]$ and $m-k \leq n$ hold due to Lemma 4.

If $m \leq k$, then $Y[m] = Y[k]$, and we are done with this part. If $m > k$, then we only have to find the next $m-k$ members of Y after $Y[k]$, and since $m-k \leq n$, these are at most n further members. Now we describe a possible way to identify $Y[k], \ldots, Y[m]$.

Let us sort the n ratios x_i/p_i in decreasing order and call this sequence R. Since $Y[k] = \min_i x_i/p_i$, the last element of R is exactly $Y[k]$. We call it the *marked* element, for later reference. In R we store not only the values x_i/p_i but also x_i and p_i separately. We move a pointer in R from left to right. For every ratio encountered in R we compute the ratio with incremented denominator ($p_i := p_i + 1$) and insert it at the correct position in R. Note that this position is always to the right of the pointer; in particular, the new ratio may become the new rightmost element of R. This step is repeated until the pointer reaches a position $m-k$ elements to the right of the marked element, and then we stop.

The analysis is simple: Since new ratios are always inserted to the right of the pointer, and R comprises *all* elements of Y between the marked $Y[k]$ and the pointer, it follows that we are at $Y[m]$ when we stop, and we can output its value. In order to support fast insertion of a new ratio into R, we also host R in a balanced search tree. Therefore this procedure can be done in $O(n \log n)$ time.

For every i we recompute p_i by $p_i := \lfloor x_i/Y[m] \rfloor$, and the amount given to the p_i agents assigned to the ith piece is x_i/p_i. ◀

Some comments on the time bound are in order. Theorem 5 actually says that we need $O(n \log n)$ time to compute the numbers of agents assigned to each piece and their shares, in an optimal solution. Under the unit cost measure, this time bound is independent of m which may be arbitrarily larger than n. The "physical" division, i.e., assigning positive values to m variables z_{ij}, takes $O(m)$ additional time in a trivial postprocessing phase.

Modifications of the algorithm can also solve a variant of Max-Min Splitting with $F = 1$ where not all goods need to be distributed. Such an algorithm would not get to the pieces being smaller than the objective value.

3.2 Minimizing the maximum

In order to minimize $\max_j y_j$ we use the sequence Y from Definition 1 as well. For formal reasons we also set $Y[0] := \infty$. The scheme is pretty much the same as for Max-Min Splitting, but as the two problems are not symmetric, care must be taken for several details that are different. Similarly as before, we start from the simple problem being inverse to Min-Max Splitting.

▶ **Lemma 6.** *For any fixed* $y > 0$ *that does not appear in* Y*, let* $k \geq n$ *be the minimum number of agents needed such that every agent has to take an amount at most* y*. This number* k *satisfies:*

(a) $k = \sum_{i=1}^{n} p_i$, *where* $p_i := \lceil x_i/y \rceil$.
(b) k *is the maximum index with* $Y[k-n] \geq y$.
(c) $Y[k-n] = \min_i x_i/(p_i - 1)$.

Proof. (a): For every i we must split the ith piece among at least $\lfloor x_i/y \rfloor$ agents. Summation over all i yields the assertion.

(b): Note that $x_i/s_i \geq y$ holds for all i and all $s_i \leq p_i - 1$, and there exist $k - n$ such pairs (i, s_i). Hence at least $k - n$ members of Y are greater than or equal to y, that is, $Y[k-n] \geq y$. Since $x_i/p_i < y$ holds for all i, we also see that no further members of Y have this property.

(c): As seen above, the $k - n$ values x_i/s_i ($s_i \leq p_i - 1$) are exactly those members of Y being greater than or equal to y, that is, they are the first $k - n$ items in Y, and clearly $\min_i x_i/(p_i - 1)$ is the last of them. ◀

Again we get a simple characterization of the optimal solution.

▶ **Lemma 7.** *The optimal objective value for* Min-Max Splitting *with* $F = 1$ *is*

$$\min \max_j y_j = Y[m - n + 1],$$

where the maximum is taken over all solutions.

Proof. We apply Lemma 6 to $y := Y[m-n+1]+\delta$ with an infinitesimal $\delta > 0$, added in order to meet the requirement that y itself does not occur in Y. The minimum number k of agents needed is, due to (b), equal to the maximum index k with $Y[k-n] > Y[m-n+1]$. This implies $k = m$. In other words, m agents have to take an amount of at most $Y[m-n+1]+\delta$ each. Since δ can be made arbitrarily small, their maximum load is bounded by $Y[m-n+1]$.

Assume that the m agents have to take even less, say $y' < Y[m - n + 1]$. We apply Lemma 6 to y'. The minimum number k' of agents needed is, due to (b), equal to the maximum index k' with $Y[k' - n] \geq y'$. Since, in particular, $Y[m + 1 - n] \geq y'$, this implies $k' \geq m + 1$. This shows that more than m agents are needed to bound their maximum load by any amount smaller than $Y[m - n + 1]$. ◀

The proof of Theorem 5 showed already that we can determine $Y[m]$ from $\bar{y}$ in $O(n \log n)$ time. By a similar procedure we can also determine $Y[m - n + 1]$, which is optimal for Min-Max Splitting due to Lemma 7. (Note that n is known from the instance.) Again this costs only $O(n \log n)$ time, and the same remarks as earlier apply to the actual construction of the solution. We can readily state:

▶ **Theorem 8.** Min-Max Splitting *with* $F = 1$ *can be solved in* $O(n \log n)$ *time.*

3.3 Splitting in linear time

An obvious question is whether $O(n \log n)$ time is actually needed. Re-inspecting the proof of Theorem 5) we see: The non-trivial case is $r := m - k > 0$ (but $r \leq n$), and there we only need to find the rth largest ratio after the marked element $Y[k]$. Thus it may not be necessary to keep our sequence R of ratios sorted. But a difficulty is that the sought element is not simply the rth largest $x_i/(p_i + 1)$, where the p_i denote the initial values obtained from $\bar{y}$. Rather, several ratios x_i/p with the same index i but varying p may be larger than the sought element. It is not immediately clear in which order we should generate new ratios and do comparisons.

But, in fact, we can achieve $O(n)$ time. We present this improvement here separately, as it does not look very practical (see below). Yet the optimal time bound may be of interest. First we need a separation property of our sequence of ratios:

▶ **Lemma 9.** *For positive numbers x, x', p, q it is impossible that $x > x'$ and*

$$\frac{x}{q} > \frac{x'}{p} > \frac{x'}{p+1} > \frac{x}{q+1}.$$

Proof. Clearly, the ratio of the two outer numbers is larger than the ratio of the two inner numbers:

$$\frac{x}{q} \cdot \frac{q+1}{x} > \frac{x'}{p} \cdot \frac{p+1}{x'}.$$

It follows $q < p$. But this implies $(q+1)x' < (p+1)x$, which contradicts the last inequality in the given chain. ◀

Now we outline a linear-time algorithm for Max-Min Splitting, improving upon the implementation details in Theorem 5. First, a few preparations and definitions are needed. We find the maximum x_i in $O(n)$ time. Without loss of generality let $x_1 = \max_i x_i$. Considering the sequence of ratios x_1/q with integers q, to the right of the marked element $Y[k]$, we call every interval $(x_1/q, x_1/(q+1)]$ a *bin*. Note that every such interval includes its right endpoint. For a set S of ratios, the *generation step* is the following procedure. For every ratio x_i/p_i in S, we increment the denominator by 1 and compute the number q of the bin that contains $x_i/(p_i+1)$. Using divisions this requires only $O(1)$ time per element. The Selection problem is to find the tth smallest number in an unsorted set, for a prescribed number t.

First we apply the generation step to all initial values x_i/p_i (that is, with the p_i obtained from $\bar{y}$, as earlier). Then we scan the bins from left to right, i.e., for increasing q. For every q we take all ratios that are currently in the qth bin and apply the generation step to them. At the same time we count the total number of ratios seen so far, including the x_1/q. As soon as this number reaches r, we know that the sought ratio is in the current bin. Lemma 9 ensures that every bin contains at most one ratio x_i/p (p integer) for every index i, thus $O(n)$ ratios altogether. Let $t := r - s$, where s is the number of ratios in all previous bins. Hence we can finally apply an $O(n)$ time Selection algorithm [1] to the current bin, in order to find the sought ratio at the tth position in this bin. For the closely related Min-Max Splitting problem we can proceed similarly. This shows:

▶ **Theorem 10.** Max-Min Splitting *and* Min-Max Splitting *with $F = 1$ can be solved in $O(n)$ time.*

A caveat is that $O(n)$-time Selection algorithms suffer from a large hidden constant in the time bound. Instead of a deterministic algorithm we may use the randomized Quickselect, but then we only get $O(n)$ expected time. One may wonder if $O(n)$ worst-case time can be accomplished without invoking Selection. However this is not possible. For instance, if all x_i are close to each other and n does not divide m, then finding $Y[m]$ is equivalent to finding the (m mod n)th largest x_i.

Figuratively speaking, our splitting problems are "Selection-complete" under "simple" linear-time reductions. This statement could be formalized in an algebraic model of computation that cares about constant factors (e.g., by counting the exact depth in a decision tree model). In a more general perspective it may be interesting to – in this way – strictly classify problems that are all linear-time solvable but of different complexity when it comes to constant factors.

4 Perfect Solutions for Fragmentation F=2

In the following we study the case $F = 2$. It is natural to represent any instance of a splitting problem, along with its solution, as a weighted graph:

▶ **Definition 11.** The *solution graph* of a solution to a splitting problem with $F = 2$ is a graph with n vertices and m edges, specified as follows. We create a vertex of weight x_i for the ith piece. Every edge uv has two *ports* at the vertices u and v. The solution specifies a set of m edges and $2m$ weights of their ports. Specifically, if the jth edge has a port at the ith vertex, then the weight of this port is z_{ij}. Every y_j is the sum of the weights of the two ports of the jth edge. We consider y_j as the weight of the jth edge. Similarly, the weights of all ports at the ith vertex must sum up to x_i. An edge can also be a loop at one vertex, and in this case it is immaterial how its weight is divided into weights of the two ports.

In this section we want to characterize which instances, given by n positive vertex weights $x_1, \dots, x_n$ and an integer m, allow a perfect solution (see Section 2). Without loss of generality we can assume $\sum_{i=1}^{n} x_i = m$, hence a perfect solution must satisfy $y_j = 1$ for all j. That is, all edge weights are 1.

4.1 Many agents make it easy

▶ **Theorem 12.** *Every instance of* Perfect Splitting *with* $F = 2$ *and* $m \geq n - 1$ *has a solution whose solution graph is a tree, possibly with loops attached to some vertices. Moreover, such a solution can be computed in* $O(n)$ *time.*

Proof. We classify the vertices in several categories depending on their weights x_i. Vertex i is called large if $x_i > 1$, normal if $x_i = 1$, medium if $1/2 < x_i < 1$, and small if $0 < x_i \leq 1/2$.

Our strategy works as follows. We create certain edges of weight 1, reduce the remaining weights of the incident vertices accordingly (and obeying some simple rules), and recursively solve the residual instances. We only need to show that the process terminates only when the vertex weights are zero, and it can be implemented in $O(n)$ time. In detail:

First suppose that $m > n$. Then there exists a large vertex i. Hence we can attach a loop to it and update $x_i := x_i - 1$ and $m := m - 1$, while n is unchanged. By an inductive argument, we can attach $m - n$ loops to large vertices, until $m = n$ is reached. The number of loops attached to every large vertex can be decided, e.g., in a greedy fashion: Choose any large vertex i and create $\lceil x_i \rceil - 1$ loops, take another large vertex, and so on, but stop as soon as the total number of loops reaches $m - n$. The computations obviously require only $O(n)$ arithmetic operations.

Next suppose that $m = n$. If all vertices are normal, then we append another loop to each vertex, and we are done. If not all vertices are normal, then there still exists some large vertex, and we attach another loop to it, as above. Thus we reach $m = n - 1$.

From now on suppose that $m = n - 1$. Assume that we find two vertices i and j such that $x_i + x_j > 1$ and $x_j < 1$. Then we join these vertices by an edge of weight 1, which is divided as follows. The port at vertex j gets the weight x_j, and the port at vertex 1 gets the rest $1 - x_j$. Accordingly we update the vertex weights to $x_j := 0$ and $x_i := x_i - 1 + x_j > 0$. This means, there remain $n := n - 1$ vertices with positive weights, and $m := m - 1$ edges to fix. Hence the invariant $m = n - 1$ is preserved in the residual instance, and we can iterate this procedure as long as possible.

It remains to prove the existence of such a pair of vertices i and j satisfying $x_i + x_j > 1$ and $x_j < 1$, and to find some efficiently. Since $m < n$, at least one medium or small vertex

exists. As long as some large or normal vertex exists, too, we obviously get a pair as desired. Now suppose that all vertices are medium or small. We can also take a pair of medium vertices, if it exists. Thus suppose in the sequel that all vertices are small, except at most one medium vertex. Now the equation $m = n - 1$ restricts the possibilities as follows. If two small vertices exist, then these are the only vertices: We have $n = 2$ and $x_1 = x_2 = 1/2$, hence we can join the two vertices by a final edge. If only one vertex is small, then there exists exactly one other vertex which is medium. Hence we still have $n = 2$ and $x_1 + x_2 = 1$, and we can insert a final edge. The case that no vertex is small is impossible, since then $n = 1$ and $m = x_1 > 0$, a contradiction. Altogether this shows that we always find two desired vertices until $n = 2$ is reached, and then we can finish up the solution.

The above proof does not only show the existence of a perfect solution whenever $m \geq n-1$, but it describes already a simple elimination algorithm that computes a perfect solution. In each step, the updates take $O(1)$ time. The next edge is always built from two vertices from certain categories (large, normal, medium, small). Since arbitrary vertices from the respective categories can be chosen, it suffices to maintain four unsorted sets of vertices, hence we can also update these sets and pick the needed vertices in $O(1)$ time. Therefore the overall time is $O(n)$.

Consider the graph of non-loop edges inserted by the algorithm until any moment. Upon insertion of every edge, the weights of its vertices were positive, and the weight of exactly one of them drops to zero. This implies the invariant that every connected component of the graph retains exactly one vertex with positive weight. From this it follows, furthermore, that every new edge merges two connected components. Thus the final solution graph is a tree, possibly with additional loops. ◀

Similarly to the time bounds for $F = 1$, we have not counted in the time for the trivial postprocessing that actually splits the pieces: Constructing the $O(m)$ loops costs $O(m)$ additional time. But the solution graph, i.e., the tree and the numbers of loops at its vertices, are computed in $O(n)$ time.

It is apparent from the algorithm that the tree, and thus the solution, is in general not unique. This gives rise to interesting additional problems, also in view of the motivations in Section 1. For instance, assuming that the vertices have pairwise distances (spatial distances, dissimilarity of tasks, etc.) we may prefer perfect solutions where also some distance measure (maximum, total, weighted, etc.) is minimized for the chosen edges.

4.2 Structural characterization and hardness

Apart from the last remark, Theorem 12 completely settles the case $m \geq n - 1$. In the following we also allow $m < n$ (but $m \geq n/2$, since otherwise not all goods can be divided with fragmentation $F = 2$). The conditions in Theorem 12 suggest the following definition.

▶ **Definition 13.** Let V be a set of elements called vertices and indexed by $1, \ldots, n$. Every vertex i has a weight positive weight x_i. We call $I \subseteq V$ an *integral set* if $\sum_{i \in I} x_i$ is an integer. We call $I \subseteq V$ a *heavy set* if $\sum_{i \in I} x_i \geq |I| - 1$.

In fact, the existence of perfect solutions can now be characterized as follows.

▶ **Theorem 14.** *An instance $(x_1, \ldots, x_n)$ of* Perfect Splitting *with $F = 2$, where $\sum_{i=1}^{n} x_i = m$ (and m is the number of agents), admits a solution if and only if V can be partitioned into heavy integral sets.*

Proof. "only if": Suppose that there exists a perfect solution. Since $F = 2$, the solution can be represented as a solution graph G as in Definition 11, with vertex set V and with m edges (some of which may be loops). Let $C(k)$ denote the kth connected component of G, where the indexing is arbitrary. Let n_k and m_k denote the number of vertices and edges, respectively, in $C(k)$. Since $\sum_{i=1}^{n} x_i = m$, every agent gets an amount of 1. Hence, for every k, the vertex set V_k of C_k is integral. Specifically, the sum of vertex weights in $C(k)$ equals m_k which is trivially an integer. Due to connectivity we also have $m_k \geq n_k - 1$, thus V_k is a heavy set.

"if": Suppose that V has a partitioning into integral sets V_k which are also heavy. The latter means that $m_k \geq n_k - 1$, where n_k is the number of vertices of V_k, and m_k denotes their total weight. For every V_k we consider an instance of Perfect Splitting with the given vertex weights and m_k agents. Due to $m_k \geq n_k - 1$ and Theorem 12, this instance has a solution with m_k agents. Since $m = \sum_k m_k$, the solutions to all these k instances together form a solution of the entire instance. ◀

While Perfect Splitting with $F = 2$ is easy for $m \geq n - 1$ due to Theorem 12, the complexity jumps for $m < n - 1$. The reason is that, unfortunately, it is hard to find a partitioning as required in Theorem 14, as we show next. At first glance hardness might appear counterintuitive because with fragmentation $F = 2$ it should always be possible, within an elimination process as in Theorem 12, to take amounts missing to some $z_{ij} = 1$ from some other piece. But the catch is that all remaining pieces might be too small, and then the fragmentation $F = 2$ is not sufficient. Anyway, by a reduction from 3-Partition (which is a natural candidate that has been reduced earlier to similar packing and scheduling problems [2]) we can show:

▶ **Theorem 15.** Perfect Splitting *with* $F = 2$ *is strongly NP-complete, and so are* Max-Min Splitting *and* Min-Max Splitting.

Proof. We give a polynomial-time reduction from the strongly NP-complete 3-Partition problem to Perfect Splitting. A triplet is a set of exactly three elements. An instance P of 3-Partition consists of $3k$ positive rational numbers that shall be partitioned into k triplets such that the sum of the numbers in each triplet is the same. The instance is a multiset, i.e., the $3k$ numbers are not necessarily distinct. Without loss of generality let their sum be k, hence the sum in each triplet must be 1. Thus we can further assume $x \leq 1$ for all x in P, otherwise P has, trivially, no solution. We also fix some small number $d > 0$, in fact, any number with $0 < d < 1/3$ will do.

For the reduction we take any given instance P with the above properties, and we transform every number x from P into $2(1/3 + dx)/(1 + d)$. Let Q be the multiset of these transformed numbers. They enjoy the following properties:
(a) Any three numbers from P sum up to 1 if and only if the three transformed numbers in Q sum up to 2.
(b) The sum of all numbers in Q is $2(3k/3 + dk)/(1 + d) = 2k$.
Let $n := 3k$ and $m := 2k$. Now we can view Q as an instance of Perfect Splitting with $F = 2$, where n is the number of pieces, and $m = 2k$ is both the number of agents and the total amount to distribute.

Assume that P has a solution. Then each of its k triplets has the sum 1. Due to (a), the three transformed numbers in Q have the sum 2. Hence the triplets form a partitioning of Q into heavy integral sets. By Theorem 14, Q has a perfect solution.

Conversely, suppose that Q has a perfect solution. Using Theorem 14 again, Q can be partitioned into heavy integral sets. Since $n - m = k$ and the sets are heavy, the partitioning

must consist of at least k sets. Remember that $x \leq 1$ for all x from P. Hence any single number in Q is at most $2(1/3+d)/(1+d) < 1$. Any two numbers in Q have a sum at least $(4/3)/(1+d) > 1$ and at most $4(1/3+d)/(1+d) < 2$. Hence any integral set needs at least three elements. It follows that Q is partitioned into exactly k triplets. Using again that these sets are heavy, the sum in each triple is at least 2. Since, due to (b), the total sum equals $2k$, the sum in each triplet is exactly 2. Using (a) again, it follows that each corresponding triplet in P has the sum 1. That means, P has a solution.

Since Perfect Splitting is a special case of the two optimization problems, the last assertion follows immediately. ◀

Usual NP-hardness holds already when m is slightly smaller than $n-1$ (giving a clear dichotomy together with Theorem 12), and the proof is less technical:

▶ **Theorem 16.** Perfect Splitting *with $F = 2$ and $m = n - t$ is NP-complete for every fixed $t \geq 2$, and so are* Max-Min Splitting *and* Min-Max Splitting.

Proof. First let $t = 2$. Consider any instance where $m = n - 2$, and $x_i < 1$ for all i. Any partitioning into heavy integral sets necessarily consists of exactly two sets I and J, with $\sum_{i\in I} x_i = |I| - 1$ and $\sum_{j\in J} x_j = |J| - 1$. Due to Theorem 14, such an instance has a solution if and only if it can be partitioned into sets I and J with these properties.

On this basis we give a reduction from the NP-complete Subset Sum problem [2]. An instance of Subset Sum consists of positive rational numbers $y_1 \ldots, y_n$, and the goal is to find a subset I of indices such that $\sum_{i\in I} y_i$ equals a prescribed value s. Subset Sum is NP-complete already in the case that $s = \sum_{k=1}^{n} y_k/2$. By scaling we can also assume $\sum_{k=1}^{n} y_k = 2$, such that we have to divide the sum into $1+1$. Now we can also assume $y_k < 1$ for all k, otherwise the instance has, trivially, no solution. Finally, we simply set $x_k := 1 - y_k$ for all k. The equivalence to Perfect Splitting with $F = 2$ and $m = n - 2$ is evident.

For $t > 2$ we finally add $2(t-2)$ further items $x_i = 1/2$ to the instance. Arguments are similar; note that the additional items must form $t - 2$ pairs, in a partitioning into heavy integral sets. ◀

4.3 Few agents make it easy, too

The reductions showing hardness led to instances with $m/n \geq 2/3$. The problem becomes "easy" if m is close to the smallest possible value $n/2$. (Readers not being familiar with fixed-parameter tractability are referred to a textbook like [7].)

▶ **Theorem 17.** Perfect Splitting *with $F = 2$ is fixed-parameter tractable (FPT) in the parameter $t := 2m - n$.*

Proof. Consider graphs with n vertices and m edges. We refer to connected components with two and three vertices as pairs and triplets, respectively. As a preparatory consideration we construct extremal graphs where the number q of vertices not being in pairs is maximized. We can assume that some pairs exist, since otherwise $q = n$ is, trivially, maximal.

If some connected component C is not a tree, then we can remove an edge such that C remains connected. We use this edge to join some pair to another component. This strictly increases q. Hence assume that all connected components are trees. If some tree has at least four vertices, then we can remove a leaf and its incident edge. The remainder of the tree is still connected and is not a pair. We append the edge and the leaf to some pair, which strictly increases q again. Hence, if q is maximized, then all connected components are pairs and triples (unless $q = n$). With p pairs and t' triples we have $m = p + 2t'$ and $n = 2p + 3t'$, hence $t = 2m - n = t'$. Since the maximum q is shown to be $q = 3t' = 3t$, we get that at most $3t$ vertices are not in pairs.

Now consider any instance of Perfect Splitting with $F = 2$, and any two indices i and j with $x_i + x_j = 1$. Assume that the instance has a solution. We consider the partitioning specified in Theorem 14 and refer to its heavy integral sets as bags.

Assume that i and j are in different bags, say, in $I \cup \{i\}$ and $J \cup \{j\}$. We rearrange them to new bags $\{i, j\}$ and $I \cup J$. The pair is obviously a heavy integral set. Since the original bags were integral and the weight $x_i + x_j = 1$ has been removed, also $I \cup J$ is integral. Since the original bags were heavy, $I \cup \{i\}$ and $J \cup \{j\}$ have sums at least $|I|$ and $|J|$, respectively. Hence $I \cup J$ has a sum at least $|I| + |J| - 1$, which means that it is heavy. This shows, under the above assumption, the existence of an alternative solution where $\{i, j\}$ is a bag.

Assume that i and j are in the same bag, but together with further indices, say, $K \cup \{i, j\}$ with $K \neq \emptyset$ is a bag. We split it in two bags K and $\{i, j\}$. Clearly, K is integral. Since $K \cup \{i, j\}$ was heavy, it has a sum at least $|K| + 1$, such that at least a sum $|K|$ remains in K, which means that K is also heavy. This shows again the existence of an alternative solution where $\{i, j\}$ is a bag.

We are ready to devise an FPT algorithm: First we pair up indices i and j with $x_i + x_j = 1$, as long as possible. (This is the "data reduction" phase, in the terminology of FPT algorithms.) Then we solve the residual instance, that is, we search for a partitioning as in Theorem 14, of the remaining indices. The given instance has a perfect solution if and only if the residual instance has.

The pairing phase is correct due to the above exchange arguments: If a solution exists at all, then it can be transformed into a solution where any desired pair $\{i, j\}$ with $x_i + x_j = 1$ forms a bag. By traversing a sorted list of the weights simultaneously in ascending and descending order, this phase is easily implemented in $O(n \log n)$ time. The residual instance has a size at most $3t$, and we may solve it naively. ◀

By way of contrast, Theorem 16 excludes an FPT (even an XP) algorithm in the parameter $t = n - m$ (unless P=NP).

5 Further Research

By Theorem 15, the optimization versions of or splitting problems with $F = 2$ (and $n > m$) are strongly NP-complete and therefore do not allow FPTAS (unless P=NP). On the other hand, the structural result in Theorem 14 together with the algorithms in Theorem 12 suggests that one should try and partition the set of pieces into $n - m$ heavy subsets (bags) and then assign agents to them such that the average amounts per agent in the bag are as balanced as possible. The latter step would work as in case $F = 1$, treating the bags as pieces. In general, the bags will not be integral, but the smaller the fractional parts of the sums are, the better the solutions should be. Minimizing the fractional parts roughly resembles the minimum makespan scheduling problem (although the objectives are also quite different). The latter problem is well studied: It is also strongly NP-complete, but it has a PTAS, and even an FPTAS when the number of machines (here corresponding to the number $n - m$ of bags) is fixed; see [3, 4]. Altogether, we conjecture that similar approximation schemes can be obtained for our splitting problems. Another conjecture is that our FPT result for small $2m - n$ extends to the optimization versions.

Other open questions were mentioned in the technical sections. Furthermore, as we have seen, the "graph-theoretic" case $F = 2$ is already subtle, but several results may be generalized to $F > 2$.

References

1 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

2 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

3 Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, 1988. doi:10.1137/0217033.

4 Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 23(2):317–327, 1976. doi:10.1145/321941.321951.

5 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

6 John Martinovic and Guntram Scheithauer. Integer rounding and modified integer rounding for the skiving stock problem. *Discrete Optimization*, 21:118–130, 2016. doi:10.1016/j.disopt.2016.06.004.

7 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Univ. Press, 2006.

Small-Space LCE Data Structure with Constant-Time Queries

Yuka Tanimura[1], Takaaki Nishimoto[2], Hideo Bannai[3], Shunsuke Inenaga[4], and Masayuki Takeda[5]

1 **Department of Informatics, Kyushu University, Japan**
2 **RIKEN Center for Advanced Intelligence Project, Chuo-ku, Tokyo, Japan**
takaaki.nishimoto@riken.jp
3 **Department of Informatics, Kyushu University, Japan**
bannai@inf.kyushu-u.ac.jp
4 **Department of Informatics, Kyushu University, Japan**
inenaga@inf.kyushu-u.ac.jp
5 **Department of Informatics, Kyushu University, Japan**
takeda@inf.kyushu-u.ac.jp

Abstract

The *longest common extension* (*LCE*) problem is to preprocess a given string w of length n so that the length of the longest common prefix between suffixes of w that start at any two given positions is answered quickly. In this paper, we present a data structure of $O(z\tau^2 + \frac{n}{\tau})$ words of space which answers LCE queries in $O(1)$ time and can be built in $O(n \log \sigma)$ time, where $1 \leq \tau \leq \sqrt{n}$ is a parameter, z is the size of the Lempel-Ziv 77 factorization of w and σ is the alphabet size. The proposed LCE data structure does not access the input string w when answering queries, and thus w can be deleted after preprocessing. On top of this main result, we obtain further results using (variants of) our LCE data structure, which include the following:

- For highly repetitive strings where the $z\tau^2$ term is dominated by $\frac{n}{\tau}$, we obtain a *constant-time and sub-linear space* LCE query data structure.
- Even when the input string is not well compressible via Lempel-Ziv 77 factorization, we still can obtain a *constant-time and sub-linear space* LCE data structure for suitable τ and for $\sigma \leq 2^{o(\log n)}$.
- The time-space trade-off lower bounds for the LCE problem by Bille et al. [J. Discrete Algorithms, 25:42-50, 2014] and by Kosolobov [CoRR, abs/1611.02891, 2016] do not apply in some cases with our LCE data structure.

1998 ACM Subject Classification G.2.1 Combinatorial Algorithms

Keywords and phrases longest common extension, truncated suffix trees, t-covers

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.10

1 Introduction

1.1 The LCE problem

The *longest common extension* (*LCE*) problem is to preprocess a given string w of length n so that the length of the longest common prefix of suffixes of w starting at two query positions is answered quickly. The LCE problem often appears as a sub-problem of many different string processing problems, e.g., approximate pattern matching [35, 17], string comparison [34],

42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017).
Editors: Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin; Article No. 10; pp. 10:1–10:15

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

and finding string regularities such as maximal repetitions (a.k.a. runs) [30, 1], distinct squares [22, 2], gapped repeats [11, 31, 18, 14], palindromes and gapped palindromes [21, 29, 39], and 2D palindromes [20].

A well known solution to the LCE problem is achieved by the suffix tree [47] augmented with a constant-time linear-space longest common ancestor (LCA) data structure [23, 3], or equivalently the inverse suffix array (ISA) and longest common prefix (LCP) array augmented with a constant-time range minimum query (RMQ) data structure [37, 3]. Either combination uses $O(n)$ words of space, answer LCE queries in $O(1)$ time, and can be constructed using $O(n)$ words of working space, in $O(n)$ time for integer alphabets or in $O(n \log \sigma)$ time for general ordered alphabets of size σ. The $O(n)$ space requirements, however, can be prohibitive for massive text, and hence the main focus of recent research has been on more space-efficient solutions with trade-offs for query time.

1.2 Space-efficient LCE data structures

In this paper, we will call data structures that use $o(n)$ words, or equivalently $o(n \log n)$ bits as *sub-linear space* data structures. Bille et al. [9] proposed the first *sub-linear* space LCE query data structure which occupies $O(\frac{n}{\tau})$ words of space, answers LCE queries in $O(\tau^2)$ time, and can be built in $O(\frac{n^2}{\tau})$ time using $O(\frac{n}{\tau})$ words of working space, for parameter range $1 \leq \tau \leq \sqrt{n}$. Bille et al. [8] developed an improved sub-linear space data structure which occupies $O(\frac{n}{\tau})$ words of space, answers LCE queries in $O(\tau)$ time, and can be built in $O(n^{\frac{3}{2}})$ *expected* time using $O(\frac{n}{\tau})$ words of working space, or in $O(n^{2+\epsilon})$ time using $O(\frac{n}{\tau})$ words of working space for parameter $1 \leq \tau \leq n$, where $0 < \epsilon < 1$. Tanimura et al. [45] proposed an LCE data structure of $O(\frac{n}{\tau})$ words of space, which can be built in faster $O(n\tau)$ time using $O(\frac{n}{\tau})$ words of working space, but takes slower $O(\tau \log \min\{\tau, \frac{n}{\tau}\})$ time for LCE queries, for parameter $1 \leq \tau \leq n$. All of these sub-linear space LCE data structures need to access to the input string when answering queries. Therefore, these data structures require extra $n\lceil \log \sigma \rceil$ *bits* of space for storing the input string. An in-place LCE data structure based on fingerprints was recently proposed [42].

There also exist *compressed* LCE data structures which store a compressed form of the input string represented as a straight-line program (a.k.a. grammar-based text compression) [41, 25, 6, 7, 24]. For compressible strings, the space usage of these data structures can be sub-linear.

1.3 Our LCE data structure

This paper proposes the *first* $O(1)$*-time* LCE data structure which takes *sub-linear space* in several reasonable cases, namely, when the string is compressible, and/or, when the alphabet size is suitably small. Our data structure has both flavours of sub-linear space and compressed LCE data structures. Namely, for parameter $1 \leq \tau \leq \sqrt{n}$, we present an LCE data structure which takes $O(z\tau^2 + \frac{n}{\tau})$ words of space, answers LCE queries in $O(1)$ time, and can be built in $O(n \log \sigma)$ time for general ordered alphabets of size σ using $O(z\tau^2 + \frac{n}{\tau})$ words of working space, where z is the size of the Lempel-Ziv 77 factorization [48] of the input string. It is known that z is a lower bound on the size of *any* grammar-based compression of the string [44], and can be very small for highly repetitive strings. In such cases where the $z\tau^2$ term is dominated by $\frac{n}{\tau}$, our LCE data structure uses sub-linear space. An interesting feature is that we do *not* actually compress the input string, i.e., do not compute the Lempel-Ziv 77 factorization, but we construct a data structure whose size is bounded by $O(z\tau^2 + \frac{n}{\tau})$.

Table 1 Deterministic LCE query data structures. n is the length of the input string, σ is the alphabet size, z is the size of the Lempel-Ziv 77 factorization of w, l is the length of the LCE, ω is the machine word size, $\epsilon > 0$ is an arbitrarily small constant, and τ is a trade-off parameter ($\dagger : 1 \leq \tau \leq n$, $\diamond : 1 \leq \tau \leq \sqrt{n}$). ISA+ consists of the inverse suffix array of w, the LCP array and the RMQ data structure. $\star$ is valid for $\omega = \Theta(\log n)$ and $\sigma \leq 2^{o(\log n)}$.

Data structure		Preprocessing		Ref
Space (bits)	Query Time	Working space	Construction time	
$O(\omega)$	$O(n)$	$O(\omega)$	-	naïve
$O(n\omega)$	$O(1)$	$O(n\omega)$	$O(n)$	ISA+
$n\lceil\log\sigma\rceil + O(\frac{n\omega}{\tau})$	$O(\tau^2)$	$O(\frac{n\omega}{\tau})$	$O(n^2/\tau)$	$\diamond$ [9]
$n\lceil\log\sigma\rceil + O(\frac{n\omega}{\tau})$	$O(\tau)$	$O(\frac{n\omega}{\tau})$	$O(n^{3/2})$ exp.	$\dagger$ [8](1)
$n\lceil\log\sigma\rceil + O(\frac{n\omega}{\tau})$	$O(\tau)$	$O(\frac{n\omega}{\tau})$	$O(n^{2+\epsilon})$	$\dagger$ [8](2)
$n\lceil\log\sigma\rceil + O(\frac{n\omega}{\tau})$	$O(\tau \log\min\{\tau, \frac{n}{\tau}\})$	$O(\frac{n\omega}{\tau})$	$O(n\tau)$	$\dagger$ [45]
$n\lceil\log\sigma\rceil + O(\omega \log n)$	$O(\log l)$	$O(\omega \log n)$	$O(n \log n)$ exp.	[42]
$O(z\omega \log n \log^* n)$	$O(\log n \log^* n)$	$O(z\omega \log n \log^* n)$	$O(n \log \sigma)$	[41],[25]
$O(z\omega \log \frac{n}{z})$	$O(\log n)$	$O(n\omega)$	$O(n)$	[24]
$O(z\omega \log \frac{n}{z})$	$O(\log n)$	$O(n \log \sigma + z\omega \log \frac{n}{z})$	$O(n \log\log \sigma + z \log^2 \frac{n}{z})$	[24]+[32]
$O((z\tau^2 + \frac{n}{\tau})\omega)$	$O(1)$	$O((z\tau^2 + \frac{n}{\tau})\omega)$	$O(n \log \sigma)$	$\diamond$ ours
$O(z^{1/3}n^{2/3}\omega)$	$O(1)$	$O(z^{1/3}n^{2/3}\omega)$	$O(n \log \sigma \log n)$	ours
$o(n \log n)$	$O(1)$	$o(n \log n)$	$o(n \log^2 n)$	$\star$ ours
$O(\sqrt{nz}\omega)$	$O(\sqrt{\frac{n}{z}})$	$O(\sqrt{nz}\omega)$	$O(n \log \sigma \log n)$	ours

Even when the input string is not well compressible via Lempel-Ziv 77, for suitably small alphabets, we can build a sub-linear space LCE data structure with $O(1)$ query time using appropriate values of τ. By choosing $\tau = (\frac{n}{z})^{\frac{1}{3}}$, our LCE query data structure takes $O(z^{\frac{1}{3}}n^{\frac{2}{3}})$ words of space, which translates to $O(n/(\log_\sigma n)^{\frac{1}{3}})$ using the well-known fact that $z = O(n/\log_\sigma n)$ (e.g. [26]). This means that our data structure can be stored in $O(n \log n/(\log_\sigma n)^{\frac{1}{3}}) = O(n(\log n)^{\frac{2}{3}}(\log \sigma)^{\frac{1}{3}})$ *bits* of space. This implies that for alphabets of size $\sigma \leq 2^{o(\log n)}$ (note that these contain polylogarithmic alphabets), our data structure takes only $o(n \log n)$ bits of space, yet answers LCE queries in $O(1)$ time. Also, our LCE data structure does not access the input string when answering queries, and hence the input string does not have to be kept. To our knowledge, this is the first sub-linear space LCE data structure for strings which are not well compressible with Lempel-Ziv 77.

The key to our efficient LCE query data structure is a hybrid use of the *truncated suffix trees* [38] and block-wise LCE queries based on t-covers [43, 9]. The q-truncated suffix tree of a string w is the compact trie (a.k.a. Patricia tree) which represents all substrings of w of length at most q. We observe that, for any $1 \leq q \leq n$, the q-truncated suffix tree can be stored in $O(zq)$ words of space, including a string to which the edges label pointers refer. We also show that the block-wise LCE query data structure based on t-covers can be efficiently built by the t-truncated suffix tree, leading to our result. Several variants of our data structure are considered, as summarized in Table 1.

The rest of this paper is organized as follows. Section 2 gives some definitions and introduces tools which will be used as building-blocks of our LCE data structure. In Section 3 we propose our new LCE data structure and analyze its time/space complexities. In Section 4 we review some lower bounds on the LCE problem and show that using our LCE data structure, these lower bounds do not apply in some cases. We conclude in Section 5.

2 Preliminaries

2.1 Notations

Let Σ be an ordered alphabet of size σ. Each element of Σ^* is called a *string*. The length of a string w is denoted by $|w|$. The empty string ε is the string of length zero, namely $|\varepsilon| = 0$. If $w = xyz$ for some strings w, x, y, z, then x, y, and z are respectively called a *prefix*, *substring*, and *suffix* of w. For any $1 \leq i \leq |w|$, let $w[i]$ denote the ith character of w. For any $1 \leq i \leq j \leq |w|$, let $w[i..j]$ denote the substring of w that begins at position i and ends at position j, namely, $w[i..j] = w[i] \cdots w[j]$. A string of length q is called a *q-gram*. For any $1 \leq q \leq |w|$, let $Substr_q(w)$ denote the set of all q-grams occurring in w and the $q-1$ suffixes of w of length shorter than q, namely, $Substr_q(w) = \{w[i..\min\{i+q-1, |w|\}] \mid 1 \leq i \leq |w|\}$.

For any string w, let $\mathsf{LCE}^w(i,j)$ denote the length of the longest common prefix of $w[i..|w|]$ and $w[j..|w|]$. We will write $\mathsf{LCE}(i,j)$ when w is clear from the context. Since $\mathsf{LCE}^w(i,i) = |w| - i$, we will only consider the case when $i \neq j$. For any integers $i \leq j$, let $[i..j]$ denote the set of integers from i to j (including i and j).

The *Lempel-Ziv 77 factorization with self-references* [48] of a string w is a sequence $\mathsf{LZ}(w) = f_1, \ldots, f_z$ of z non-empty substrings of w such that $w = f_1 \cdots f_z$ and for $1 \leq i \leq z$,

- $f_i = w[|f_1 \cdots f_{i-1}| + 1] \in \Sigma$ if $s[|f_1 \cdots f_{i-1}| + 1]$ is a character not occurring in $f_0 \cdots f_{i-1}$,
- f_i is the longest prefix of $f_i \cdots f_z$ such that f_i is a substring of w beginning at a position in range $[1..|f_1 \cdots f_{i-1}|]$,

where $f_0 = \varepsilon$. The *size* of $\mathsf{LZ}(w)$ is the number z of factors $f_1, \ldots, f_z$, and is denoted as $|\mathsf{LZ}(w)| = z$. For instance, for string $w = \mathtt{abababcababababcababababcd}$ of length 22, $\mathsf{LZ}(w) = \mathtt{a}, \mathtt{b}, \mathtt{abab}, \mathtt{c}, \mathtt{abababcababababc}, \mathtt{d}$ and $|\mathsf{LZ}(w)| = 6$.

Our model of computation is a standard word RAM with machine word size $\omega \geq \log n$. The space requirements will be evaluated by the number of words unless otherwise stated.

2.2 Tools

We will use the following tools as building blocks of our LCE data structure.

t-covers. For any positive integer t, a set $D \subseteq [0..t-1]$ is called a *t-difference-cover* if $[0..t-1] = \{(x-y) \bmod t \mid x, y \in D\}$, namely, every element in $[0..t-1]$ can be expressed by a difference between two elements in D modulo t. For any positive integer n, a set $S \subseteq [1..n]$ is called a *t-cover* of $[1..n]$ if $S = \{i \in [1..n] \mid (i \bmod t) \in D\}$ with some t-difference-cover D, and there is a constant-time computable function $h(i,j)$ that for any $1 \leq i, j \leq n-t$, $0 \leq h(i,j) \leq t$ and $i + h(i,j), j + h(i,j) \in S$.

▶ **Lemma 1** ([36]). *For any integer t, there exists a t-difference-cover $D(t)$ of size $O(\sqrt{t})$ which can be computed in $O(\sqrt{t})$ time.*

▶ **Lemma 2** ([12]). *For any integer t ($\leq n$), there exists a t-cover of size $O(\frac{n}{\sqrt{t}})$ which can be computed in $O(\frac{n}{\sqrt{t}})$ time.*

In what follows, we will denote by $S(t)$ an arbitrary t-cover of $[1..n]$ which satisfies the conditions of Lemma 2. See Figure 1 for an example of a t-cover $S(t)$.

Truncated suffix trees. For convenience, we assume that any string w ends with a special end-marker \$ that appears nowhere else in w. Let $n = |w|$. For any $1 \leq q \leq n$, the *q-truncated suffix tree* of w, denoted q-$\mathsf{TST}(w)$, is a Patricia tree which represents $Substr_q(w)$. Namely, q-$\mathsf{TST}(w)$ is an edge-labeled rooted tree such that: (1) Each edge is labeled with a non-empty

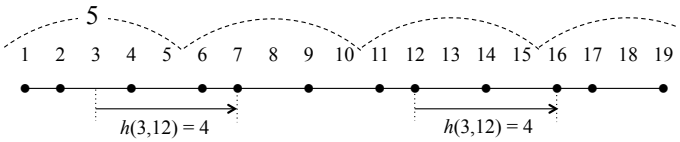


Figure 1 Let $t = 5$ and $D = \{1, 2, 4\}$. This figure shows an example of a 5-cover $S(5) = \{1, 2, 4, 6, 7, 9, 11, 12, 14, 16, 17, 19\}$. The black dots represent the elements in $S(5)$. For instance, we have $h(3, 12) = 4$, namely, $3 + 4, 12 + 4 \in S(5)$.

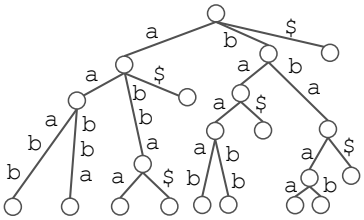


Figure 2 5-TST(w) with string $w =$ `baabbaabbaaabbaabba`\$.

substring of w; (2) Each internal node v has at least two children, and the labels of the out edges of v begin with distinct characters; (3) For any leaf u, there is at least one position $1 \leq i \leq n$ such that $w[i..\min\{i + q - 1, n\}]$ is the string obtained by concatenating the edge labels from the root to u; (4) For any position $1 \leq i \leq n$ in w, there is a unique leaf u such that $w[i..\min\{i + q - 1, n\}]$ is the string obtained by concatenating the edge labels from the root to u.

Informally speaking, q-TST(w) can be obtained by trimming the full suffix tree of w so that any path from the root represents a substring of length at most q. Clearly, the number of leaves in q-TST(w) is equal to $|Substr_q(w)|$. We assume that the leaves of q-TST(w) are sorted in lexicographical order. Figure 2 shows an example of a q-TST(w). For any node u of q-TST(w), $str(u)$ denotes the string spelled out by the path from the root to u.

In the case of the full suffix tree (n-TST(w)) of string w of length n, each edge label x is represented by a pair (i, j) of positions in w such that $x = w[i..j]$. We call w as the *reference string* for the full suffix tree, and this way the full suffix tree can be stored in $O(n)$ space. For q-TST(w), Vitale et al. [46] showed how to represent q-TST(w) in $O(|Substr_q(w)|)$ space, including the reference string, and how to construct them efficiently, both in time and space.

▶ **Lemma 3** ([46]). *Let w be any string of length n over an ordered alphabet of size σ. For any $1 \leq q \leq n$, let $y = |Substr_q(w)|$. Then, there exists a reference string w' of length $O(y)$ for q-TST(w). Moreover, q-TST(w) with the leaves sorted in lexicographical order, and a reference string w' can be constructed in $O(n \log \sigma)$ time with $O(y)$ working space.*

We also show the following lemma.

▶ **Lemma 4.** *q-TST(w) can be represented in $O(zq)$ space, where $z = |$LZ(w)$|$.*

Proof. By Lemma 3, it suffices to show that $|Substr_q(w)| = O(zq)$. For each q-gram $p \in Substr_q(w)$, let $locc_w(p)$ be the beginning position of the leftmost occurrence of p in w. If $q = 1$, then clearly $|Substr_1(w)| \leq z$ and hence the lemma holds. If $q \geq 2$ then the interval $[locc_w(p)..locc_w(p) + q - 1]$ must cross the boundary of two adjacent factors of LZ(w), since otherwise the interval is completely contained in a single factor of LZ(w) but this contradicts that $[locc_w(p)..locc_w(p) + q - 1]$ is the leftmost occurrence of p in w. Clearly, the maximum

number of q-grams that can cross a boundary of $\mathsf{LZ}(w)$ is $q-1$. Hence, the total number of distinct q-grams in w is $O(zq)$. Also, $Substr_q(w)$ contains q substrings $w[n-q+1], \ldots, w[n]$ of w which are shorter than q. Overall, we obtain $|Substr_q(w)| = O(zq)$. ◀

The next theorem follows from Lemmas 3 and 4 and an obvious fact that $|Substr_q(w)| \leq n$.

▶ **Theorem 5.** *Given a string w of length n over an ordered alphabet of size σ and integer $1 \leq q \leq n$, we can construct an $O(\min\{zq, n\})$-space representation of q-$\mathsf{TST}(w)$ in $O(n \log \sigma)$ time with $O(\min\{zq, n\})$ working space.*

In what follows, we will only consider interesting cases where $zq < n$ for a given $1 \leq q \leq n$, and will simply use $O(zq)$ to denote the size of q-$\mathsf{TST}(w)$.

3 Our LCE data structure

3.1 Overview of our algorithm

The general framework of our space-efficient LCE algorithm follows the approach of Gawrychowski et al.'s LCE algorithm for strings over a general ordered alphabet [19]. Namely, we compute $\mathsf{LCE}(i,j)$ using the two following types of queries:

$$\mathsf{ShortLCE}_t(i,j) = \min(\mathsf{LCE}(i,j), t),$$

$$\mathsf{LongLCE}_t(i,j) = \begin{cases} \lfloor \mathrm{LCE}(i,j)/t \rfloor & \text{if } i,j \in S(t), \\ \perp & \text{otherwise.} \end{cases}$$

$\mathsf{LCE}(i,j)$ is computed in the following manner. Let $\delta = h(i,j)$. Recall that $\delta \leq t$ can be computed in constant time and that $i+\delta, j+\delta \in S(t)$. First, we compare up to the first δ characters of $w[i..|w|]$ and $w[j..|w|]$ using $\mathsf{ShortLCE}_t(i,j)$. If $l_1 = \mathsf{ShortLCE}_t(i,j)$ is shorter than t, then $\mathsf{LCE}(i,j) = l_1$. If $l_1 = t$, then $\mathsf{LCE}(i,j)$ is at least t long. To check if it further extends, we compute $l_2 = \mathsf{LongLCE}_t(i+\delta, j+\delta)$, and $l_3 = \mathsf{ShortLCE}_t(i+\delta+l_2, j+\delta+l_2)$. Finally, we get $\mathsf{LCE}(i,j) = \delta + t \cdot l_2 + l_3$. See also Figure 3.

The main difference between Gawrychowski et al.'s method and ours is in how to compute $\mathsf{ShortLCE}_t(i,j)$. While they use a Union-Find structure that takes $O(n)$ working space (for $O(n)$ queries) as a main tool, we use an augmented $2t$-$\mathsf{TST}(w)$ for $Substr_{2t}(w)$ which occupies $O(zt + \frac{n}{\sqrt{t}})$ total space, answers $\mathsf{ShortLCE}_t(i,j)$ queries in $O(1)$ time, and can be constructed in $O(n \log \sigma)$ time with $O(zt)$ working space. How to answer $\mathsf{LongLCE}_t(i,j)$ queries is equivalent to Gawrychowski et al.'s, namely, we sample the positions from $S(t)$ so that LCE queries for these sampled positions can be answered in $O(1)$ time. We show how to build the data structure for $\mathsf{LongLCE}_t(i,j)$ queries by using t-$\mathsf{TST}(w)$ for $Substr_t(w)$ in $O(n \log \sigma)$ time with $O(zt)$ working space.

3.2 $\mathsf{ShortLCE}_t$ queries

For $\mathsf{ShortLCE}_t(i,j)$ queries, we use $2t$-$\mathsf{TST}(w)$ which represents the set $Substr_{2t}(w)$ of all substrings of w of length at most $2t$. For any position $1 \leq i \leq n$, let p_i denote the substring of w that begins at position i and is of length at most $2t$, namely, $p_i = w[i..\min\{i+2t-1, n\}]$. Notice that $Substr_{2t}(w) = \bigcup_{i=1}^{n} \{p_i\}$. For any position $1 \leq i \leq n$ in w, let $\ell(i) = u$ iff u is the leaf of $2t$-$\mathsf{TST}(w)$ such that $str(u) = p_i$. Basically, we will compute $\mathsf{ShortLCE}_t(i,j)$ by efficiently finding the LCA of the corresponding leaves $\ell(i)$ and $\ell(j)$ on $2t$-$\mathsf{TST}(w)$. The reason that we use $2t$-$\mathsf{TST}(w)$ rather than t-$\mathsf{TST}(w)$ will become clear later.

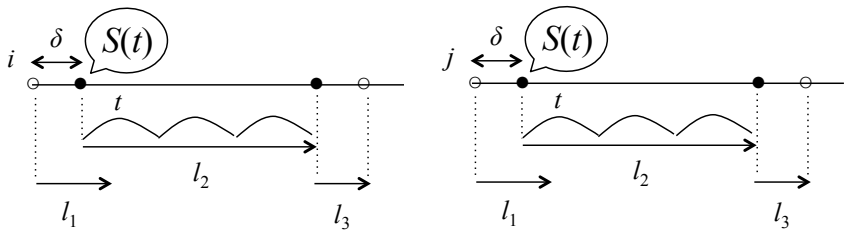


Figure 3 Illustration of an overview of our $\mathsf{LCE}(i,j)$ algorithm. We are given two positions i and j in string w. First, we compute $l_1 = \mathsf{ShortLCE}_t(i,j)$. If $l_1 < t$, then $\mathsf{LCE}(i,j) = l_1$. Otherwise, we compute $\mathsf{LongLCE}_t(i+\delta, j+\delta)$ where $i+\delta, j+\delta \in S(t)$ and $0 \le \delta \le t$. We finally compute $l_3 = \mathsf{ShortLCE}(i+\delta+l_2, j+\delta+l_2)$ where $l_2 = t\mathsf{LongLCE}_t(i+\delta, j+\delta)$. Then $\mathsf{LCE}(i,j) = \delta + l_2 + l_3$.

Now the key is how to access $\ell(i)$ for a given position i in w. As our goal is to build a sub-linear space data structure for $\mathsf{ShortLCE}_t$ queries, we cannot afford to store a pointer to $\ell(i)$ from every position $1 \le i \le n$. Thus, we store such a pointer only from every t-th positions in w. We call these positions as *sampled positions*. Formally, for every sampled position $j \in Q_{t,n} = \{1 + kt \mid 0 \le k \le \lceil \frac{n}{t} \rceil - 1\}$ we explicitly store a pointer from j to its corresponding leaf $\ell(j)$ on $2t$-$\mathsf{TST}(w)$. Also, for each position $1 \le i \le n$ in w, let $\alpha(i) = \max\{j \in Q_{t,n} \mid j \le i\}$. Namely, $\alpha(i)$ is the closest sampled position in $Q_{t,n}$ to the left of i (or it is i itself if $i \in Q_{t,n}$).

Given a position $1 \le i \le n$, $\alpha(i)$ can be computed in $O(1)$ time with a simple arithmetic operation. Hence, we can access the leaf $\ell(\alpha(i))$ for the closest sampled position $\alpha(i)$ in $O(1)$ time. The next task is to locate $\ell(i)$. To describe our constant-time algorithm, let us consider a conceptual graph $G = (V, E)$ such that $V = \mathit{Substr}_{2t}(w)$ and $E = \{(u, c, v) \mid u[1..2t-1] = v[2..2t], c = v[1]\}$, where (u, c, v) represents a directed edge labeled c from u to v. This graph G is equivalent to the edge-reversed de Bruijn graph of order $2t$, with extra nodes for the $2t-1$ suffixes of w which are shorter than $2t$. It is clear that there is a one-to-one correspondence between the leaves of $2t$-$\mathsf{TST}(w)$ and the nodes of the graph G. Thus, we will identify each leaf of $2t$-$\mathsf{TST}(w)$ with the nodes of graph G.

▶ **Lemma 6.** *Given $2t$-$\mathsf{TST}(w)$ for a string w of length n and for any $1 \le 2t \le n$, we can construct the graph G in $O(n)$ time using $O(zt)$ working space.*

Proof. The de Bruijn graph of order q for a string of length n can be constructed in $O(n)$ time using space linear in the size of the output de Bruijn graph, provided that q-$\mathsf{TST}(w)$ is already constructed [13]. By setting $q = 2t$, adding extra $2t-1$ nodes for the suffixes that are shorter than $2t$, and reversing all the edges, we obtain our graph $G = (V, E)$.

The number of nodes in V is clearly equal to $|\mathit{Substr}_{2t}(w)|$. Also, since each edge in E corresponds to a distinct substring in $\mathit{Substr}_{2t+1}(w)$, the number of edges in E is equal to $|\mathit{Substr}_{2t+1}(w)|$. By a similar argument to the proof of Lemma 4, we obtain $|V| = |\mathit{Substr}_{2t}(w)| = O(zt)$ and $|E| = |\mathit{Substr}_{2t+1}(w)| = O(zt)$. ◀

Let $d = i - \alpha(i)$. A key observation here is that there is a path of length d from node p_i to node $p_{\alpha(i)}$ in this graph G. Since G is a graph, however, it is not easy to quickly move from $p_{\alpha(i)}$ to p_i. To overcome this difficulty, we consider a spanning tree of G of which the root is $p_n = w[n] = \$$. Let T denote any spanning tree of G. See Figure 4 for examples of the graph G and its spanning tree T. Although some edges are lost in spanning tree T, it is enough for our purpose. Namely, the following lemma holds.

▶ **Lemma 7.** *Any spanning tree T of G satisfies the following properties: (1) There is a non-branching path of length $2t$ from the root p_n to the node p_{n-2t}. (2) For any $1 \leq i \leq n-2t-1$ and $0 \leq d < t$, let g be the d-th ancestor of $p_{\alpha(i)}$. Then, $g[1..t] = p_i[1..t]$.*

Proof. The first property is immediate from the fact that the last character $w[n] = \$$ occurs nowhere else in w and the root represents $p_n = \$$.

Since $d < t$ and $|p_{\alpha(i)}| = 2t$, we have $p_{\alpha(i)}[d..d+t-1] = p_i[1..t]$. By the first property and $\alpha(i) \leq i \leq n-2t-1$, the depth of node $p_{\alpha(i)}$ is at least $2t$. Also, by following the in-coming edge of each node in the reversed direction, we delete the first character of the corresponding string. Hence, $p_i[1..t]$ is a prefix of the d-th ancestor g of $p_{\alpha(i)}$. ◀

We are ready to show the main result of this section.

▶ **Theorem 8.** *For any string w of length n and integer $1 \leq t \leq n$, a data structure of size $O(zt + \frac{n}{t})$ can be constructed in $O(n \log \sigma)$ time using $O(zt)$ working space such that subsequent $\mathsf{ShortLCE}_t(i,j)$ queries for any $1 \leq i, j \leq n$ can be answered in $O(1)$ time, where $z = |\mathsf{LZ}(w)|$.*

Proof. We use a spanning tree T enhanced with a *level ancestor* data structure [5, 4] which can be constructed in time and space linear in the size of the input tree T.

Given two positions i, j in w, we answer $\mathsf{ShortLCE}_t(i,j)$ query as follows:

1. Compute the closest sampled positions $\alpha(i)$ and $\alpha(j)$ by simple arithmetics.
2. Access the nodes $p_{\alpha(i)}$ and $p_{\alpha(j)}$ in the spanning tree T using pointers from the sampled positions $\alpha(i)$ and $\alpha(j)$, respectively.
3. Let $d = i - \alpha(i)$ and $d' = j - \alpha(j)$. Access the d-th ancestor u of $p_{\alpha(i)}$ and the d'-th ancestor of $p_{\alpha(j)}$ using level ancestor queries on T.
4. Compute the LCA x of the two leaves u and v on $2t$-$\mathsf{TST}(w)$, and return $\min\{|str(x)|, t\}$.

The correctness follows from Lemma 7. Since each step of the above algorithm takes $O(1)$ time, we can answer $\mathsf{ShortLCE}_t(i,j)$ in $O(1)$ time. By Lemma 4, the size of $2t$-$\mathsf{TST}(w)$ with an LCA data structure is $O(zt)$, and also the size of the spanning tree T with a level ancestor data structure is $O(|Substr_{2t}(w)|) = O(zt)$. In addition, we store pointers from the $\Theta(\frac{n}{t})$ sampled positions to their corresponding nodes in T. Overall, the total space requirement of our data structures is $O(zt + \frac{n}{t})$. We can build these data structures in a total of $O(n \log \sigma)$ time using $O(zt)$ working space by Theorem 5 and Lemma 6. ◀

3.3 $\mathsf{LongLCE}_t$ queries

At a high level, our $\mathsf{LongLCE}_t(i,j)$ query algorithm is an adaptation of the t-cover based algorithm by Puglisi and Turpin [43], which was later re-discovered by Bille et al. [9]. Gawrychowski et al. [19] showed that an $O(\frac{n}{\sqrt{t}})$-space data structure, which answers $\mathsf{LongLCE}_t(i,j)$ query in $O(1)$ time, can be constructed in $O(n \log t)$ time with $t = \Omega(\log^2 n)$ for a string of length n over a general ordered alphabet. In this section, we show the same data structure as Gawrychowski et al. can be constructed in $O(n \log \sigma)$ time with $O(zt + \frac{n}{t})$ working space for a general ordered alphabet of size σ and any $1 \leq t \leq n$.

Consider a t-cover $S(t)$ of $[1..n]$ for some t-difference-cover D. For each position $i \in S(t)$ such that $i + t - 1 \leq n$, the substring $b_i = w[i..i+t-1]$ is said to be a *t-block*. The goal here is to answer the block-wise LCE value $\mathsf{LongLCE}_t(i,j)$ for two given positions in the t-cover $S(t)$. Since we query $\mathsf{LongLCE}_t(i,j)$ only for positions $i, j \in S(t)$ and the answer to $\mathsf{LongLCE}_t(i,j)$ is a multiple of t, we can regard each t-block as a single character. Thus,

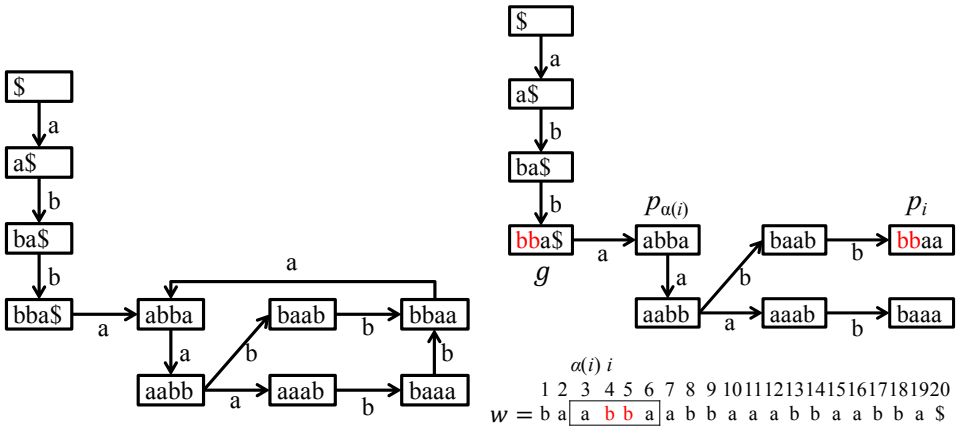


Figure 4 The left graph G is the edge-reversed de Bruijn graph of order $2t$, with extra nodes for the $2t-1$ suffixes of w which are shorter than $2t$, where $t=2$ and w is the same string as in Figure 2. An edge from u to v labeled character c represents $c \cdot u[1..2t-1] = v$. The right tree is a spanning tree of the left graph. Let $i=4$ and $\alpha(i)=3$. Then $p_i = bbaa$, $p_{\alpha(i)} = abba$. Let g be the d-th ancestor of $p_{\alpha(i)}$ in the right tree, where $0 \le d < t$. Then $g[1..t] = p_i[1..t]$ holds by Lemma 7.

we sort all t-blocks in lexicographical order, and encode each t-block by its lexicographical rank. Since each t-block is of length t, we can sort the t-blocks in $O(\frac{n}{\sqrt{t}} \log \frac{n}{\sqrt{t}})$ time with $O(zt + \frac{n}{\sqrt{t}})$ working space by using any suitable comparison-based sorting algorithm and our $O(1)$-time $\mathsf{ShortLCE}_t$ query data structure of Section 3.2. The next lemma shows that we can actually compute the lexicographical ranks of all t-blocks more efficiently.

▶ **Lemma 9.** *Let w be an input string of length n and $1 \le t \le n$ be an integer. Given the data structure for $\mathsf{ShortLCE}_t$ queries of Theorem 8 for w, we can sort all t-blocks of w in lexicographic order in $O(zt + \frac{n}{\sqrt{t}})$ time using $O(zt + \frac{n}{t})$ working space, where $z = |\mathsf{LZ}(w)|$.*

Proof. We insert new (non-branching) nodes to $2t$-$\mathsf{TST}(w)$ such that every t-gram in w is represented by an explicit node. This increases the size of the tree by a constant factor. We also associate each node u such that $|str(u)| = t$ with the lexicographical rank of the t-gram $str(u)$ among all t-grams in w. Then, we associate each leaf ℓ of the tree such that $|str(\ell)| \ge t$ with its ancestor v which represents a t-gram. All these can be preformed in $O(zt)$ total time by standard depth-first traversals on the tree.

Then, for each t-block $b_i = w[i..i+t-1]$, we can access a leaf ℓ of $2t$-$\mathsf{TST}(w)$ such that $str(\ell)[1..t] = w[i..i+t-1]$ in $O(1)$ time using the algorithm of Theorem 8, and we return the rank of the ancestor v of ℓ that represents $b_i = w[i..i+t-1]$. Since there are $O(\frac{n}{\sqrt{t}})$ t-blocks in w, it takes a total of $O(zt + \frac{n}{\sqrt{t}})$ time. The working space is $O(zt + \frac{n}{t})$ by Theorem 8. ◀

There is an alternative algorithm to sort the t-blocks, as follows:

▶ **Lemma 10.** *For any string w of length n over an alphabet of size σ, any integer $1 \le t \le n$, we can sort all t-blocks in lexicographic order in $O(n \log \sigma)$ time using $O(zt)$ working space, where $z = |\mathsf{LZ}(w)|$.*

Proof. We use t-$\mathsf{TST}(w)$ and the reversed de Bruijn graph of order t. We associate each leaf of the tree representing a t-gram with its lexicographical rank among all leaves in the tree.

Let r be the graph node which represents $w[n] = \$$. We simply traverse the graph while scanning the input string w from right to left. For each $1 \le i \le n$, this gives us the graph node representing $b_i = w[i..i+t-1]$ and hence the corresponding leaf of t-$\mathsf{TST}(w)$.

t-$\mathsf{TST}(w)$ and the reversed de Bruijn graph can be constructed in $O(n \log \sigma)$ time with $O(zt)$ working space. The ranks of the leaves in t-$\mathsf{TST}(w)$ can be easily computed in $O(zt)$ time by a standard tree traversal. Traversing the reversed de Bruijn graph takes $O(n \log \sigma)$ time. Hence the lemma holds. ◀

For each $i \in S(t)$, let r_i be the rank of the t-block $b_i = w[i..i+t-1]$ computed by any of the algorithms above. Clearly $r_i \in [1..n]$. For simplicity, assume $\sqrt{t}$ is an integer. For each position $i \in D$ (where D is the underlying t-difference cover), let $\#_i = r_i r_{i+t} \cdots r_{i+m_i t}$, where $m_i = \frac{n-i+1}{\sqrt{t}} - 1$. We create a string $code(w) = \#_1\$_1 \cdots \#_k\$_k$ of length $|S(t)| = O(\frac{n}{\sqrt{t}})$. Since each $\#_i$ is a string over the integer alphabet $[1..S(t)] \subset [1..n]$ and $|D| = O(\sqrt{t})$, we can regard $code(w)$ as a string over an integer alphabet of size $O(n)$. Then, we build the suffix array, the inverse suffix array, the LCP array [37] of $code(w)$ and an range minimum query (RMQ) data structure [3] for the LCP array. For any position $i \in S(t)$ on the original string w, we can compute its corresponding position i' on $code(w)$ as $i' = |\#_1\$_1\#_2\$_2 \cdots \#_{x-1}\$_{x-1}| + \frac{i-x}{t} + 1$ where $x = i \bmod t$. Now, $\mathsf{LongLCE}_t(i,j)$ query for two positions $i, j \in S(t)$ on the original string w reduces to an LCE query for the corresponding positions on $code(w)$, which can be answered in $O(1)$ time using an RMQ on the LCP array. All these arrays and the RMQ data structure can be built in $O(\frac{n}{\sqrt{t}})$ time [27, 28, 3].

▶ **Theorem 11.** *For any string of length n and integer $1 \le t \le n$, a data structure of size $O(\frac{n}{\sqrt{t}})$ can be constructed in $O(n \log \sigma)$ time using $O(zt + \frac{n}{t})$ working space such that subsequent $\mathsf{LongLCE}_t(i,j)$ queries for any $1 \le i, j \le n$ can be answered in $O(1)$ time, where $z = |\mathsf{LZ}(w)|$.*

Proof. We need $O(\frac{n}{\sqrt{t}})$ working space for the encoded string $code(w)$ and its suffix array plus LCP array enhanced with an RMQ data structure. Then the theorem follows from Theorem 8, and Lemma 9 or Lemma 10. ◀

3.4 Main result and variants

In what follows, let w be an input string of length n and $z = |\mathsf{LZ}(w)|$. By Theorem 8 and Theorem 11 shown in the previous subsections, we obtain the main theorem of this paper:

▶ **Theorem 12.** *For any integer $1 \le t \le n$, an LCE data structure of size $O(zt + \frac{n}{\sqrt{t}})$ can be constructed in $O(n \log \sigma)$ time with $O(zt + \frac{n}{\sqrt{t}})$ working space such that subsequent $\mathsf{LCE}(i,j)$ query for any $1 \le i, j \le n$ can be answered in $O(1)$ time.*

We can also obtain the following variants of our LCE data structure.

▶ **Corollary 13.** *For any integer $1 \le t \le n$, an LCE data structure of size $O(z^{\frac{1}{3}} n^{\frac{2}{3}})$ can be constructed in $O(n \log \sigma \log n)$ time with $O(z^{\frac{1}{3}} n^{\frac{2}{3}})$ working space such that subsequent $\mathsf{LCE}(i,j)$ query for any $1 \le i, j \le n$ can be answered in $O(1)$ time.*

Proof. The LCE data structure of Theorem 12 for $t = (\frac{n}{z})^{\frac{2}{3}} < n$ takes $O(z^{\frac{1}{3}} n^{\frac{2}{3}})$ space. Since we do not compute z, we are not able to compute the exact value of $(\frac{n}{z})^{\frac{2}{3}}$. However, we can find the value of t for which the difference between the actual size of t-$\mathsf{TST}(w)$ and $\lceil \frac{n}{\sqrt{t}} \rceil$ is smallest, by doubling-then-binary searches for t. Since the size of the resulting LCE data structure can be by a constant factor larger than the smallest variant of our LCE data structure, it is clearly bounded by $O(z^{\frac{1}{3}} n^{\frac{2}{3}})$. The above method takes $O(n \log \sigma \log n)$ total time and uses $O(z^{\frac{1}{3}} n^{\frac{2}{3}})$ total working space. ◀

▶ **Corollary 14.** *For alphabets of size $\sigma \leq 2^{o(\log n)}$, an LCE data structure of size $o(n \log n)$* bits *can be constructed in $o(n \log^2 n)$ time with $o(n \log n)$* bits *of working space such that subsequent $\mathsf{LCE}(i, j)$ query for any $1 \leq i, j \leq n$ can be answered in $O(1)$ time.*

Proof. By plugging the well-known fact that $z = O(n/\log_\sigma n)$ into the result of Corollary 13, we get $O(n/(\log_\sigma n)^{\frac{1}{3}})$ for the space bound. Thus our data structure can be stored in $\mathcal{S}(n) = O(n(\log n)^{\frac{2}{3}}(\log \sigma)^{\frac{1}{3}})$ *bits* of space in the transdichotomous word RAM [16] with machine word size $\omega = \Theta(\log n)$. Hence, for alphabets of size $\sigma \leq 2^{o(\log n)}$, we obtain an LCE data structure with the claimed bounds. ◀

We can also obtain a new time-space trade-off LCE data structure. Observe that using the data structure of Theorem 8 for $1 \leq d \leq n$, we can answer $\mathsf{ShortLCE}_t$ queries for any $1 \leq t \leq n$ in $O(\max\{1, \frac{t}{d}\})$ time. Hence the following theorem holds.

▶ **Theorem 15.** *For any integers $1 \leq t' \leq t \leq n$, a data structure of size $O(zt' + \frac{n}{\sqrt{t}} + \frac{n}{t'})$ can be constructed in $O(n \log \sigma)$ time with $O(zt + \frac{n}{\sqrt{t}} + \frac{n}{t'})$ working space such that subsequent $\mathsf{LCE}(i, j)$ query for any $1 \leq i, j \leq n$ can be answered in $O(\frac{t}{t'})$ time.*

Theorem 15 implies the following: (1) By setting $t' = t$, we obtain Theorem 12. Moreover, by choosing also $t \leftarrow (\frac{n}{z})^{2/3}$, we obtain a data structure of size $O(z^{1/3}n^{2/3})$ answering LCE queries in constant time, which coincides with Corollary 13. This is the smallest data structure among the fastest data structures with two parameters t and t'. (2) By setting $t' = \sqrt{t}$ and for $t = n/z$, we get a data structure of size $O(\sqrt{nz})$ answering LCE queries in $O(\sqrt{\frac{n}{z}})$ time. This is the fastest data structure among the smallest data structures with two parameters t and t'. Note that when we do not know z, this data structure of at most $O(\sqrt{nz})$ space can be constructed in $O(n \log \sigma \log n)$ preprocessing time and $O(\sqrt{nz})$ working space as in Corollary 13. Although the parameters cannot be arbitrarily chosen, the space-query time product obtained here is optimal with fastest construction to date.

Moreover, we can reduce the zt term in the working space of Theorem 15 to zt' by increasing the preprocessing time. The bottleneck of the working space is in sorting t-blocks, i.e., Lemma 9 or Lemma 10. Since any two t-blocks can be compared in $O(\frac{t}{t'})$ time using $O(\frac{t}{t'})$ $\mathsf{ShortLCE}_{t'}$ queries, we can get the following theorem using any suitable comparison-based sorting algorithm instead of Lemma 9 or Lemma 10.

▶ **Theorem 16.** *We can construct the data structure of Theorem 15 in $O(\frac{n}{t'} \log \frac{n}{t} + n \log \sigma)$ time and $O(zt' + \frac{n}{\sqrt{t}} + \frac{n}{t'})$ working space.*

4 Lower bounds vs upper bounds for the LCE problem

Let $\mathcal{T}(n)$ and $\mathcal{S}(n)$ respectively denote the query time and data structure size (in *bits*) of an arbitrary LCE data structure for an input string of length n.

Brodal et al. [10] showed that in the non-uniform cell probe model, any RMQ data structure for a string of length n which uses $\frac{n}{t}$ bits of additional space for any $1 \leq t \leq n$ must take $\Omega(t)$ query time (i.e., $\Omega(t)$ character accesses or cell probes). Their proof assumes that each character in the string is stored in a separate cell, and counted the minimum number of character accesses required to answer an RMQ. Although their proof uses a binary string of length n where each character takes only a single bit, the above assumption is valid in a commonly accepted case that the underlying alphabet size is 2^ω, where ω denotes the size of each cell (i.e. machine word). Then, Bille et al. [9] showed that RMQ queries on any binary string of length n can be reduced to LCE queries on the same binary string, with

$\Theta(\log n)$ additional bits of space. This implies that, again assuming that each character is stored in a separate cell, any LCE data structure for a binary string of length n which uses $\mathcal{S}(n) = \frac{n}{t} + \Theta(\log n)$ additional bits of space must take $\mathcal{T}(n) = \Omega(t)$ query time, for parameter $1 \leq t \leq \frac{n}{\log n}$. Recently, Kosolobov [33] showed another result on time-space product trade-off lower bound in the non-uniform cell probe model, which can be formalized as follows:

▶ **Theorem 17** ([33]). *In the non-uniform cell probe model where each character is stored in a separate cell, for any $\mathcal{S}(n)$, there exists $\sigma = 2^{\Omega(\mathcal{S}(n)/n)}$ such that for any LCE data structure for a string over the alphabet $\Sigma = \{1, \ldots, \sigma\}$, which takes $\mathcal{S}(n)$ bits of space and answers LCE queries in $\mathcal{T}(n)$ time (i.e., with $\mathcal{T}(n)$ character accesses or cell probes), $\mathcal{T}(n)\mathcal{S}(n) = \Omega(n \log n)$ holds.*

The lower bound by Kosolobov is optimal for the considered range of the alphabet size $\sigma = 2^{\Omega(\mathcal{S}(n)/n)}$, since the data structure of Bille et al. [8] achieves $\mathcal{T}(n)\mathcal{S}(n) = O(n \log n)$.

Interestingly, our LCE data structure proposed in Section 3 reveals that there are some cases where the above lower bounds do not apply. For highly compressible strings where zt is dominated by $\frac{n}{\sqrt{t}}$, our LCE data structure of Theorem 12 takes $O(\frac{n \log n}{t})$ *bits* of space for $1 \leq t \leq n$ with machine word of size $\omega = \Theta(\log n)$. Hence, for parameter $1 \leq t' \leq \frac{\sqrt{n}}{\log n}$ we get $\mathcal{S}(n) = O(\frac{n}{t'})$. Since our data structure of Theorem 12 always achieves $\mathcal{T}(n) = O(1)$ for any parameter setting, we observe that Bille et al.'s lower bound do not apply for highly repetitive strings. Notice also that our LCE data structure of Corollary 14 achieves $\mathcal{T}(n)\mathcal{S}(n) = o(n \log n)$ for alphabet size $\sigma \leq 2^{o(\log n)}$. This shows that the alphabet size $\sigma = 2^{\Omega(\mathcal{S}(n)/n)}$ is important for Kosolobov's lower bound to hold.

Kosolobov [33] did suggest a possibility to overcome his lower bound when σ is small, and the input string can be *packed*, where $\log_\sigma n$ characters can occupy a memory cell, allowing the algorithm to read $\log_\sigma n$ characters with one memory access. We show below that this is also possible. An input string of length n can be considered as a bit string of length $n \log \sigma$. Let $t = \log n$, and first consider the $\mathsf{ShortLCE}_{\log n}$ queries on the bit string. When the original string is available in a packed representation, the longest common prefix of two substrings strings of length $\log n$ bits can be computed in constant time using no extra space using bit operations, namely, by taking the bitwise exclusive or (XOR) and computing the position of the most significant set bit (msb), or without msb, by multiple lookups on a table of total size $o(n)$ bits. Next, consider the $\mathsf{LongLCE}_{\log n}$ queries on the bit string. By simply using the same data structure as described in Section 3.3 for the bit string of length $n \log \sigma$, we can answer $\mathsf{LongLCE}_{\log n}$ queries in constant time using a data structure of size $O(\frac{n \log \sigma}{\sqrt{\log n}} \log(n \log \sigma)) = O(n\sqrt{\log n} \log \sigma)$ bits. Using the two queries, we can answer an LCE query for arbitrary positions i, j of the original string in constant time with $\lfloor (\mathsf{LCE}(i \cdot \log \sigma, j \cdot \log \sigma)) / \log \sigma \rfloor$. Since the size of the data structure is $\mathcal{S}(n) = O(n\sqrt{\log n} \log \sigma)$ bits, we obtain $\mathcal{T}(n)\mathcal{S}(n) = o(n \log n)$ for $\sigma \leq 2^{o(\sqrt{\log n})}$. Our LCE data structure based on truncated suffix trees is superior for larger σ, and also when the input string is highly repetitive and compressible since it does not require the original string.

5 Conclusions and open questions

In this paper, we presented an LCE data structure which uses $O(zt + \frac{n}{\sqrt{t}})$ words of space and answers in LCE queries in $O(1)$ time, for parameter $1 \leq t \leq \sqrt{n}$. This data structure can be constructed in $O(n \log \sigma)$ time with $O(zt + \frac{n}{\sqrt{t}})$ working space. Using the fact that

$z = O(n/\log_\sigma n)$ and suitably choosing t, our method achieves the first $O(1)$-time sub-linear space LCE data structure for alphabets of size $\sigma \leq 2^{o(\log n)}$.

An interesting open question is whether we can improve the total space requirement to $O(zt+\frac{n}{t})$. The bottleneck is the $\mathsf{LongLCE}_t$ data structure that uses $O(zt+\frac{n}{\sqrt{t}})$ space. Another open question is whether we can compute the size z of the Lempel-Ziv 77 factorization in $O(n \log \sigma)$ time with sub-linear working space. This is motivated for computing the value of t which optimizes our space bound $O(zt + \frac{n}{t})$. A little has been done in this line of research: Nishimoto et al. [40] showed how to compute the Lempel-Ziv 77 factorization in $O(n\text{polylog}(n))$ time with $O(z \log n \log^* n)$ working space. Fischer et al. [15] showed an algorithm which computes an approximation of the Lempel-Ziv 77 factorization of size $(1+\epsilon)z$ in $O(\frac{1}{\epsilon} n \log n)$ time with $O(z)$ working space, for any $0 < \epsilon \leq 1$.

Another direction of further research is to give a tighter upper bound for the size of the t-truncated suffix trees than zt. We observed that there exists a string of length n for which zt is greater by a factor of $\sqrt{n}$ than the actual size of the t-truncated suffix tree for some t.

Acknowledgments. We thank Dmitry Kosolobov for explaining his work [33] to us.

References

1 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. A new characterization of maximal repetitions by Lyndon trees. In *Proc. SODA 2015*, pages 562–571, 2015.

2 Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. *CoRR*, abs/1610.03421, 2016.

3 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *Proc. Latin 2000*, pages 88–94, 2000.

4 Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004.

5 Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. *Journal of Computer and System Sciences*, 48(2):214–230, 1994.

6 Philip Bille, Anders Roy Christiansen, Patrick Hagge Cording, and Inge Li Gørtz. Finger search in grammar-compressed strings. In *Proc. FSTTCS 2016*, pages 36:1–36:16, 2016.

7 Philip Bille, Inge Li Gørtz, Patrick Hagge Cording, Benjamin Sach, Hjalte Wedel Vildhøj, and Søren Vind. Fingerprints in compressed strings. *J. Comput. Syst. Sci.*, 86:171–180, 2017.

8 Philip Bille, Inge Li Gørtz, Mathias Bæk Tejs Knudsen, Moshe Lewenstein, and Hjalte Wedel Vildhøj. Longest common extensions in sublinear space. In *Proc. CPM 2015*, pages 65–76, 2015.

9 Philip Bille, Inge Li Gørtz, Benjamin Sach, and Hjalte Wedel Vildhøj. Time-space trade-offs for longest common extensions. *J. Discrete Algorithms*, 25:42–50, 2014.

10 Gerth Stølting Brodal, Pooya Davoodi, and S. Srinivasa Rao. On space efficient two dimensional range minimum data structures. *Algorithmica*, 63(4):815–830, 2012.

11 Gerth Stølting Brodal, Rune B. Lyngsø, Christian N. S. Pedersen, and Jens Stoye. Finding maximal pairs with bounded gap. In *Proc. CPM 1999*, pages 134–149, 1999.

12 Stefan Burkhardt and Juha Kärkkäinen. Fast lightweight suffix array construction and checking. In *Proc. CPM 2003*, pages 55–69, 2003.

13 Bastien Cazaux, Thierry Lecroq, and Eric Rivals. Construction of a de Bruijn graph for assembly from a truncated suffix tree. In *LATA 2015*, pages 109–120, 2015.

14 Maxime Crochemore, Roman Kolpakov, and Gregory Kucherov. Optimal bounds for computing α-gapped repeats. In *Proc. LATA 2016*, pages 245–255, 2016.

15 Johannes Fischer, Travis Gagie, Pawel Gawrychowski, and Tomasz Kociumaka. Approximating LZ77 via small-space multiple-pattern matching. *CoRR*, abs/1504.06647, 2015.

16 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.

17 Zvi Galil and Raffaele Giancarlo. Improved string matching with k mismatches. *ACM SIGACT News*, 17:52–54, 1986.

18 Pawel Gawrychowski, Tomohiro I, Shunsuke Inenaga, Dominik Köppl, and Florin Manea. Efficiently finding all maximal α-gapped repeats. In *Proc. STACS 2016*, pages 39:1–39:14, 2016.

19 Pawel Gawrychowski, Tomasz Kociumaka, Wojciech Rytter, and Tomasz Walen. Faster longest common extension queries in strings over general alphabets. In *Proc. CPM 2016*, pages 5:1–5:13, 2016.

20 Sara Geizhals and Dina Sokol. Finding maximal 2-dimensional palindromes. In *Proc. CPM 2016*, pages 19:1–19:12, 2016.

21 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

22 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *J. Comput. Syst. Sci.*, 69(4):525–546, 2004.

23 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

24 Tomohiro I. Longest common extensions with recompression. In *Proc. CPM 2017*, 2017. To appear.

25 Shunsuke Inenaga. A faster longest common extension algorithm on compressed strings and its applications. In *Proc. PSC 2015*, pages 1–4, 2015.

26 Juha Kärkkäinen. Repetition-based text indexes. *Ph.D. thesis, University of Helsinki, Department of Computer Science*, 1999.

27 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, 2006.

28 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Proc. CPM 2001*, pages 181–192, 2001.

29 Roman Kolpakov and Gregory Kucherov. Searching for gapped palindromes. *Theor. Comput. Sci.*, 410(51):5365–5373, 2009.

30 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *Proc. FOCS 1999*, pages 596–604, 1999.

31 Roman M. Kolpakov and Gregory Kucherov. Finding repeats with fixed gap. In *Proc. SPIRE 2000*, pages 162–168, 2000.

32 Dominik Köppl and Kunihiko Sadakane. Lempel-Ziv computation in compressed space (LZ-CICS). In *Proc. DCC 2016*, pages 3–12, 2016.

33 Dmitry Kosolobov. Tight lower bounds for the longest common extension problem. *CoRR*, abs/1611.02891, 2016.

34 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2):557–582, 1998.

35 Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theor. Comput. Sci.*, 43:239–249, 1986.

36 Mamoru Maekawa. A square root N algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst.*, 3(2):145–159, 1985.

37 Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993.

38 Joong Chae Na, Alberto Apostolico, Costas S. Iliopoulos, and Kunsoo Park. Truncated suffix trees and their application to data compression. *Theor. Comput. Sci.*, 1-3(304):87–101, 2003. doi:10.1016/S0304-3975(03)00053-7.

39 Shintaro Narisada, Diptarama, Kazuyuki Narisawa, Shunsuke Inenaga, and Ayumi Shinohara. Computing longest single-arm-gapped palindromes in a string. In *Proc. SOFSEM 2017*, pages 375–386, 2017.

40 Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Dynamic index and LZ factorization in compressed space. In *Proc. PSC 2016*, pages 158–170, 2016.

41 Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Fully dynamic data structure for LCE queries in compressed space. In *MFCS 2016*, pages 72:1–72:15, 2016.

42 Nicola Prezza. In-place longest common extensions. *CoRR*, abs/1608.05100, 2016.

43 Simon J. Puglisi and Andrew Turpin. Space-time tradeoffs for longest-common-prefix array computation. In *Proc. ISAAC 2008*, pages 124–135, 2008.

44 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003.

45 Yuka Tanimura, Tomohiro I, Hideo Bannai, Shunsuke Inenaga, Simon J. Puglisi, and Masayuki Takeda. Deterministic sub-linear space LCE data structures with efficient construction. In *Proc. CPM 2016*, pages 1:1–1:10, 2016.

46 Luciana Vitale, Alvaro Martín, and Gadiel Seroussi. Space-efficient representation of truncated suffix trees, with applications to Markov order estimation. *Theor. Comput. Sci.*, 595:34–45, 2015.

47 P. Weiner. Linear pattern-matching algorithms. In *Proc. of 14th IEEE Ann. Symp. on Switching and Automata Theory*, pages 1–11, 1973.

48 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, 1977.

ZX-Calculus: Cyclotomic Supplementarity and Incompleteness for Clifford+T Quantum Mechanics

Emmanuel Jeandel[1], Simon Perdrix[2], Renaud Vilmart[3], and Quanlong Wang[4]

1 **Université de Lorraine, CNRS, Inria, LORIA, Nancy, France**
emmanuel.jeandel@loria.fr
2 **Université de Lorraine, CNRS, Inria, LORIA, Nancy, France**
simon.perdrix@loria.fr
3 **Université de Lorraine, CNRS, Inria, LORIA, Nancy, France**
renaud.vilmart@loria.fr
4 **Dept. of Computer Science, University of Oxford, UK**
quanlong.wang@cs.ox.ac.uk

Abstract

The ZX-Calculus is a powerful graphical language for quantum mechanics and quantum information processing. The completeness of the language – i.e. the ability to derive any true equation – is a crucial question. In the quest of a complete ZX-calculus, supplementarity has been recently proved to be necessary for quantum diagram reasoning (MFCS 2016). Roughly speaking, supplementarity consists in merging two subdiagrams when they are parameterized by antipodal angles. We introduce a generalised supplementarity – called cyclotomic supplementarity – which consists in merging n subdiagrams at once, when the n angles divide the circle into equal parts. We show that when n is an odd prime number, the cyclotomic supplementarity cannot be derived, leading to a countable family of new axioms for diagrammatic quantum reasoning.

We exhibit another new simple axiom that cannot be derived from the existing rules of the ZX-Calculus, implying in particular the incompleteness of the language for the so-called Clifford+T quantum mechanics. We end up with a new axiomatisation of an extended ZX-Calculus, including an axiom schema for the cyclotomic supplementarity.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic

Keywords and phrases Quantum Computing, ZX-Calculus, Incompleteness, Clifford+T

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.11

1 Introduction

The ZX-Calculus is a powerful diagrammatic language for reasoning in quantum mechanics introduced by Coecke and Duncan [8]. Every diagram is composed of three kinds of vertices: red and green dots which are parametrised by an angle, and a yellow box; and each diagram represents a matrix thanks to the so-called standard interpretation. Moreover, any quantum transformation can be expressed using ZX-diagrams, meaning they are *universal*. For instance, some particular states can be represented as evidenced by [9]. The language initially describes pure quantum state transformations, though some work has been made to adapt it to non pure evolutions [7, 10] and measurement-based quantum computing [12].

Unlike quantum circuits, the ZX-Calculus comes with a set of equalities between diagrams that preserve the matrix that is represented. Hence, using a succession of locally applied

such equalities, one can prove that two diagrams represent the same matrix, for the language is *sound* i.e. all the equalities do indeed preserve the matrix.

The converse of soundness is called *completeness*. Here, it amounts to being able to transform any diagram into another one, as long as both represent the same matrix. Hence, the concept of completeness is here totally defined by one particular interpretation, the *standard interpretation*, unlike other definitions of completeness such as in [18] in which it is related to a whole family of interpretations.

It has been proven that the ZX-Calculus is in general not complete [17]. Yet, some restrictions have been proven to be complete. The $\frac{\pi}{2}$-fragment – the language restricted to angles that are multiples of $\frac{\pi}{2}$, which represents the *stabiliser quantum mechanics* – is complete [1], its pseudo-normal form using graph states introduced in the case of the ZX-Calculus in [11]. Moreover, this proof can be adapted to show the completeness of a ZX-like calculi used for graphically representing Spekken's toy model [4, 20] or for graphically representing the real matrices [15]. The π-fragment – representing the *real stabiliser quantum mechanics* – is also complete, with a slightly adapted set of rules [13].

A fragment is *approximately universal* when any quantum transformation can be approached with arbitrarily great precision using only the angles in the fragment. Sadly, the $\frac{\pi}{2}$-fragment is not approximately universal, but the $\frac{\pi}{4}$-fragment is [19]. It is called the *Clifford+T quantum mechanics*. Completeness for this fragment was an open question, one of the main ones in the fields of categorical quantum mechanics [6] – even though a partial answer has been given for the fragment composed of path diagrams involving angles multiple of $\frac{\pi}{4}$ [2].

In this paper, we show that in the ZX-Calculus, the $\frac{\pi}{4}$-fragment is not complete, showing that a scalar equality is derivable using matrices, but not diagrammatically. We propose to replace the "inverse rule" by this equality, and show that it can prove the former one as well as a third one: the "zero rule". Notice that this axiomatisation has been recently turned into complete axiomatisation of the ZX-calculus for this fragment [14].

We also show that an infinite number of fragments are also incomplete, by proving that a generalised form of the "supplementarity rule" [16] cannot be derived in them. Supplementarity, which has been proved to be necessary, consists in merging two subdiagrams when they are parameterized by antipodal angles. The generalised supplementarity – called cyclotomic supplementarity – consists in merging n subdiagrams at once, when the n angles divide the circle into equal parts. We show that when n is an odd prime number, the cyclotomic supplementarity cannot be derived, leading to a countable family of new axioms for diagrammatic quantum reasoning.

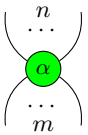
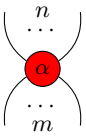
Finally, we propose to add the new scalar equation, as well as the cyclotomic supplementarity to the set of rules, and to get rid of the now obsolete "inverse" and "zero" rules. We address the question of the incompleteness of the – new – general ZX-calculus with a modified version of the proof by Zamdzhiev and Schröder de Witt [17], for theirs does not stand any more because of the introduction of the cyclotomic supplementarity.

We present the ZX-Calculus in section 2, prove the incompleteness of the $\frac{\pi}{4}$-fragment and give a new scalar rule in section 3, and in section 4 we show how to generalise the supplementarity rule, discuss the way some are derivable from others, present the altered set of rules and give a new proof of incompleteness of the general ZX-Calculus. Some proofs are omitted in this paper, you can find them at `https://hal.archives-ouvertes.fr/hal-01445707`.

2 ZX-Calculus

2.1 Diagrams and standard interpretation

A ZX-diagram $D : k \to l$ is an open graph with k inputs and l outputs – read from top to bottom – and is generated by:

$R_Z^{(n,m)}(\alpha) : n \to m$		$R_X^{(n,m)}(\alpha) : n \to m$	
$H : 1 \to 1$		$e : 0 \to 0$	
$\mathbb{I} : 1 \to 1$		$\sigma : 2 \to 2$	
$\epsilon : 2 \to 0$		$\eta : 0 \to 2$	

where $n, m \in \mathbb{N}$ and $\alpha \in \mathbb{R}$

and the two compositions:

- Spacial Composition: for any $D_1 : a \to b$ and $D_2 : c \to d$, $D_1 \otimes D_2 : a + c \to b + d$ consists in placing D_1 and D_2 side by side, D_2 on the right of D_1.
- Sequential Composition: for any $D_1 : a \to b$ and $D_2 : b \to c$, $D_2 \circ D_1 : a \to c$ consists in placing D_1 on the top of D_2, connecting the outputs of D_1 to the inputs of D_2.

The standard interpretation of the stabiliser ZX-diagrams associates to any diagram $D : n \to m$ a linear map $[\![D]\!] : \mathbb{C}^{2^n} \to \mathbb{C}^{2^m}$ inductively defined as follows:

$$[\![D_1 \otimes D_2]\!] := [\![D_1]\!] \otimes [\![D_2]\!] \qquad [\![D_2 \circ D_1]\!] := [\![D_2]\!] \circ [\![D_1]\!]$$

$$\left[\!\!\left[\; \right]\!\!\right] := (1) \qquad \left[\!\!\left[\; | \; \right]\!\!\right] := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \left[\!\!\left[\; \right]\!\!\right] := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$[\![\;]\!] := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad [\![\;]\!] := \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix} \qquad [\![\;]\!] := \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$[\![\;]\!] := \left(1 + e^{i\alpha}\right) \qquad \left[\!\!\left[\; \right]\!\!\right] := 2^m \overbrace{\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & e^{i\alpha} \end{pmatrix}}^{2^n} \quad (n + m > 0)$$

For any $n, m \geq 0$ and $\alpha \in \mathbb{R}$, $\left[\!\!\left[\; \right]\!\!\right] = \left[\!\!\left[\; \right]\!\!\right]^{\otimes m} \circ \left[\!\!\left[\; \right]\!\!\right] \circ \left[\!\!\left[\; \right]\!\!\right]^{\otimes n}$

(where $M^{\otimes 0} = (1)$ and $M^{\otimes k} = M \otimes M^{\otimes k-1}$ for any $k \in \mathbb{N}^* := \mathbb{N} \setminus \{0\}$).

To simplify, the red and green nodes will be represented empty when holding a 0 angle:

Also in order to make the diagrams a little less heavy, when n copies of the same subdiagram occur, we will use the notation $(.)^{\otimes n}$ as defined above.

With the general calculus – with angles being in $\mathbb{R}$ – we can represent any matrix of size a power of 2 i.e. ZX-Diagrams are universal:

$$\forall A \in \mathbb{C}^{2^n} \times \mathbb{C}^{2^m}, \quad \exists D, \quad [\![D]\!] = A$$

This requires dealing with an uncountable set of angles, so it is generally preferred to work with *approximate* universality – the ability to approximate any linear map with arbitrary accuracy – in which only a finite set of angles is involved. The $\frac{\pi}{4}$-fragment – ZX-diagrams where all angles are multiples of $\frac{\pi}{4}$ – is one such approximately universal fragment, whereas the $\frac{\pi}{2}$-fragment is not.

2.2 Calculus

The diagrammatic representation of a matrix is not unique in the ZX-Calculus. Hence, a set of equalities has been proposed to axiomatise the language. This set is summed up in Figure 1.

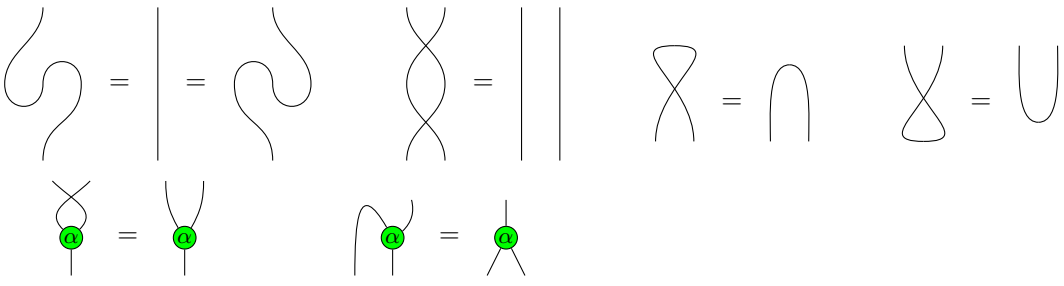
The initial set of axioms [8] included the rules (S1), (S2), (S3), (B1), (B2), (K1), (K2) and (H). The rule (EU) has been proven to be necessary in [11] and the rules (ZO) and (IV) result from [3, 5]. Finally, the rule (SUP) has been added in [16].

When we can show that a diagram D_1 is equal to another one, D_2, using a succession of equalities of this set, we write $ZX \vdash D_1 = D_2$. Given that the rules are sound, this implies that $[\![D_1]\!] = [\![D_2]\!]$. The rules can be applied to any subdiagram, meaning, for any diagram D:

$$(ZX \vdash D_1 = D_2) \quad \Rightarrow \quad \begin{cases} (ZX \vdash D_1 \circ D = D_2 \circ D) & \wedge \quad (ZX \vdash D \circ D_1 = D \circ D_2) \\ (ZX \vdash D_1 \otimes D = D_2 \otimes D) & \wedge \quad (ZX \vdash D \otimes D_1 = D \otimes D_2) \end{cases}$$

Scalars. We will identify diagrams with 0 input and 0 output – hence representing a 1×1 matrix – with scalars. We will not ignore them in this paper, while in some versions of the ZX-Calculus, the global phase or even all the scalars are ignored. Ignoring them would imply taking the risk of ignoring a zero scalar, which can lead to false statements – if $ZX \vdash 0 \times D_1 = 0 \times D_2$, we can not say that $ZX \vdash D_1 = D_2$. The first rules to palliate it appeared in [3] and were simplified in [5].

Only Topology Matters is a paradigm – to be taken as a rule – stating that any wire of a ZX-diagram can be bent at will, without changing its semantics:



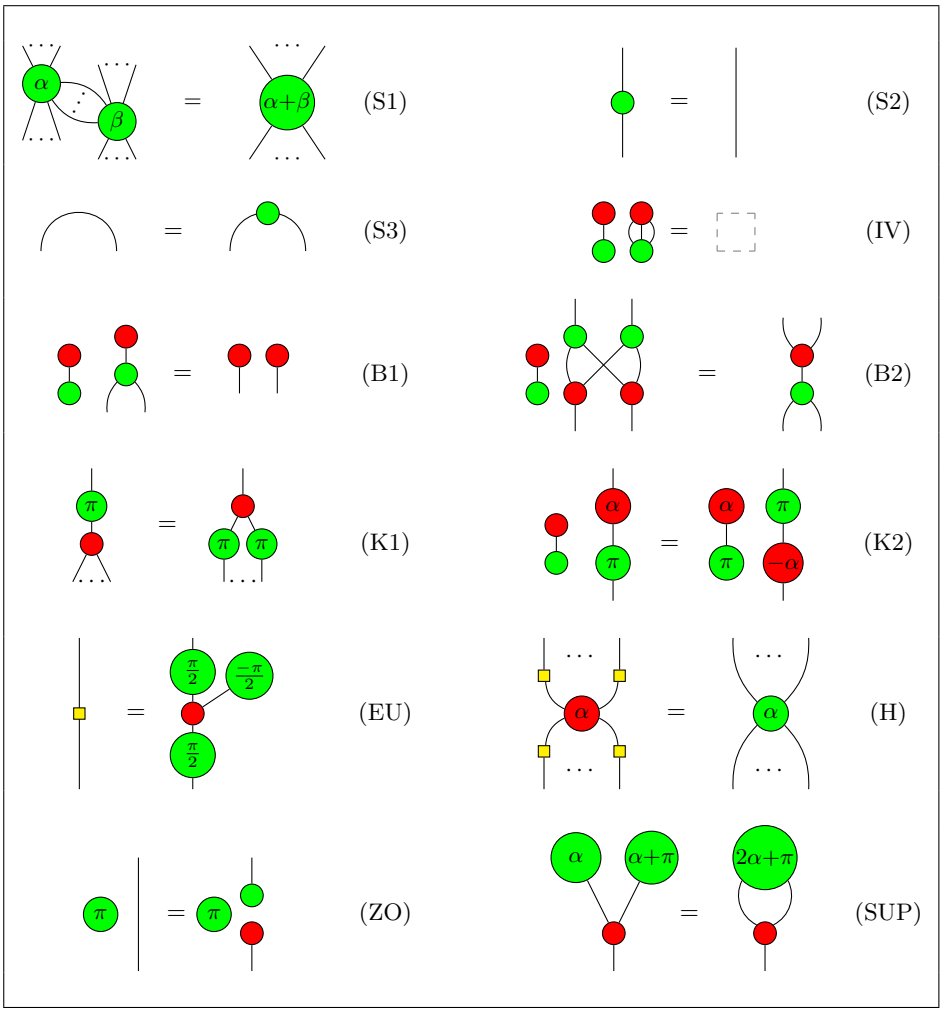
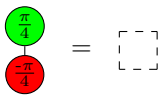


Figure 1 Rules for the ZX-calculus with scalars, augmented with the supplementarity rule [16]. All of these rules also hold when flipped upside-down, or with the colours red and green swapped. The right-hand side of (IV) is an empty diagram. $(\cdots)$ denote zero or more wires, while $(\vdots)$ denote one or more wires.

3 The $\frac{\pi}{4}$-Fragment is not Complete

In this section, we identify the following simple equation (E), which is sound – both sides of the equation represent the scalar 1 – but which cannot be derived from the rules of the ZX-Calculus expressed in Figure 1.



(E)

Since equation (E) only involves angles multiple of $\frac{\pi}{4}$, it implies the incompleteness of the $\frac{\pi}{4}$-fragment of the ZX-Calculus. In the following, we exhibit a simple invariant of the ZX-Calculus to prove that (E) is not derivable, and then we show that (E) subsumes two existing rules of the ZX-Calculus – namely (IV) and (ZO) –, leading to a simpler – (IV) and (ZO) rules are replaced by (E) – but more expressive ZX-Calculus that we call ZX_E.

3.1 A Graphical Invariant for the ZX-Calculus

We introduce a simple graphical quantity for ZX-diagrams, the parity of the number of odd-degree red dots plus the number of H-dots (yellow squares), formally defined as follows:

▶ **Definition 1.** Given a ZX-diagram $D : n \to m$, let $[\![D]\!]^{\bullet} \in \{0, 1\}$ be inductively defined as

$[\![D_1 \otimes D_2]\!]^{\bullet} = [\![D_1 \circ D_2]\!]^{\bullet} = [\![D_1]\!]^{\bullet} + [\![D_2]\!]^{\bullet} \bmod 2$, $\left[\!\!\left[\begin{array}{c} n \\ \cdots \\ \alpha \\ \cdots \\ m \end{array} \right]\!\!\right]^{\bullet} = n + m \bmod 2$, $\left[\!\!\left[\begin{array}{c} \\ \\ \blacksquare \\ \\ \\ \end{array} \right]\!\!\right]^{\bullet} = 1$,

and $[\![.]\!]^{\bullet} = 0$ for all the other generators.

One can define similarly $[\![.]\!]^{\circ}$ as the parity of the number of odd-degree green dots plus the number of H-dots. Notice that for any scalar $D : 0 \to 0$, $[\![D]\!]^{\circ} + [\![D]\!]^{\bullet} = 0 \bmod 2$, thanks to the well known degree sum formula which implies that the sum of the degree of the vertices of a graph is even. More generally, for any $D : n \to m$, $[\![D]\!]^{\circ} + [\![D]\!]^{\bullet} = n + m \bmod 2$, which is clearly an invariant of the ZX-calculus since all the rules preserve the number of inputs/outputs. As a consequence, a rule preserves $[\![.]\!]^{\circ}$ if and only if it preserves $[\![.]\!]^{\bullet}$.

▶ **Lemma 2** (Invariant). *For any ZX-diagram D_1, if $[\![D_1]\!] \neq 0$ and $ZX \vdash D_1 = D_2$, then $[\![D_1]\!]^{\bullet} = [\![D_2]\!]^{\bullet}$.*

Proof. Notice that all the rules in Figure 1, but (ZO), preserve $[\![.]\!]^{\bullet}$. Since $[\![D_1]\!] \neq 0$, the scalar π cannot appear in any derivation transforming D_1 into D_2, thus (ZO) is not applied, as a consequence $[\![D_1]\!]^{\bullet} = [\![D_2]\!]^{\bullet}$. ◀

▶ **Proposition 3.** *Equation* (E) *is not derivable using the rules in Figure 1: $ZX \nvdash$* (E).

Proof. The two diagrams of equation (E) are non zero, and they differ for $[\![.]\!]^{\bullet}$, so according to lemma 2, $ZX \nvdash$ (E). ◀

Since the diagrams of equation (E) are in the $\frac{\pi}{4}$-fragment, it implies that the $\frac{\pi}{4}$-fragment of the ZX-calculus is not complete.

▶ Remark. $\left[\!\!\left[\begin{array}{cc} \frac{\pi}{4} & \frac{\pi}{4} \\ \frac{-\pi}{4} & \frac{-\pi}{4} \end{array} \right]\!\!\right] = \left[\!\!\left[\begin{array}{c} \\ \\ \\ \end{array} \right]\!\!\right]$ the "doubled" version of (E), contrary to it, is derivable in the ZX-Calculus.

By completeness of the $\frac{\pi}{2}$-fragment, for any ZX-diagrams D_1 and D_2 in this particular fragment, if $[\![D_1]\!] = [\![D_2]\!] \neq 0$, then $[\![D_1]\!]^{\bullet} = [\![D_2]\!]^{\bullet}$. This property is obviously not true in the $\frac{\pi}{4}$-fragment, equation (E) being a counter example. However, this property is also satisfied by other, a priori not complete, fragments:

▶ **Proposition 4.** *For any $k \neq 0 \bmod 4$ and any two diagrams D_1, D_2 with angles multiple of $\frac{\pi}{k}$, if $[\![D_1]\!] = [\![D_2]\!] \neq 0$ then $[\![D_1]\!]^{\bullet} = [\![D_2]\!]^{\bullet}$.*

Proof. Given $k > 0$ and D a ZX-diagram with angles multiple of $\frac{\pi}{k}$, one can show, by induction on D, that $(\sqrt{2}^{[\![D]\!]^{\bullet}})\,[\![D]\!]$ is a matrix whose entries are in $\mathbb{Q}[e^{\frac{i\pi}{k}}]$, the smallest subfield of $\mathbb{C}$ which contains $e^{\frac{i\pi}{k}}$. Since there is a non-zero entry in $[\![D_1]\!]$, there exist $q_1, q_2 \in \mathbb{Q}[e^{\frac{i\pi}{k}}]$ such that $\sqrt{2}^{[\![D_1]\!]^{\bullet}} q_1 = \sqrt{2}^{[\![D_2]\!]^{\bullet}} q_2 \neq 0$, so $\sqrt{2}^{([\![D_1]\!]^{\bullet} - [\![D_2]\!]^{\bullet})} \in \mathbb{Q}[e^{\frac{i\pi}{k}}]$. Suppose $\sqrt{2} \in \mathbb{Q}[e^{\frac{i\pi}{k}}]$:

(i) If $k = 2 \bmod 4$, then $i = e^{\frac{i\pi}{2}} \in \mathbb{Q}[e^{\frac{i\pi}{k}}]$, so $e^{\frac{-i\pi}{4}} = \sqrt{2}\frac{1-i}{2} \in \mathbb{Q}[e^{\frac{i\pi}{k}}]$. Therefore $e^{\frac{i\pi}{2k}} = (e^{\frac{i\pi}{k}})^{\frac{k+2}{4}} \times e^{\frac{-i\pi}{4}} \in \mathbb{Q}[e^{\frac{i\pi}{k}}]$. This implies $\mathbb{Q}[e^{\frac{i\pi}{2k}}] = \mathbb{Q}[e^{\frac{i\pi}{k}}]$ which is not possible as they are vector spaces over $\mathbb{Q}$ of dimension respectively $\varphi(4k) = 2\varphi(2k)$ and $\varphi(2k)$ where φ is Euler's totient function, and $\varphi(2k) \neq 0$.

(ii) If k is odd then $2k = 2 \bmod 4$. Moreover, since $\mathbb{Q}[e^{i\frac{\pi}{k}}] \subseteq \mathbb{Q}[e^{i\frac{\pi}{2k}}]$, $\sqrt{2} \in \mathbb{Q}[e^{i\frac{\pi}{2k}}]$ which is impossible according to the previous case (i).

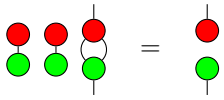
Thus $\sqrt{2} \notin \mathbb{Q}[e^{\frac{i\pi}{k}}]$ when $k \neq 0 \bmod 4$, so $[\![D_1]\!]^{\bullet} = [\![D_2]\!]^{\bullet}$. ◀

3.2 A Simpler and More Expressive ZX-calculus

Equation (E) cannot be derived in the ZX-calculus (proposition 3), as a consequence we propose to add this equation (E) as a rule of the language to make it more expressive. We show in the following that the introduction of this new rule makes the two scalar rules (ZO) and (IV) obsolete, leading to a language with less rules than the one define in Figure 1.

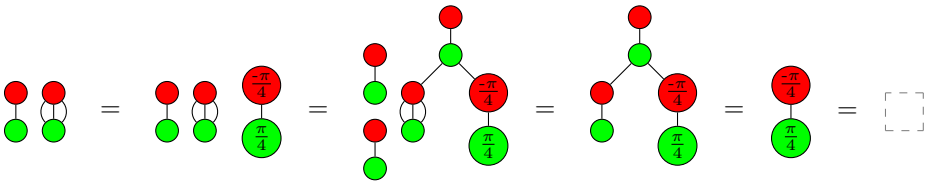
Let us define $ZX_E = \{(\text{E})\} \cup ZX \setminus \{(\text{IV}), (\text{ZO})\}$. First, notice that, thanks to [5], the so-called Hopf law is derivable from $ZX \setminus \{(\text{IV}), (\text{ZO})\}$, and hence from ZX_E:

▶ **Lemma 5.**

$ZX \setminus \{(\text{IV}), (\text{ZO})\} \vdash$  (HL)

▶ **Proposition 6.** (IV) *is derivable from* ZX_E.

Proof. Using (E), (B1), (HL), (S2) and (S1):



◀

▶ **Proposition 7.** (ZO) *is derivable from* ZX_E.

Hence $ZX_E \vdash (\text{IV}), (\text{ZO})$.

▶ Remark. The other rules of the language remain, a priori, necessary in the presence of equation (E). In particular the supplementarity which has been recently proved to be necessary in ZX [16] is necessary in ZX_E: one can prove using the interpretation $[\![.]\!]^{\sharp}_{k,l}$ – defined in [16] – with $k = 3$ and $l = 8$ that $ZX_E \setminus \{(\text{SUP})\} \nvdash (\text{SUP})$.

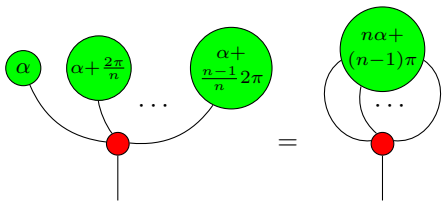
▶ Remark. One may want to generalise equation (E), replacing the particular angles $\pm\frac{\pi}{4}$ by some generic angle α. Proposition 4 is a strong evidence that such a generalisation is not possible and that the language requires at least one rule which is specific to the $\frac{\pi}{4}$ angle.

4 Cyclotomic Supplementarity

4.1 Generalisation of (SUP)

The concept of supplementarity in quantum diagram reasoning has been first introduced by Coecke and Edwards [9], turned into a simple but necessary rule (SUP in Figure 1) in [16]. Roughly speaking, supplementarity consists in merging two dots sharing the same neighbour when the difference of their angles is π, i.e. when the two angles are antipodal.

We generalize this concept to cyclotomic supplementarity as follows: for any $n \in \mathbb{N}^*$, n dots sharing the same neighbour can be merged when their angles divide the circle into equal parts (cyclotomy), i.e. when their angles are of the form $\alpha + \frac{2k\pi}{n}$ for $k \in [\![0; n-1]\!]$:



(SUP$_n$)

Notice that there are n green dots in the left diagram, and n parallel wires in the right diagram.

Any of these equations is valid for the standard interpretation of ZX-diagrams:

▶ **Proposition 8.** (SUP$_n$) *is sound.*

Cyclotomic supplementarity has a generalisation: the green dots can be merged not only when they share the same neighbour, but also when they share the same neighbourhood. It leads to the notion of cyclotomic twins, which generalise the notion of antiphase twins [13]:

▶ **Definition 9** (Cyclotomic Twins)**.** n dots in a ZX-diagram are cyclotomic twins if:

- they have the same colour
- their angles divide the circle into equal parts $\left(\alpha + \frac{2k\pi}{n} \text{ for } k \in [\![0; n-1]\!]\right)$
- they have the same neighbourhood: for any vertex, the number of wires connecting it to any of the twins is the same

▶ **Proposition 10** (Cyclotomic Twins and Supplementarity)**.** *With* $ZX \cup \{(\text{SUP}_n)\}_{n\in\mathbb{N}}$*, cyclotomic twins can be merged.*

The rest of the section is dedicated to the structures of this family of equations: we show that (SUP$_n$) is necessary when n is an odd prime number and that (SUP$_n$) can be derived when n is not prime. As a consequence, we exhibit a countable family of equations that cannot be derived in the ZX-calculus.

4.2 The Set of Supplementarity Rules for Prime Numbers

It is not necessary to define the supplementarity rules for all numbers $n \in \mathbb{N}$ as axioms. For instance, we will prove that their restriction to the set of prime numbers is enough to show all the others.

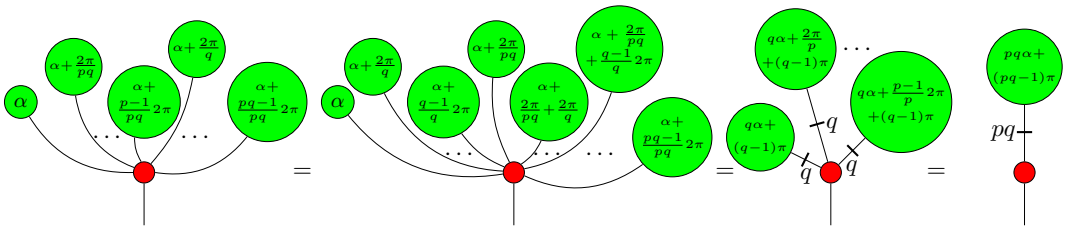
Let $\mathbb{P}$ be the set of prime numbers.

▶ **Theorem 11.**

- $\forall p, q \in \mathbb{N}^*, \quad ZX_E \cup \{(\text{SUP}_p), (\text{SUP}_q)\} \vdash (\text{SUP}_{pq})$
- $\forall n \in \mathbb{N}^*, \quad ZX_E \cup \{(\text{SUP}_p)\}_{p\in\mathbb{P}} \vdash (\text{SUP}_n)$
- $\forall p \in \mathbb{P}, p \geq 3, \quad ZX_E \cup \{(\text{SUP}_q)\}_{q\in\mathbb{P}\setminus\{p\}} \nvdash (\text{SUP}_p)$

Proof.

First statement: If n is not prime, its supplementarity can be derived. Indeed, suppose n can be decomposed in two numbers p and q ($n = pq$), for which we know the supplementarity rule.

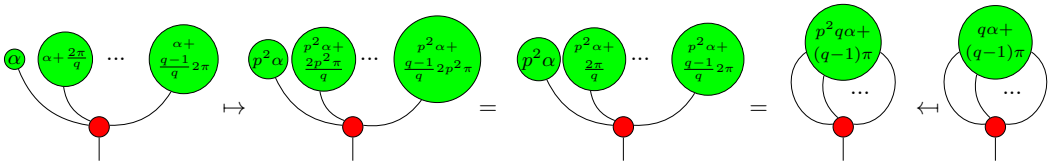


with p-ticked edge representing p parallel wires. The first equality is just a rearranging of the branches, the second uses (SUP_q) p times and the last one exploits Proposition 10 with $p(q\alpha + (q-1)\pi) + (p-1)\pi = pq\alpha + (pq-1)\pi$.

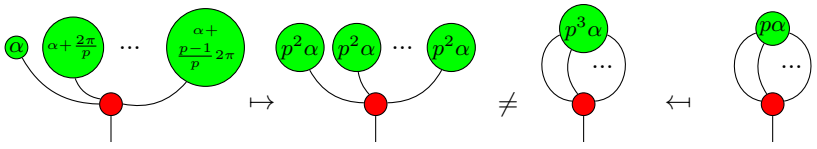
Second Statement: As a direct consequence of the previous statement, since (SUP_1) is trivial, the supplementarity rules for prime numbers are enough to derive all the others.

Third Statement: Let $p \in \mathbb{P}$ and $p \geq 3$. Let us consider the interpretation $[\![.]\!]_{p^2}$ which amounts to multiplying all the angles of a diagram by p^2.

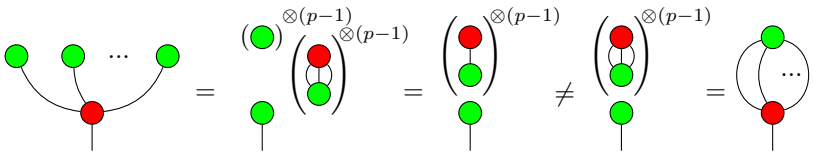
- The interpretation $[\![.]\!]_{p^2}$ coincides with the interpretation $[\![.]\!]^\sharp_{1,p^2-1}$ defined in [16]. As stated in this article, since the first parameter is odd and the second one is even, all the rules of ZX\{(SUP_2)} hold.
- The rule (E) also holds. Indeed, p is odd, and whether p mod 8 is $1, 3, 5$ or 7, p^2 mod $8 = 1$, so $p^2\frac{\pi}{4} = \frac{\pi}{4}$ mod 2π.
- The rule (SUP_q) when $q \in \mathbb{P}$, $q \neq p$ holds, since $gcd(p^2, q) = 1$:



- The rule (SUP_p) does not hold:



Indeed, when $\alpha = 0$, for instance, using on the left side $p-1$ times (B1) and (IV), and on the right side (IV) and $\frac{p-1}{2}$ times the Hopf law (HL), since $p \geq 3$:



Every rule but the p-supplementarity (with $p \in \mathbb{P}$ and $p \geq 3$) holds with this interpretation, so it cannot be derived from the others:
$\forall p \in \mathbb{P}, p \geq 3, \quad ZX_E \cup \{(\mathrm{SUP}_n)\}_{n\in\mathbb{P}} \setminus \{(\mathrm{SUP}_p)\} \nvdash (\mathrm{SUP}_p)$ ◀

▶ **Corollary 12.** *For any $n \geq 3$ odd, the $\frac{\pi}{2n}$-fragment of the ZX_E-Calculus is incomplete.*

Proof. Let p be an odd prime factor of n. Theorem 11 proves that $ZX_E \nvdash (SUP_p)$, and notice that all the angles involved in the rule are multiples of $\frac{\pi}{2p}$, hence in the $\frac{\pi}{2n}$-fragment. ◀

▶ Remark. We can also notice that all the rules (SUP_n) respect the quantity $[\![.]\!]$, so that the rule (E) remains necessary.

4.3 Discussion on the Supplementarity's Derivability Structure

Let p and q be two natural numbers. We have previously shown $ZX_E \cup \{(\text{SUP}_p), (\text{SUP}_q)\} \vdash (\text{SUP}_{pq})$. In other words, (SUP_p) can be deduced from the supplementarity of the dividers of p. Now, can we deduce this same equality from the supplementarity of some of its multiples?

The first result comes when p is odd:

▶ **Proposition 13.**

$$\forall p, q \in \mathbb{N}^*, \qquad (p = 1 \bmod 2) \quad \Rightarrow \quad \{(\text{HL}), (\text{IV}), (\text{SUP}_p), (\text{SUP}_{pq})\} \vdash (\text{SUP}_q)$$

There exists another – weaker – derivation when p is even:

▶ **Proposition 14.**

$$\forall p, q \in \mathbb{N}^*, \qquad \{(\text{HL}), (\text{IV}), (\text{SUP}_p), (\text{SUP}_{p^2q})\} \vdash (\text{SUP}_{pq})$$

▶ Remark. In the last two propositions, we require that the ZX be "general" i.e. with angles either real or a rational multiple of π because we need α/p to be in the fragment in both cases. Though, the result can be expanded to any fragment for some α provided α/p be in the fragment.

To sum up:

$$ZX_E \cup \{(\text{SUP}_p), (\text{SUP}_q)\} \vdash (\text{SUP}_{pq})$$

$$ZX_E \cup \{(\text{SUP}_p), (\text{SUP}_{p^2q})\} \vdash (\text{SUP}_{pq})$$

$$(p = 1 \bmod 2) \quad \Rightarrow \quad ZX_E \cup \{(\text{SUP}_p), (\text{SUP}_{pq})\} \vdash (\text{SUP}_q)$$

4.4 Updated Set of Rules

We propose to add the generalisation of the supplementarity rule to the set of rules of the ZX-Calculus, and to restrict to the set necessary when dealing with particular fragments. We notice that the rule (K1) is derivable from the others [5] so we can get rid of it, and the new set of rules of the ZX-Calculus is shown in figure 2.

▶ Remark. We can prove that (SUP_2) is not derivable from $ZX_E \setminus \{(\text{SUP}_2)\}$, using the interpretation $[\![.]\!]^\sharp_{k,l}$ defined in [16] with $k = 3$ and $l = 8$.

However, it is important to notice that we have not proven that (SUP_2) can not be derived from the rest once the cyclotomic supplementarity is added. Indeed, the family of interpretations used in the proof of Theorem 11 only works when p is odd, and the one used previously, $[\![.]\!]^\sharp_{k,l}$, does not hold for many supplementarity rules.

The rule (SUP_2) is all the more peculiar as, due to the Hopf law (HL), it is the only supplementarity rule that creates a non-trivial scalar – except for the supplementarity rules for even numbers, which anyway derive from (SUP_2). For instance it may create a scalar worth 0 – when applied with $\alpha = 0$ – which is the first step towards proving the rule (ZO) in the $\frac{\pi}{4}$-fragment. Moreover, it is the only supplementarity that can not be proven to be necessary by simply multiplying the angles by a constant.

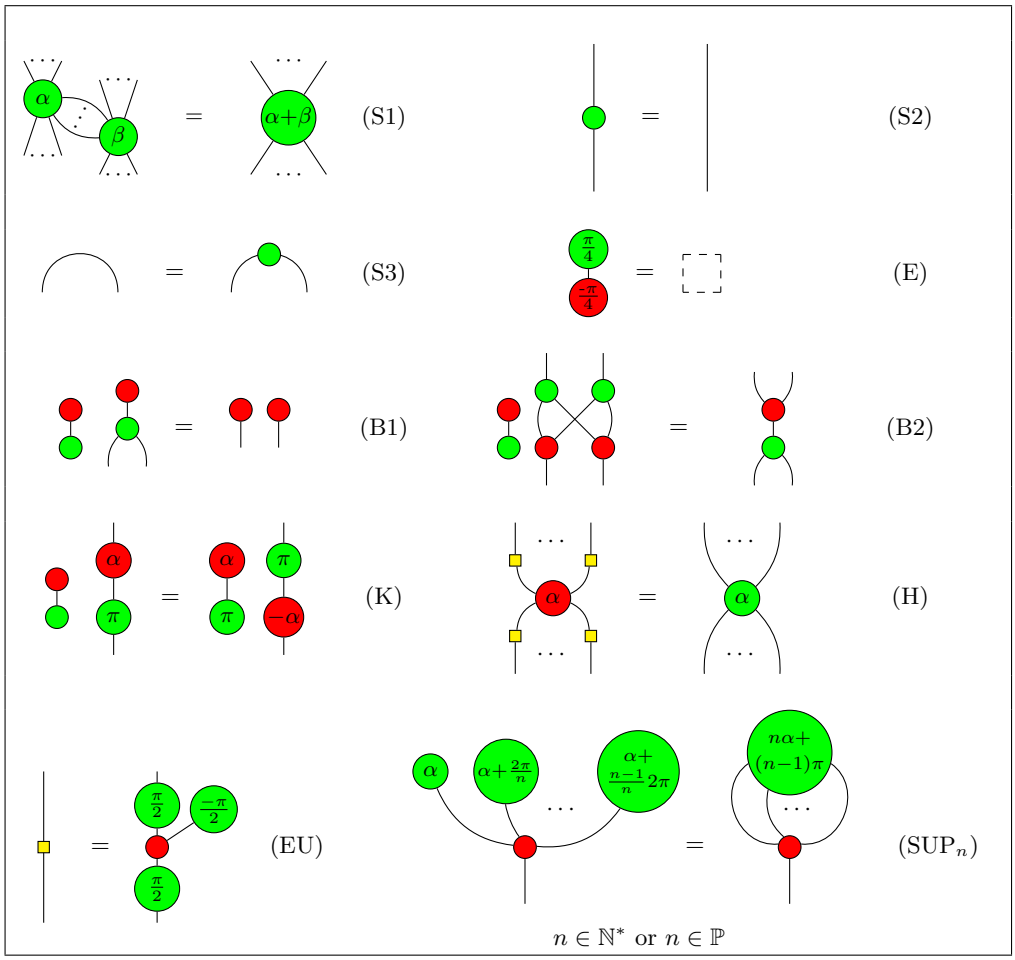


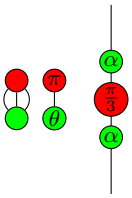
Figure 2 New set of rules for the ZX-calculus with scalars. All of these rules also hold when flipped upside-down, or with the colours red and green swapped. The right-hand side of (E) is an empty diagram. $(\cdots)$ in (S1) and (H) denote 0 or more wires, while $(\overset{\cdot}{.}\overset{\cdot}{.})$ denote 1 or more wires.

4.5 The General ZX-Calculus is still Not Complete

The argument given by Schröder de Witt and Zamdzhiev [17] to show the incompleteness of the general ZX-Calculus is not valid anymore – when multiplying the angles by any integer, there is at least one supplementarity that does not hold. But we can patch the demonstration to make it valid again.

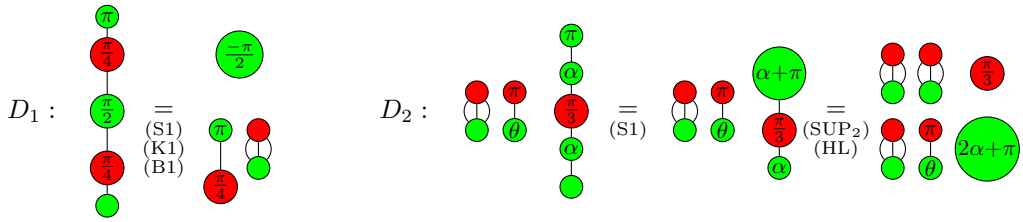
▶ **Theorem 15.** *The general ZX-Calculus is incomplete with the set of rules in figure 2.*

Proof. We will make the proof using a combination of ZX-rules and matrix calculus on the interpretations of the diagrams. Consider the following diagrams:

$D_1 :=$  and $D_2 :=$ 

We will try to express D_1 in the form D_2. One can notice that $[\![D_1]\!] = [\![D_2]\!]$ when $\alpha_0 = \frac{\pi}{2} - \arccos\left(\sqrt{\frac{2}{3}}\right)$ and $\theta_0 = \arccos\left(\frac{\sqrt{2}}{2} + \frac{\sqrt{3}}{6}\right)$. We can even show:

No more than four values for α are possible when decomposing D_1 in the form D_2: When applying the π green state at the top and the 0 green state at the bottom of both D_1 and D_2, we end up with:



In order for their interpretations to be equal, we need:

$$e^{i\frac{\pi}{4}}\sqrt{2}e^{-i\frac{\pi}{4}} = \frac{1}{2}e^{i\theta}(1+e^{i\frac{\pi}{3}})(1+e^{i(2\alpha+\pi)}) \quad \text{i.e.} \quad \frac{\sqrt{2}}{2} = e^{i(\theta+\frac{\pi}{6}+\alpha+\frac{\pi}{2})}\cos\left(\frac{\pi}{6}\right)\cos\left(\alpha+\frac{\pi}{2}\right)$$

So using the modulus, $\left|\cos\left(\alpha+\frac{\pi}{2}\right)\right| = \sqrt{\frac{2}{3}}$, thus $\alpha = \pm\frac{\pi}{2} \pm \arccos\left(\sqrt{\frac{2}{3}}\right) \bmod 2\pi$.

α is not a rational multiple of π: One can check that $e^{i\alpha_0}$ is a root of the polynomial $3X^4 + 2X^2 + 3$ which is irreducible in $\mathbb{Z}$ (since 30203 is a prime number, thanks to Cohn's irreducibility criterion, $3X^4+2X^2+3$ is irreducible in $\mathbb{Z}$). The polynomial is not cyclotomic because its coefficient of higher degree is not 1, hence $e^{i\alpha_0}$ is not a root of unity, i.e. α_0 is not a rational multiple of π. As a consequence, none of the $\pm\frac{\pi}{2} \pm \arccos\left(\sqrt{\frac{2}{3}}\right)$ are rational multiples of π.

Now, let us put back all the pieces together. Assume $ZX \vdash D_1 = D_2$ for some α and θ. Then there exists a finite sequence of rules of the ZX that transforms D_2 into D_1. We define $q \in \mathbb{N}^*$ such that for any (SUP_p) in the sequence, $p \leq q$, and $S = \{k(q+4)! + 1 \mid k \in \mathbb{N}\}$. For all $q' \in S$ and for $(\!|.|\!)_{q'}$ the interpretation that multiplies the angles by q', the rules of the ZX are preserved, $(\!|D_1|\!)_{q'} = D_1$, and $(\!|D_2|\!)_{q'}$ is in the form D_2. Indeed:

- $(q+4)!$ is clearly a multiple of 8 so $q'\frac{\pi}{4} = \frac{\pi}{4} \bmod 2\pi$, so all the rules but the supplementarity rules hold, and $(\!|D_1|\!)_{q'} = D_1$ since it is in the $\frac{\pi}{4}$-fragment.
- for any $p \in \mathbb{P}$ such that $p \leq q$, then $(q+4)! = 0 \bmod p$ so $gcd(p, q') = 1$, which implies that (SUP_p) also holds.
- $(q+4)!$ is a multiple of 6 so $q'\frac{\pi}{3} = \frac{\pi}{3} \bmod 2\pi$ hence $(\!|D_2|\!)_{q'}$ is in the form D_2.

Then, $ZX \vdash D_1 = (\!|D_2|\!)_{q'}$.

D_1 has a finite number of decompositions in the form D_2, but $\left\{(\!|D_2|\!)_{q'} \mid q' \in S\right\}$ is infinite – since α is an irrational multiple of π – and all these diagrams are decompositions of D_1 in the form D_2, hence we end up with a contradiction.
So $ZX \nvdash D_1 = D_2$, which proves the incompleteness. ◀

References

1 Miriam Backens. The zx-calculus is complete for stabilizer quantum mechanics. *New Journal of Physics*, 16(9):093021, 2014. doi:10.1088/1367-2630/16/9/093021.

2 Miriam Backens. The zx-calculus is complete for the single-qubit clifford+t group. *Electronic Proceedings in Theoretical Computer Science*, 2014. doi:10.4204/EPTCS.172.21.

3 Miriam Backens. Making the stabilizer zx-calculus complete for scalars. *Electronic Proceedings in Theoretical Computer Science*, 2015. doi:10.4204/EPTCS.195.2.

4 Miriam Backens and Ali Nabi Duman. A complete graphical calculus for spekkens' toy bit theory. *Foundations of Physics*, pages 1–34, 2014. doi:10.1007/s10701-015-9957-7.

5 Miriam Backens, Simon Perdrix, and Quanlong Wang. A simplified stabilizer zx-calculus. *Electronic Proceedings in Theoretical Computer Science*, 2016. doi:10.4204/EPTCS.236.1.

6 Categorical quantum mechanics: Zx-completeness. URL: http://cqm.wikidot.com/zx-completeness.

7 Bob Coecke. Axiomatic description of mixed states from selinger's cpm-construction. *Electron. Notes Theor. Comput. Sci.*, 210:3–13, July 2008. doi:10.1016/j.entcs.2008.04.014.

8 Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011. doi:10.1088/1367-2630/13/4/043016.

9 Bob Coecke and Bill Edwards. Three qubit entanglement within graphical z/x-calculus. *Electronic Proceedings in Theoretical Computer Science*, 52:22–33, 2011. doi:10.4204/EPTCS.52.3.

10 Bob Coecke and Simon Perdrix. Environment and classical channels in categorical quantum mechanics. *Logical Methods in Computer Science*, Volume 8, Issue 4, November 2012. doi:10.2168/LMCS-8(4:14)2012.

11 Ross Duncan and Simon Perdrix. Graphs states and the necessity of euler decomposition. *Mathematical Theory and Computational Practice*, 5635:167–177, 2009. doi:10.1007/978-3-642-03073-4.

12 Ross Duncan and Simon Perdrix. Rewriting measurement-based quantum computations with generalised flow. *Lecture Notes in Computer Science*, 6199:285–296, 2010. doi:10.1007/978-3-642-14162-1_24.

13 Ross Duncan and Simon Perdrix. Pivoting makes the zx-calculus complete for real stabilizers. *Electronic Proceedings in Theoretical Computer Science*, 2013. doi:10.4204/EPTCS.171.5.

14 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A complete axiomatisation of the zx-calculus for clifford+ t quantum mechanics. *arXiv preprint arXiv:1705.11151*, 2017.

15 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Y-calculus: A language for real matrices derived from the zx-calculus. In *Conference on Quantum Physics and Logics (QPL'17)*, 2017.

16 Simon Perdrix and Quanlong Wang. Supplementarity is necessary for quantum diagram reasoning. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 76:1–76:14, Krakow, Poland, August 2016. doi:10.4230/LIPIcs.MFCS.2016.76.

17 Christian Schröder de Witt and Vladimir Zamdzhiev. The zx-calculus is incomplete for quantum mechanics. *Electronic Proceedings in Theoretical Computer Science*, 2014. doi:10.4204/EPTCS.172.20.

18 Peter Selinger. Finite dimensional hilbert spaces are complete for dagger compact closed categories. *Logical Methods in Computer Science*, 8(4):1–12, 2012. doi:10.2168/LMCS-8(3:06)2012.

19 Peter Selinger. Quantum circuits of t-depth one. *Phys. Rev. A*, 87:042302, Apr 2013. doi:10.1103/PhysRevA.87.042302.

20 Robert Spekkens. Evidence for the epistemic view of quantum states: A toy theory. *Phys. Rev. A*, 75:032110, Mar 2007. doi:10.1103/PhysRevA.75.032110.

Counting Problems for Parikh Images

Christoph Haase[1], Stefan Kiefer[2], and Markus Lohrey[3]

1 **University of Oxford, UK**
2 **University of Oxford, UK**
3 **University of Siegen, Germany**

Abstract

Given finite-state automata (or context-free grammars) $\mathcal{A}, \mathcal{B}$ over the same alphabet and a Parikh vector $\vec{p}$, we study the complexity of deciding whether the number of words in the language of $\mathcal{A}$ with Parikh image $\vec{p}$ is greater than the number of such words in the language of $\mathcal{B}$. Recently, this problem turned out to be tightly related to the cost problem for weighted Markov chains. We classify the complexity according to whether $\mathcal{A}$ and $\mathcal{B}$ are deterministic, the size of the alphabet, and the encoding of $\vec{p}$ (binary or unary).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Parikh images, finite automata, counting problems

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.12

1 Introduction

In our recent papers [5, 7], the authors started an investigation of the so called *cost problem*: Given a Markov chain whose transitions are labelled with non-negative integers and which has a designated target state t, a probability threshold τ, and a Boolean combination of linear inequalities over one variable $\varphi(x)$, the cost problem asks whether the accumulated probability p_φ of paths achieving a value consistent with φ when reaching t is at least τ. It has been shown in [5] by the first two authors that the cost problem can be decided in PSPACE. In [7] the upper bound was improved to membership in the counting hierarchy CH, and the same upper bound has been shown for the related problem of computing a certain bit of the aforementioned probability p_φ. At the algorithmic core of those complexity results [5, 7] are the following two counting problems: Given a finite-state automaton $\mathcal{A}$ over a finite alphabet Σ and a Parikh vector $\vec{p}$ (i.e., a function mapping every alphabet symbol from Σ to $\mathbb{N}$), we denote by $N(\mathcal{A}, \vec{p})$ the number of words accepted by $\mathcal{A}$ whose Parikh image is $\vec{p}$. Then BitParikh is the problem of computing a certain bit of the number $N(\mathcal{A}, \vec{p})$ for a given finite-state automaton $\mathcal{A}$ and a Parikh vector $\vec{p}$. Further, PosParikh is the problem of checking whether $N(\mathcal{A}, \vec{p}) > N(\mathcal{B}, \vec{p})$ for two given automata $\mathcal{A}$ and $\mathcal{B}$ (over the same alphabet) and a Parikh vector $\vec{p}$. We proved in [7] that BitParikh and PosParikh both belong to the counting hierarchy if the input automata are deterministic and the Parikh vectors are encoded in binary, and we used these results to show that the cost problem belongs to CH.

The counting hierarchy is defined similarly to the polynomial-time hierarchy using counting quantifiers, see [1] or Section 2.3 for more details. It is contained in PSPACE and this inclusion is believed to be strict. In recent years, several numerical problems, for which only PSPACE upper bounds had been known, have been shown to be in CH. Two of the most important and fundamental such problems are PosSLP and BitSLP: PosSLP is the problem of deciding whether a given arithmetic circuit over the operations $+$, $-$ and $\times$ evaluates to a positive

Table 1 The complexity landscape of PosParikh. References to propositions proving the stated complexity bounds are in parentheses.

Parikh vector	size of Σ	DFA	NFA	CFG
unary encoding	unary	in L (10)	NL-compl. (10)	P-compl. (10)
	fixed	PL-compl.(2)		
	variable	PP-compl. (2, 8, 9)		
binary encoding	unary	in L (10)	NL-compl. (10)	DP-compl. (10)
	fixed	PosMatPow-hard, in CH (1, 2)	PSPACE-compl. (8, 9)	PEXP-compl. (8, 9)
	variable	PosSLP-hard [5], in CH (1)		

number, and BitSLP asks whether a certain bit of the computed number is equal to 1. Note that an arithmetic circuit with n gates can evaluate to a number in the order of 2^{2^n}; hence the number of output bits can be exponential and a certain bit of the output number can be specified with polynomially many bits. It has been shown in [5, Prop. 5] that the cost problem is hard for both PosSLP and PP (probabilistic polynomial time).

The tight relationship between the cost problem and counting problems for Parikh images motivates the investigation of the complexity of BitParikh and PosParikh also for other variants: Instead of a DFA, one can specify the language by an NFA or even a context-free grammar (CFG). Indeed, Kopczyński [9] recently asked about the complexity of computing the number of words with a given Parikh image accepted by a CFG. Other natural input parameters are the alphabet size (variable size, fixed size or even singleton) and the encoding of Parikh vectors (unary or binary). In this paper we carry out a detailed complexity analysis of PosParikh for the different settings. Our complexity results are collected in Table 1. In Section 6 we discuss possible extensions to BitParikh.

Interestingly, we show that PosParikh for DFA over a two-letter alphabet and Parikh vectors encoded in binary is hard for PosMatPow. The latter problem was recently introduced by Galby, Ouaknine and Worrell [3] and asks, given a square integer matrix $M \in \mathbb{Z}^{m\times m}$, a linear function $f\colon \mathbb{Z}^{m\times m} \to \mathbb{Z}$ with integer coefficients, and a positive integer n, whether $f(M^n) \geq 0$, where all numbers in M, f and n are encoded in binary. Note that the entries of M^n are generally of size exponential in the size of n. It is shown in [3] that PosMatPow can be decided in polynomial time for fixed dimension $m = 2$. The same holds for $m = 3$ provided that M is given in unary [3]. The general PosMatPow problem is in CH; in fact, it is is reducible to PosSLP, but the complexity of PosMatPow is left open in [3]. In particular, it is not known whether PosMatPow is easier to decide than PosSLP. Our result that PosParikh is PosMatPow-hard already for a fixed-size alphabet while PosSLP-hardness seems to require an alphabet of variable size [5] could be seen as an indication that PosMatPow is easier to decide than PosSLP.

Due to space constraints, we can only sketch some proofs in the main part. Full proofs can be found in [6].

1.1 Related Work

A problem related to the problem PosParikh is the computation of the number of all words of a given length n in a language L. If n is given in unary encoding, then this problem can be solved in NC^2 for every fixed unambiguous context-free language L [2]. On the other hand, there exists a fixed context-free language $L \subseteq \Sigma^*$ (of ambiguity degree

two) such that if the function $a^n \mapsto \#(L \cap \Sigma^n)$ can be computed in polynomial time, then EXPTIME = NEXPTIME [2]. Counting the number of words of a given length encoded in unary that are accepted by a given NFA (which is part of the input in contrast to the results of [2]) is #P-complete [10, Remark 3.4]. The corresponding problem for DFA is equivalent to counting the number of paths between two nodes in a directed acyclic graph, which is the canonical #L-complete problem. Note that for a fixed alphabet and Parikh vectors encoded in unary, the computation of $N(\mathcal{A}, \vec{p})$ for an NFA (resp. DFA) $\mathcal{A}$ can be reduced to the computation of the number of words of a given length encoded in unary accepted by an NFA (resp. DFA) $\mathcal{A}'$: In that case, one can easily compute in logspace a DFA $\mathcal{A}_{\vec{p}}$ for the set of all words with Parikh image $\vec{p}$ and then construct the product automaton of $\mathcal{A}$ and $\mathcal{A}_{\vec{p}}$.

2 Preliminaries

2.1 Counting Problems for Parikh Images

Let $\Sigma = \{a_1, \ldots, a_m\}$ be a finite alphabet. A Parikh vector is vector of m non-negative integers, i.e., an element of $\mathbb{N}^m$. Let $u \in \Sigma^*$ be a word. For $a \in \Sigma$, we denote by $|u|_a$ the number of times a occurs in u. The Parikh image $\Psi(u) \in \mathbb{N}^m$ of u is the Parikh vector counting how often every alphabet symbol of Σ occurs in u, i.e., $\Psi(u) := (|u|_{a_1}, \ldots, |u|_{a_m})$. The Parikh image of a language $L \subseteq \Sigma^*$ is defined as $\Psi(L) := \{\Psi(u) : u \in L\} \subseteq \mathbb{N}^m$.

We use standard language accepting devices in this paper. A non-deterministic finite-state automaton (NFA) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, F, \Delta)$, where Q is a finite set of control states, Σ is a finite alphabet, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of final states, and $\Delta \subseteq Q \times \Sigma \times Q$ is a set of transitions. We write $p \xrightarrow{a} q$ whenever $(p, a, q) \in \Delta$. For convenience, we sometimes label transitions with words $w \in \Sigma^+$. Such a transition corresponds to a chain of transitions that are consecutively labelled with the symbols of w. We call $\mathcal{A}$ a deterministic finite-state automaton (DFA) if for all $p \in Q$ and all $a \in \Sigma$ there is at most one state $q \in Q$ with $p \xrightarrow{a} q$. Given $u = a_1 a_2 \cdots a_n \in \Sigma^*$, a run ϱ of $\mathcal{A}$ on u is a finite sequence of control states $\varrho = p_0 p_1 \cdots p_n$ such that $p_0 = q_0$ and $p_{i-1} \xrightarrow{a_i} p_i$ for all $1 \leq i \leq n$. We call ϱ accepting whenever $p_n \in F$ and define the language accepted by $\mathcal{A}$ as $L(\mathcal{A}) := \{u \in \Sigma^* : \mathcal{A} \text{ has an accepting run on } u\}$. Finally, context-free grammars (CFG) are defined as usual.

Let Σ be an alphabet of size m and $\vec{p} \in \mathbb{N}^m$ be a Parikh vector. For a language acceptor $\mathcal{A}$ (a DFA, NFA, or CFG), we denote by $N(\mathcal{A}, \vec{p})$ the number of words in $L(\mathcal{A})$ with Parikh image $\vec{p}$, i.e.,

$$N(\mathcal{A}, \vec{p}) := \#\{u \in L(\mathcal{A}) : \Psi(u) = \vec{p}\}.$$

We denote the counting function that maps $(\mathcal{A}, \vec{p})$ to $N(\mathcal{A}, \vec{p})$ by #Parikh. For complexity considerations, we have to specify

- the type of $\mathcal{A}$ (DFA, NFA, CFG),
- the encoding of (the numbers in) $\vec{p}$ (unary or binary), and
- whether the underlying alphabet is fixed or part of the input (variable).

For instance, we speak of #Parikh for DFA over a fixed alphabet and Parikh vectors encoded in binary. The same terminology is used for the following computational problems:

PosParikh

INPUT: Language acceptors $\mathcal{A}, \mathcal{B}$ over an alphabet Σ of size m and a Parikh vector $\vec{p} \in \mathbb{N}^m$.

QUESTION: Is $N(\mathcal{A}, \vec{p}) > N(\mathcal{B}, \vec{p})$?

BitParikh

INPUT: Language acceptor $\mathcal{A}$ over an alphabet Σ of size m, a Parikh vector $\vec{p} \in \mathbb{N}^m$, and a number $i \in \mathbb{N}$ encoded binary.

QUESTION: Is the i-th bit of $N(\mathcal{A}, \vec{p})$ equal to one?

Note that for a Parikh vector $\vec{p}$ encoded in binary, the number $N(\mathcal{A}, \vec{p})$ is at most doubly exponential in the input length (size of $\mathcal{A}$ plus number of bits in $\vec{p}$), and this bound can be reached. Hence, the number of bits in $N(\mathcal{A}, \vec{p})$ is at most exponential, and a certain position in the binary encoding of $N(\mathcal{A}, \vec{p})$ can be specified with polynomially many bits.

The following two results from [5, 7] are the starting point for our further investigations in this paper (see Section 2.3 below for the formal definition of the counting hierarchy):

▶ **Theorem 1** ([5, 7]). *For DFA over a variable alphabet and Parikh vectors encoded in binary, the problems* BitParikh *and* PosParikh *belong to the counting hierarchy. Moreover, the problem* PosParikh *(resp.,* BitParikh*) is* PosSLP*-hard (resp.,* BitSLP*-hard).*[1]

2.2 Graphs

A (finite directed) multi-graph is a tuple $G = (V, E, s, t)$, where V is a finite set of nodes, E is a finite set of edges, and the mapping $s\colon E \to V$ (resp., $t\colon E \to V$) assigns to each edge its source node (resp., target node). A loop is an edge $e \in E$ with $s(e) = t(e)$. A path (of length n) in G from u to v is a sequence of edges $e_1, e_2, \ldots, e_n$ such that $s(e_1) = u$, $t(e_n) = v$, and $t(e_i) = s(e_{i+1})$ for all $1 \leq i \leq n-1$. The out-degree of a node $v \in V$ is the number $\#s^{-1}(v)$ of outgoing edges of v.

An edge-weighted multi-graph is a tuple $G = (V, E, s, t, w)$, where (V, E, s, t) is a multi-graph and $w\colon E \to \mathbb{N}$ assigns a weight to every edge. We can define the ordinary multi-graph $\tilde{G}$ induced by G by replacing every edge $e \in E$ by $k = w(e)$ many edges $e_1, \ldots, e_k$ with $s(e_i) = s(e)$ and $t(e_i) = t(e)$. For $u, v \in V$ and $n \in \mathbb{N}$, define $N(G, u, v, n)$ as the number of paths in $\tilde{G}$ from u to v of length n. Note that the different edges $e_1, \ldots, e_k$ that replaced an edge e with $w(e) = k$ are distinguished in paths.

2.3 Computational Complexity

We assume familiarity with basic complexity classes such as L (deterministic logspace), NL, P, NP, PH (the polynomial time hierarchy) and PSPACE. The class DP is the class of all intersections $K \cap L$ with $K \in \mathsf{NP}$ and $L \in \mathsf{coNP}$. Hardness for a complexity class will always refer to logspace reductions.

A counting problem is a function $f\colon \Sigma^* \to \mathbb{N}$ for a finite alphabet Σ. A counting class is a set of counting problems. A logspace reduction from a counting problem $f\colon \Sigma^* \to \mathbb{N}$ to a counting problem $g\colon \Gamma^* \to \mathbb{N}$ is a logspace computable function $h\colon \Sigma^* \to \Gamma^*$ such that for all $x \in \Sigma^*$: $f(x) = g(h(x))$. Note that no post-computation is allowed. Such reductions are also called parsimonious. Hardness for a counting class will always refer to parsimonious logspace reductions.

The counting class $\#\mathsf{P}$ contains all functions $f\colon \Sigma^* \to \mathbb{N}$ for which there exists a non-deterministic polynomial-time Turing machine M such that for every $x \in \Sigma^*$, $f(x)$ is the number of accepting computation paths of M on input x. The class PP (probabilistic polynomial time) contains all problems A for which there exists a non-deterministic polynomial-time

[1] In [5] only the PosSLP-hardness of PosParikh is explicitly shown, but the construction from [5] implies that BitParikh is BitSLP-hard.

Turing machine M such that for every input x, $x \in A$ if and only if more than half of all computation paths of M on input x are accepting. By a famous result of Toda [16], $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$, where P^{PP} is the class of all languages that can be decided in deterministic polynomial time with the help of an oracle from PP. Hence, if a problem is PP-hard, then this can be seen as a strong indication that the problem does not belong to PH (otherwise PH would collapse). If we replace in the definitions of #P and PP non-deterministic polynomial-time Turing machines by non-deterministic logspace Turing machines (resp., non-deterministic polynomial-space Turing machines; non-deterministic exponential-time Turing machines), we obtain the classes #L and PL (resp., #PSPACE and PPSPACE; #EXP and PEXP). Ladner [11] has shown that a function f belongs to #PSPACE if and only if for a given input x and a binary encoded number i the i-th bit of $f(x)$ can be computed in PSPACE. It follows that PPSPACE = PSPACE. It is well known that PP can be also defined as the class of all languages L for which there exist two #P-functions f_1 and f_2 such that $x \in L$ if and only if $f_1(x) > f_2(x)$, and similarly for PL and PEXP.

The levels of the *counting hierarchy* C_i^p ($i \geq 0$) are inductively defined as follows: $\mathsf{C}_0^p = \mathsf{P}$ and $\mathsf{C}_{i+1}^p = \mathsf{PP}^{\mathsf{C}_i^p}$ (the set of languages accepted by a PP-machine as above with an oracle from C_i^p) for all $i \geq 0$. Let $\mathsf{CH} = \bigcup_{i \geq 0} \mathsf{C}_i^p$ be the counting hierarchy. It is not difficult to show that $\mathsf{CH} \subseteq \mathsf{PSPACE}$, and most complexity theorists conjecture that $\mathsf{CH} \subsetneq \mathsf{PSPACE}$. Hence, if a problem belongs to the counting hierarchy, then the problem is probably not PSPACE-complete. More details on the counting hierarchy can be found in [1].

3 Parikh Counting Problems for DFA

Recall that PosParikh (resp., BitParikh) is PosSLP-hard (resp., BitSLP-hard), see Theorem 1. The variable alphabet and binary encoding of Parikh vectors are crucial for the proof of the lower bound. In this section, we complement Theorem 1 by showing further results for DFA when the alphabet is not unary. The results of this section are collected in the following proposition.

▶ **Proposition 2.** *For DFA, we have:*

(i) #Parikh *(resp.* PosParikh*) is* #L*-complete (resp.* PL*-complete) for a* fixed *alphabet of size at least two and Parikh vectors encoded in* unary*.*

(ii) #Parikh *(resp.* PosParikh*) is* #P*-complete (resp.* PP*-complete) for a* variable *alphabet and Parikh vectors encoded in* unary*.*

(iii) PosParikh *is* PosMatPow*-hard for a* fixed binary *alphabet and Parikh vectors encoded in* binary*.*

Proof sketch of Proposition 2(i) and (ii). We only sketch the main ideas, all details can be found in [6]. Regarding (i), the lower bound for #L follows via a reduction from the canonical #L-complete problem of computing the number of paths between two nodes in a directed acyclic graph [12], and for the PL lower bound one reduces from the problem whether the number of paths from s to t_0 is larger than the number of paths from s to t_1. For the upper bound, let $\mathcal{A}$ be a DFA over a fixed alphabet and $\vec{p}$ be a Parikh vector encoded in unary. A non-deterministic logspace machine can guess an input word for $\mathcal{A}$ symbol by symbol. Thereby, the machine only stores the current state of $\mathcal{A}$ (which needs logspace) and the binary encoding of the Parikh image of the word produced so far. The machine stops when the Parikh image reaches the input vector $\vec{p}$ and accepts iff the current state is final. Note that since the input Parikh vector $\vec{p}$ is encoded in unary notation, all numbers that appear in the accumulated Parikh image stored by the machine need only logarithmic space.

Moreover, since the alphabet has fixed size, logarithmic space suffices to store the whole Parikh image. The number of accepting computations of the machine is exactly $N(\mathcal{A}, \vec{p})$, which yields the upper bound for #L as well as for PL.

Regarding (ii), the #P-lower bound for #Parikh follows from a reduction from #3SAT, see e.g. [13, p. 442], where the unfixed alphabet allows for representing assignments of Boolean variables via individual alphabet symbols. For the #P-upper bound, let $\mathcal{A}$ be a DFA and $\vec{p}$ be a Parikh vector encoded in unary. A non-deterministic polynomial-time Turing machine can first non-deterministically produce an arbitrary word w with $\Psi(w) = \vec{p}$. Then, it checks in polynomial time whether $w \in L(\mathcal{A})$, in which case it accepts. The proof that PosParikh is PP-complete is similar and can be found in [6]. ◀

Statement (iii) is the most difficult part of Proposition 2. We split the proof into several lemmas below. As stated in Section 1, the PosMatPow problem asks, given a square integer matrix $M \in \mathbb{Z}^{m\times m}$, a linear function $f\colon \mathbb{Z}^{m\times m} \to \mathbb{Z}$ with integer coefficients, and a positive integer n, whether $f(M^n) \geq 0$. Unless stated otherwise, subsequently we assume that all numbers are encoded in binary. Here, we show that PosParikh is PosMatPow-hard for DFA over two-letter alphabets and Parikh vectors encoded in binary. We first establish several lemmas that will enable us to prove this proposition. The first lemma is a variant of the well-known correspondence between matrix powering and counting paths in a directed graph. In the following, by $M_{i,j}$ we denote the entry at position (i, j) of the matrix M.

▶ **Lemma 3.** *Given a matrix $M \in \mathbb{Z}^{m\times m}$, and $i, j \in \{1, \ldots, m\}$, one can compute in logspace an edge-weighted multi-graph $G = (V, E, s, t, w)$ and $v_i^+, v_j^+, v_j^- \in V$ such that for all $n \in \mathbb{N}$ we have $(M^n)_{i,j} = N(G, v_i^+, v_j^+, n) - N(G, v_i^+, v_j^-, n)$.*

Proof. In the following we write $M_{i,j}^n$ to mean $(M^n)_{i,j}$. Define an edge-weighted multi-graph $G = (V, E, s, t, w)$ as follows. Let $V := \{v_k^+, v_k^- : 1 \leq k \leq m\}$. For all $k, \ell \in \{1, \ldots, m\}$, if $M_{k,\ell} > 0$ then include in E an edge e from v_k^+ to v_ℓ^+ with $w(e) = M_{k,\ell}$, and an edge e from v_k^- to v_ℓ^- with $w(e) = M_{k,\ell}$. Similarly, if $M_{k,\ell} < 0$ then include in E an edge e from v_k^+ to v_ℓ^- with $w(e) = -M_{k,\ell}$, and an edge e from v_k^- to v_ℓ^+ with $w(e) = -M_{k,\ell}$. We prove by induction on n that we have for all $k, \ell \in \{1, \ldots, m\}$:

$$M_{k,\ell}^n = N(G, v_k^+, v_\ell^+, n) - N(G, v_k^+, v_\ell^-, n)$$

Note that this implies the statement of the lemma. For the induction base, let $n = 0$. If $k = \ell$ then $M_{k,\ell}^n = 1$, $N(G, v_k^+, v_\ell^+, 0) = 1$, and $N(G, v_k^+, v_\ell^-, 0) = 0$. If $k \neq \ell$ then $M_{k,\ell}^n = 0 = N(G, v_k^+, v_\ell^+, 0) = N(G, v_k^+, v_\ell^-, 0)$. For the inductive step, let $n \in \mathbb{N}$ and suppose $M_{k,\ell}^n = N(G, v_k^+, v_\ell^+, n) - N(G, v_k^+, v_\ell^-, n)$ for all k, ℓ. For $s \in \{1, \ldots, m\}$ write $I^+(s) := \{\ell \in \{1, \ldots, m\} : M_{\ell,s} > 0\}$ and $I^-(s) := \{\ell \in \{1, \ldots, m\} : M_{\ell,s} < 0\}$. For $v, v', v'' \in V$ write $\tilde{N}(G, v, v', v'', n+1)$ for the number of paths in $\tilde{G}$ (the unweighted version of G) from v to v'' of length $n+1$ such that v' is the vertex visited after n steps. We have for all $k, s \in \{1, \ldots, m\}$:

$$\begin{aligned}
M_{k,s}^{n+1} &= \sum_{\ell=1}^{m} M_{k,\ell}^n M_{\ell,s} \\
&\overset{\text{(ind. hyp.)}}{=} \sum_{\ell=1}^{m} N(G, v_k^+, v_\ell^+, n) M_{\ell,s} \; - \; \sum_{\ell=1}^{m} N(G, v_k^+, v_\ell^-, n) M_{\ell,s} \\
&= \sum_{\ell \in I^+(s)} N(G, v_k^+, v_\ell^+, n) M_{\ell,s} \; + \; \sum_{\ell \in I^-(s)} N(G, v_k^+, v_\ell^-, n)(-M_{\ell,s}) \; -
\end{aligned}$$

$$\begin{aligned}
& \sum_{\ell \in I^+(s)} N(G, v_k^+, v_\ell^-, n) M_{\ell,s} \; - \sum_{\ell \in I^-(s)} N(G, v_k^+, v_\ell^+, n)(-M_{\ell,s}) \\
= \; & \sum_{\ell \in I^+(s)} \tilde{N}(G, v_k^+, v_\ell^+, v_s^+, n+1) \; + \sum_{\ell \in I^-(s)} \tilde{N}(G, v_k^+, v_\ell^-, v_s^+, n+1) \; - \\
& \sum_{\ell \in I^+(s)} \tilde{N}(G, v_k^+, v_\ell^-, v_s^-, n+1) \; - \sum_{\ell \in I^-(s)} \tilde{N}(G, v_k^+, v_\ell^+, v_s^-, n+1) \\
= \; & N(G, v_k^+, v_s^+, n+1) - N(G, v_k^+, v_s^-, n+1)
\end{aligned}$$

This completes the induction proof. ◀

In a next step, we extend the previous lemma to matrix powering followed by the application of a linear function:

▶ **Lemma 4.** *Given a matrix $M \in \mathbb{Z}^{m \times m}$ and a linear function $f \colon \mathbb{Z}^{m \times m} \to \mathbb{Z}$ with integer coefficients, one can compute in logspace an edge-weighted multi-graph $G = (V, E, s, t, w)$ and $v_0, v^+, v^- \in V$ such that $f(M^n) = N(G, v_0, v^+, n+2) - N(G, v_0, v^-, n+2)$ for all $n \in \mathbb{N}$.*

Proof. Denote by $b_{i,j} \in \mathbb{Z}$ the coefficients of f, i.e., for $i, j \in \{1, \ldots, m\}$ let $b_{i,j} \in \mathbb{Z}$ such that for all $A \in \mathbb{Z}^{m \times m}$ we have $f(A) = \sum_{i=1}^m \sum_{j=1}^m b_{i,j} A_{i,j}$. By Lemma 3, one can compute in logspace for all $i, j \in \{1, \ldots, m\}$ an edge-weighted multi-graph $G_{i,j}$ with vertex set $V_{i,j}$, and vertices $v_{i,j}^0, v_{i,j}^+, v_{i,j}^- \in V_{i,j}$ such that for all $n \in \mathbb{N}$ we have:

$$M_{i,j}^n = N(G_{i,j}, v_{i,j}^0, v_{i,j}^+, n) - N(G_{i,j}, v_{i,j}^0, v_{i,j}^-, n) \tag{1}$$

Compute the desired edge-weighted multi-graph G as follows. For each $i, j \in \{1, \ldots, m\}$ include in G (a fresh copy of) the edge-weighted multi-graph $G_{i,j}$. Further, include in G fresh vertices v_0, v^+, v^-, and edges with weight 1 from v_0 to $v_{i,j}^0$, for each $i, j \in \{1, \ldots, m\}$. Further, for each $i, j \in \{1, \ldots, m\}$ with $b_{i,j} > 0$, include in G an edge from $v_{i,j}^+$ to v^+ with weight $b_{i,j}$, and an edge from $v_{i,j}^-$ to v^- with weight $b_{i,j}$. Similarly, for each $i, j \in \{1, \ldots, m\}$ with $b_{i,j} < 0$, include in G an edge from $v_{i,j}^+$ to v^- with weight $-b_{i,j}$, and an edge from $v_{i,j}^-$ to v^+ with weight $-b_{i,j}$. It remains to show that $f(M^n) = N(G, v_0, v^+, n+2) - N(G, v_0, v^-, n+2)$ for all $n \in \mathbb{N}$. Indeed, any path of length $n+2$ from v_0 to v^+ must start with an edge from v_0 to $v_{i,j}^0$ for some i, j, continue with a path of length n from $v_{i,j}^0$ to either $v_{i,j}^+$ or $v_{i,j}^-$, and finish with an edge to v^+. Hence, writing $I^+ := \{(i,j) : 1 \le i, j \le m,\ b_{i,j} > 0\}$ and $I^- := \{(i,j) : 1 \le i, j \le m,\ b_{i,j} < 0\}$ we have

$$N(G, v_0, v^+, n+2) = \sum_{(i,j) \in I^+} N(G, v_{i,j}^0, v_{i,j}^+, n) \cdot b_{i,j} + \sum_{(i,j) \in I^-} N(G, v_{i,j}^0, v_{i,j}^-, n) \cdot (-b_{i,j}).$$

Similarly we have:

$$N(G, v_0, v^-, n+2) = \sum_{(i,j) \in I^+} N(G, v_{i,j}^0, v_{i,j}^-, n) \cdot b_{i,j} + \sum_{(i,j) \in I^-} N(G, v_{i,j}^0, v_{i,j}^+, n) \cdot (-b_{i,j}).$$

Hence we have:

$$\begin{aligned}
f(M^n) &= \sum_{i=1}^m \sum_{j=1}^m M_{i,j}^n \cdot b_{i,j} \\
&\overset{(1)}{=} \sum_{i=1}^m \sum_{j=1}^m N(G, v_{i,j}^0, v_{i,j}^+, n) \cdot b_{i,j} \; - \sum_{i=1}^m \sum_{j=1}^m N(G, v_{i,j}^0, v_{i,j}^-, n) \cdot b_{i,j} \\
&= N(G, v_0, v^+, n+2) - N(G, v_0, v^-, n+2)
\end{aligned}$$

This proves the lemma. ◀

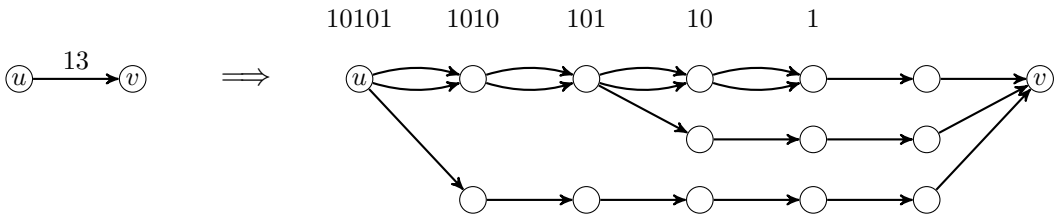


Figure 1 Illustration of the construction of the unweighted multi-graph from Lemma 5. We assume $k = 6$. The binary representation of 13 is 10101. The binary numbers over the nodes on the right hand side correspond to w-values that occur during the construction, but are not part of the output. Each binary number over a node indicates the number of paths to v.

Next, we show that one can obtain from an edge-weighted multi-graph a corresponding DFA such that the number of paths in the graph corresponds to the number of words with a certain Parikh image accepted by the DFA. The proof is split into a couple of intermediate steps.

▶ **Lemma 5.** *Given an edge-weighted multi-graph $G = (V, E, s, t, w)$ (with w in binary), $v_0, v_1 \in V$ and a number $k \in \mathbb{N}$ in unary such that $k \geq 1 + \max_{e \in E} \lfloor \log_2 w(e) \rfloor$, one can compute in logspace an unweighted multi-graph $G' := (V', E', s', t')$ with $V' \supseteq V$ such that for all $n \in \mathbb{N}$ we have $N(G, v_0, v_1, n) = N(G', v_0, v_1, n \cdot k)$.*

Proof. Note that k is at least the size of the binary representation of the largest weight in G. Define a mapping $b\colon E \to \mathbb{N}$ with $b(e) = k$ for all $e \in E$. Define G' so that it is obtained from G by iterating the following construction. Let $e \in E$ with $b(e) > 1$. If $w(e) = 1$ then replace e by a fresh path of length $b(e)$ (with $w(e') = b(e') = 1$ for all edges e' on that path). If $w(e) = 2j$ for some $j \in \mathbb{N}$ then introduce a fresh vertex v and two fresh edges e_1, e_2 from $s(e)$ to v with $b(e_1) = b(e_2) = w(e_1) = w(e_2) = 1$ and another fresh edge e_3 from v to $t(e)$ with $b(e_3) = b(e) - 1$ and $w(e_3) = j$. Finally, if $w(e) = 2j + 1$ for some $j \in \mathbb{N}$ then proceed similarly, but additionally introduce fresh vertices that create a new path of length $b(e)$ from $s(e)$ to $t(e)$ (with $w(e') = b(e') = 1$ for all edges e' on that path). By this construction, every edge e is eventually replaced by $w(e)$ paths of length k. The construction is illustrated in Figure 1.

For the logspace claim, note that it is not necessary to store the whole graph for this construction. The binary representation of k has logarithmic size and can be stored, and a copy of k can be counted down, keeping track of the b-values in the construction. The edges can be dealt with one by one. It is not necessary to store the values $w(e') = j$ for the created fresh edges; rather those values can be derived from the binary representation of the original weight $w(e)$ and the current b-value (acting as a "pointer" into the binary representation of $w(e)$). ◀

▶ **Lemma 6.** *Given an unweighted multi-graph $G = (V, E, s, t)$ and $v_0, v_1 \in V$, one can compute in logspace unweighted multi-graphs $G_0 = (V_0, E_0, s_0, t_0)$ and $G_1 = (V_1, E_1, s_1, t_1)$ with $V_0 \supseteq V$ and $V_1 \supseteq V$ such that for all $n \in \mathbb{N}$ we have $N(G_0, v_0, v_1, n+2) = N(G, v_0, v_1, n)$ and $N(G_1, v_0, v_1, n + 2) = N(G, v_0, v_1, n) + 1$.*

Proof. For G_0 redirect all edges adjacent to v_0 to a fresh vertex v_0^*, and similarly redirect all edges adjacent to v_1 to a fresh vertex v_1^*. Then add an edge from v_0 to v_0^*, and an edge from v_1^* to v_1.

For G_1 do the same, and in addition add a fresh vertex v, and add edges from v_0 to v, and from v to v_1, and a loop on v. This adds a path from v_0 to v_1 of length $n + 2$. ◀

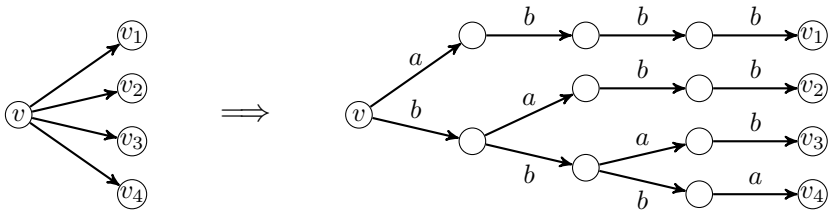


Figure 2 Illustration of the construction of the DFA from Lemma 7. We assume $d = 4$.

▶ **Lemma 7.** *Given an unweighted multi-graph $G = (V, E, s, t)$, $v_0, v_1 \in V$ and a number d in unary so that d is at least the maximal out-degree of any node in G, one can compute in logspace a DFA $\mathcal{A} = (Q, \Sigma, q_0, F, \Delta)$ with $\Sigma = \{a, b\}$ such that for all $n \in \mathbb{N}$ we have $N(G, v_0, v_1, n) = N(\mathcal{A}, \vec{p})$ where $\vec{p}(a) = n$ and $\vec{p}(b) = n \cdot (d-1)$.*

Proof. Define $\mathcal{A}$ so that $Q \supseteq V$, $q_0 = v_0$, and $F = \{v_1\}$. Include states and transitions in $\mathcal{A}$ so that for every edge e (from v to v', say) in G there is a run from v to v' in $\mathcal{A}$ of length d so that exactly one transition on this run is labelled with a, and the other $d-1$ transitions are labelled with b. Importantly, each edge e is associated to exactly one such run. The construction is illustrated in Figure 2. The DFA $\mathcal{A}$ is of quadratic size and can be computed in logspace. It follows from the construction that any path of length n in G corresponds to a run of length $n \cdot d$ in $\mathcal{A}$, with n transitions labelled with a, and $n \cdot (d-1)$ transitions labelled with b. This implies the statement of the lemma. ◀

Proof of Proposition 2(iii). The above lemmas enable us to prove part (iii) from Proposition 2. Consider an instance of PosMatPow, i.e., a square integer matrix $M \in \mathbb{Z}^{m \times m}$, a linear function $f \colon \mathbb{Z}^{m \times m} \to \mathbb{Z}$ with integer coefficients, and a positive integer n. Using Lemma 4 we can compute in logspace edge-weighted multi-graphs G_+ with vertices v_0^+, v^+ and G_- with vertices v_0^-, v^- such that

$$f(M^n) = N(G_+, v_0^+, v^+, n+2) - N(G_-, v_0^-, v^-, n+2)\,.$$

Let $k := 1 + \max_{e \in E} \lfloor \log_2 w(e) \rfloor$, where E is the union of the edge sets of G_+ and G_-. Using Lemma 5 we can compute unweighted multi-graphs G'_+, G'_- such that

$$N(G_+, v_0^+, v^+, n+2) = N(G'_+, v_0^+, v^+, (n+2) \cdot k) \text{ and}$$
$$N(G_-, v_0^-, v^-, n+2) = N(G'_-, v_0^-, v^-, (n+2) \cdot k)\,.$$

Hence,

$$f(M^n) = N(G'_+, v_0^+, v^+, (n+2) \cdot k) - N(G'_-, v_0^-, v^-, (n+2) \cdot k)\,.$$

Using Lemma 6 we can compute unweighted multi-graphs G''_+, G''_- such that

$$1 + N(G'_+, v_0^+, v^+, (n+2) \cdot k) = N(G''_+, v_0^+, v^+, (n+2) \cdot k + 2) \text{ and}$$
$$N(G'_-, v_0^-, v^-, (n+2) \cdot k) = N(G''_-, v_0^-, v^-, (n+2) \cdot k + 2)\,.$$

Hence,

$$f(M^n) + 1 = N(G''_+, v_0^+, v^+, (n+2) \cdot k + 2) - N(G''_-, v_0^-, v^-, (n+2) \cdot k + 2)\,.$$

Let d denote the maximal out-degree of any node in G''_+ or G''_-. Let $\vec{p}\colon \{a,b\} \to \mathbb{N}$ with $\vec{p}(a) = (n+2)\cdot k + 2$ and $\vec{p}(b) = ((n+2)\cdot k + 2)\cdot(d-1)$. Using Lemma 7 we can compute DFA $\mathcal{A}, \mathcal{B}$ over the alphabet $\{a,b\}$ such that

$$N(G''_+, v_0^+, v^+, (n+2)\cdot k+2) = N(\mathcal{A}, \vec{p}) \qquad \text{and} \qquad N(G''_-, v_0^-, v^-, (n+2)\cdot k+2) = N(\mathcal{B}, \vec{p})\,.$$

Hence, $f(M^n) + 1 = N(\mathcal{A}, \vec{p}) - N(\mathcal{B}, \vec{p})$. So $f(M^n) \geq 0$ if and only if $f(M^n) + 1 > 0$ if and only if $N(\mathcal{A}, \vec{p}) > N(\mathcal{B}, \vec{p})$. All mentioned computations can be performed in logspace. ◀

4 Parikh Counting Problems for NFA and CFG

In this section, we show the remaining results for NFA and CFG from Table 1 when the alphabet is not unary. The following theorem states upper bounds for PosParikh and #Parikh for NFA and CFG.

▶ **Proposition 8.** *For an alphabet of variable size, #Parikh (resp., PosParikh) is in*
(i) *#P (resp., PP) for CFG with Parikh vectors encoded in unary;*
(ii) *#PSPACE (resp., PSPACE) for NFA with Parikh vectors encoded in binary; and*
(iii) *#EXP (resp., PEXP) for CFG with Parikh vectors encoded in binary.*

Proof (sketch). In all cases, the proof is a straightforward adaption of the proof for the upper bounds in Proposition 2(i), see [6]. ◀

The following proposition states matching lower bounds for PosParikh for the cases considered in Proposition 8:

▶ **Proposition 9.** *For a fixed alphabet of size two, PosParikh is hard for*
(i) PP *for NFA and Parikh vectors encoded in* unary*;*
(ii) PSPACE *for NFA and Parikh vectors encoded in* binary*; and*
(iii) PEXP *for CFG and Parikh vectors encoded in* binary*.*

Proof (sketch). We only provide the main ideas for the lower bounds, all details can be found in [6]. Let us sketch the proof for (i). The proof is based on the fact that those strings (over an alphabet Σ) that do not encode a valid computation (called erroneous below) of a polynomial-time bounded non-deterministic Turing machine M started on an input x (with $|x| = n$) can be produced by a small NFA [15] (and this holds also for polynomial-space bounded machines, which is important for (ii)). Suppose the NFA $\mathcal{A}$ generates all words that end in an accepting configuration of M, or that are erroneous and end in a rejecting configuration. Symmetrically, suppose that $\mathcal{B}$ generates all words that are erroneous and end in an accepting configuration, or that end in a rejecting configuration. We then have that $\#(L(\mathcal{A}) \cap \Sigma^{g(n)}) - \#(L(\mathcal{B}) \cap \Sigma^{g(n)})$ equals the difference between the number of accepting paths and rejecting paths of M. Here, $g(n)$ is a suitably chosen polynomial.

Let $h\colon \Sigma^* \to \{0,1\}^*$ be the morphism that maps the i-th element of Σ (in some enumeration) to $0^{i-1}10^{\#\Sigma - i}$. Moreover, let $\mathcal{A}_h$ and $\mathcal{B}_h$ be NFA for $h(L(\mathcal{A}))$ and $h(L(\mathcal{B}))$, respectively, and let $\vec{p}$ be the Parikh vector with $\vec{p}(0) := g(n)\cdot(\#\Sigma - 1)$ and $\vec{p}(1) := g(n)$. Then $N(\mathcal{A}_h, \vec{p}) - N(\mathcal{B}_h, \vec{p}) = \#(L(\mathcal{A}) \cap \Sigma^{g(n)}) - \#(L(\mathcal{B}) \cap \Sigma^{g(n)})$ equals the difference between the number of accepting paths and rejecting paths of M.

The proof for (ii) is similar. For (iii) we use the fact that those strings that do not encode a valid computation of an exponential-time bounded non-deterministic Turing machine started on an input x can be produced by a small CFG [8]. ◀

In our construction above, we do not construct an NFA (resp., CFG) $\mathcal{A}$ and a Parikh vector $\vec{p}$ such that $N(\mathcal{A}, \vec{p})$ is *exactly* the number of accepting computations of M on the given input. This is the reason for not stating hardness for #P (resp., #PSPACE #EXP) in the above proposition (we could only show hardness under Turing reductions, but not parsimonious reductions).

5 Unary alphabets

A special case of PosParikh that has been ignored so far is that of a unary alphabet. Of course, for a unary alphabet a word is determined by its length, and a Parikh vector is a single number. Moreover, there is not much to count: Either a language $L \subseteq \{a\}^*$ contains no word of length n or exactly one word of length n. Thus, PosParikh reduces to the question whether for a given length n (encoded in unary or binary) the word a^n is accepted by $\mathcal{A}$ and rejected by $\mathcal{B}$. In this section we clarify the complexity of this problem for (i) unary DFA, NFA, and CFG, and (ii) lengths encoded in unary and binary. In the case of lengths encoded in binary, PosParikh is tightly connected to the *compressed word problem*: Given a unary DFA (resp., NFA, CFG) $\mathcal{A}$ and a number n in binary encoding, determine if $a^n \in L(\mathcal{A})$. In particular, if this problem belongs to a complexity class that is closed under complement (e.g. L, NL, P), then PosParikh belongs to the same class.

▶ **Proposition 10.** *For unary alphabets,* PosParikh *is*
(i) *in* L *for DFA with Parikh vectors encoded in* binary*;*
(ii) NL*-complete for NFA irrespective of the encoding of the Parikh vector; and*
(iii) P*-complete for CFG with Parikh vectors encoded in unary.*
(iv) DP*-complete for CFG with Parikh vectors encoded in binary.*

We only give the proof for Part (ii), all remaining proofs can be found in [6]. To this end, we employ a recent result of Sawa [14]. For $a, b \in \mathbb{N}$ we write $a + b\mathbb{N}$ for the set $\{a + b \cdot i : i \in \mathbb{N}\}$. Given a unary NFA $\mathcal{A} = (Q, \{a\}, q_0, F, \Delta)$ with $p, q \in Q$ and $n \in \mathbb{N}$ we write $p \xrightarrow{n} q$ if there is a run of length n from p to q. The subsequent lemma gives an easy criterion that allows for deciding when a word is in the language of a unary NFA.

▶ **Lemma 11** ([14, Lemma 3.1])**.** *Let $\mathcal{A} = (Q, \{a\}, q_0, F, \Delta)$ be a unary NFA with $m := |Q| \geq 2$. Let $n \geq m^2$. Then $a^n \in L(\mathcal{A})$ if and only if there are $q \in Q$, $q_f \in F$, $b \in \{1, \ldots, m\}$, and $c \in \{m^2 - b - 1, \ldots, m^2 - 2\}$ with $n \in c + b\mathbb{N}$ and $q_0 \xrightarrow{m-1} q \xrightarrow{b} q \xrightarrow{c-(m-1)} q_f$.*

We can now give the proof of Proposition 10 (ii):

Proof of Proposition 10(ii). We first show that the compressed word problem for unary NFA is in NL, from which we can conclude NL-membership for PosParikh for unary NFA with Parikh vectors encoded in binary.

Let $\mathcal{A} = (Q, \{a\}, q_0, F, \Delta)$ be the given unary NFA, and let $n \in \mathbb{N}$ be given in binary. We claim that, given two states $p_1, p_2 \in Q$ and a number $c \in \mathbb{N}$ whose binary representation is of size logarithmic in the input size, we can check in NL whether $p_1 \xrightarrow{c} p_2$ holds. To prove the claim, consider the directed graph G with vertex set $Q \times \{0, \ldots, c\}$ and an edge from (q_1, i) to (q_2, j) if and only if $q_1 \xrightarrow{1} q_2$ and $j = i + 1$. The graph G can be computed by a logspace transducer. Then $p_1 \xrightarrow{c} p_2$ holds if and only if (p_2, c) is reachable from $(p_1, 0)$ in G. The claim follows as graph reachability is in NL.

Now we give an NL algorithm for the compressed word problem. If $n < m^2$ then guess $q_f \in F$ and check, using the claim above, in NL whether $q_0 \xrightarrow{n} q_f$. If $n \geq m^2$ we use Lemma 11

as follows. We run over all $q \in Q$, $q_f \in F$, $b \in \{1, \ldots, m\}$, and $c \in \{m^2 - b - 1, \ldots, m^2 - 2\}$ (all four values can be stored in logspace), and check (i) whether $n \in c + b\mathbb{N}$ and (ii) $q_0 \xrightarrow{m-1} q \xrightarrow{b} q \xrightarrow{c-(m-1)} q_f$ holds. Condition (i) can be checked in logspace (as in the proof of Proposition 10), and condition (ii) can be checked in NL by the above claim.

It follows that PosParikh is in NL for unary NFA with Parikh vectors encoded in binary: Given NFA $\mathcal{A}, \mathcal{B}$ and $n \in \mathbb{N}$ in binary, we have $N(\mathcal{A}, n) > N(\mathcal{B}, n)$ (where we identify the mapping $\vec{p} : \{a\} \to \mathbb{N}$ with the single number $\vec{p}(a)$) if and only if $N(\mathcal{A}, n) = 1$ and $N(\mathcal{B}, n) = 0$, which holds if and only if $a^n \in L(\mathcal{A})$ and $a^n \notin L(\mathcal{B})$. Since NL is closed under complement, the latter condition can be checked in NL.

It remains to show the NL lower bound, which we obtain via a reduction from the graph reachability problem. This problem is to decide whether for a given directed graph $G = (V, E)$ and vertices $s, t \in V$ there is a path from s to t. By adding a loop at node t, this is equivalent to the existence of a path in G from s to t of length $n = \#V$. Let $\mathcal{A}$ be the NFA obtained from G by labelling every edge with the terminal symbol a and making s (resp., t) the initial (resp., unique final) state. Moreover, let $\mathcal{B}$ be an NFA with $L(\mathcal{B}) = \emptyset$. Then $N(\mathcal{A}, n) > N(\mathcal{B}, n)$ if and only if $a^n \in L(\mathcal{A})$ if and only if there is a path in G from s to t of length $n = |V|$. ◀

6 Open problems

Our PEXP-hardness proof for PosParikh on context-free languages and binary encoded Parikh vectors requires non-deterministic context-free languages. It might be interesting to see whether this problem belongs to the counting hierarchy for deterministic pushdown automata or the subclass of visibly pushdown automata. For this, one might try to generalise our techniques for DFA from [7], which rely on results from algebraic graph theory (Tutte's matrix tree theorem and the BEST theorem for counting Eulerian cycles in digraphs), to deterministic pushdown automata or visibly pushdown automata.

We believe that results similar to those shown for PosParikh can be also shown for BitParikh. For instance, the proof of Proposition 2(ii) (showing that #Parikh is #P-complete for DFA over a variable alphabet and Parikh vectors encoded in unary) shows that BitParikh for DFA over a variable alphabet and Parikh vectors encoded in unary is complete for the complexity class MP. The class MP contains all problems which can be solved in polynomial time with the additional information of one bit from a #P-function [4]. Moreover, one might also consider the problem of computing $N(\mathcal{A}, \vec{p})$ modulo a fixed number k. This should yield completeness results for Mod_k-classes.

References

1 E. Allender and K. W. Wagner. Counting hierarchies: Polynomial time and constant depth circuits. *Bulletin of the EATCS*, 40:182–194, 1990.

2 A. Bertoni, M. Goldwurm, and N. Sabadini. The complexity of computing the number of strings of given length in context-free languages. *Theor. Comput. Sci.*, 86(2):325–342, 1991.

3 E. Galby, J. Ouaknine, and J. Worrell. On matrix powering in low dimensions. In *Proc. STACS 2015*, volume 30 of *LIPIcs*, pages 329–340, 2015.

4 F. Green, J. Köbler, K. W. Regan, T. Schwentick, and J.Torán. The power of the middle bit of a #p function. *Journal of Computer and System Sciences*, 50(3):456–467, 1995.

5 C. Haase and S. Kiefer. The odds of staying on budget. In *Proc. ICALP 2015, Part II*, volume 9135 of *LNCS*, pages 234–246. Springer, 2015.

6 C. Haase, S. Kiefer, and M. Lohrey. Efficient quantile computation in markov chains via counting problems for parikh images. *CoRR*, abs/1601.04661, 2016. URL: `http://arxiv.org/abs/1601.04661`.

7 C. Haase, S. Kiefer, and M. Lohrey. Computing quantiles in Markov chains with multi-dimensional costs. In *Proc. LICS 2017*. IEEE, 2017. To appear.

8 H. B. Hunt III, D. J. Rosenkrantz, and T. G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *J. Comput. Syst. Sci.*, 12(2):222–268, 1976.

9 E. Kopczyński. Complexity of problems of commutative grammars. *Log. Meth. Comput. Sci.*, 11(1), 2015.

10 D. Kuske and M. Lohrey. First-order and counting theories of omega-automatic structures. *J. Symbolic Logic*, 73:129–150, 2008.

11 R. E. Ladner. Polynomial space counting problems. *SIAM J. Comput.*, 18(6):1087–1097, 1989.

12 M. Mahajan and V. Vinay. A combinatorial algorithm for the determinant. In *Proc. SODA 1997*, pages 730–738. ACM/SIAM, 1997.

13 C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

14 Z. Sawa. Efficient construction of semilinear representations of languages accepted by unary nondeterministic finite automata. *Fundam. Inform.*, 123(1):97–106, 2013.

15 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proc. STOC 1973*, pages 1–9. ACM, 1973.

16 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

Communication Complexity of Pairs of Graph Families with Applications*

Sudeshna Kolay[1], Fahad Panolan[2], and Saket Saurabh[3]

1 **Eindhoven University of Technology, Netherlands**
2 **Department of Informatics, University of Bergen, Norway**
3 **Department of Informatics, University of Bergen, Norway, and The Institute of Mathematical Sciences, HBNI, Chennai, India**

Abstract

Given a graph G and a pair $(\mathcal{F}_1, \mathcal{F}_2)$ of graph families, the function $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ takes as input, two induced subgraphs G_1 and G_2 of G, such that $G_1 \in \mathcal{F}_1$ and $G_2 \in \mathcal{F}_2$ and returns 1 if $V(G_1) \cap V(G_2) = \emptyset$ and 0 otherwise. We study the communication complexity of this problem in the two-party model. In particular, we look at pairs of hereditary graph families. We show that the communication complexity of this function, when the two graph families are hereditary, is sublinear if and only if there are finitely many graphs in the intersection of these two families. Then, using concepts from parameterized complexity, we obtain nuanced upper bounds on the communication complexity of $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$. A concept related to communication protocols is that of a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family of a graph G. A collection $\mathcal{F}$ of subsets of $V(G)$ is called a $(\mathcal{F}_1, \mathcal{F}_2)$-*separating family* for G, if for any two vertex disjoint induced subgraphs $G_1 \in \mathcal{F}_1, G_2 \in \mathcal{F}_2$, there is a set $F \in \mathcal{F}$ with $V(G_1) \subseteq F$ and $V(G_2) \cap F = \emptyset$. Given a graph G on n vertices, for any pair $(\mathcal{F}_1, \mathcal{F}_2)$ of hereditary graph families with sublinear communication complexity for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, we give an enumeration algorithm that finds a subexponential sized $(\mathcal{F}_1, \mathcal{F}_2)$-separating family. In fact, we give an enumeration algorithm that finds a $2^{o(k)}n^{\mathcal{O}(1)}$ sized $(\mathcal{F}_1, \mathcal{F}_2)$-separating family; where k denotes the size of a minimum sized set S of vertices such that $V(G) \setminus S$ has a bipartition (V_1, V_2) with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. We exhibit a wide range of applications for these separating families, to obtain combinatorial bounds, enumeration algorithms as well as exact and FPT algorithms for several problems.

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

Keywords and phrases Communication Complexity, Separating Family, FPT algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.13

1 Introduction

The two party communication complexity, introduced by Yao [17], is an important research area in theoretical computer science with many applications. This notion of complexity is particularly useful for proving lower bounds for VLSI computation, parallel computation, data structures as well as circuit lower bounds. In this model of communication, there are two players, Alice and Bob, holding inputs $x \in X$ and $y \in Y$ respectively, and they want to compute a given function $f : X \times Y \to \{0, 1\}$, by communicating as few bits as possible. It is assumed that both players have infinite computational power. However, communicating the results to the other player could be very costly. The minimum number of

* The research leading to these results has received funding from the European Research Council (ERC) via grants Rigorous Theory of Preprocessing, reference 267959 and PARAPPROX, reference 306992.

bits communicated, for any pair of inputs (x, y), to compute the function f, is called the (deterministic) communication complexity of f, denoted by $D(f)$. One such communication complexity problem, which has garnered a lot of attention, is the Clique vs Independent Set problem, introduced by Yannakakis [16]. For an n-vertex graph G, the Clique vs Independent Set problem is defined as follows. Alice gets a clique C in G and Bob gets an independent set I in G. Here both Alice and Bob know the graph G and their goal is to decide whether the clique and the independent set intersect in some vertex, by exchanging as few bits as possible. In other words, define the function $\mathsf{CIS}_G(C, I)$ as the cardinality of $V(C) \cap V(I)$ (note that $|V(C) \cap V(I)| \in \{0, 1\}$) and, Alice and Bob want to compute $\mathsf{CIS}_G(C, I)$. It can be shown that $D(\mathsf{CIS}_G) = \mathcal{O}(\log^2 n)$. One can also show that $D(\mathsf{CIS}_G) = \Omega(\log n)$, using the fooling set technique, a method to show communication lower bounds. Closing the gap between the upper and lower bound of CIS_G is a long standing open problem. Very recently, in 2015, Göös et al. [9] showed a near optimal lower bound of $\tilde{\Omega}(\log^2 n)$ for the problem, where $\tilde{\Omega}(m)$ hides divisors poly-logarithmic in m. Later, Göös et al. [8] showed that the same lower bound holds even for randomized communication complexity of the problem. Other versions of two party communication protocols deal with the concepts of nondeterministic and co-nondeterministic protocols. There are many works which study the cost of co-nondeterministic communication protocols of the Clique vs Independent Set problem [11, 1, 15, 7]. For more details on non-deterministic, co-nondeterministic and randomized communication complexities, [13] can be referred.

In this work, we study the communication complexity of graph properties that generalize the function CIS_G. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two hereditary graph properties. That is, $\mathcal{F}_1$ and $\mathcal{F}_2$ are two families of graphs such that if $G \in \mathcal{F}_i, i \in \{1, 2\}$, then all induced subgraphs of G are also in $\mathcal{F}_i$. We define a $(\mathcal{F}_1, \mathcal{F}_2)$ communication problem as follows. For any fixed n-vertex graph G, Alice gets an induced subgraph G_1 of G and Bob gets an induced subgraph G_2 of G, such that $G_i \in \mathcal{F}_i, i \in \{1, 2\}$, and their objective is to check whether $V(G_1)$ and $V(G_2)$ intersect, by communicating as few bits as possible. In other words, we define a function $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ as $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}(G_1, G_2) = 1$ if $V(G_1)$ and $V(G_2)$ do not intersect and 0 otherwise, where G_1 and G_2 are induced subgraphs of G and $G_i \in \mathcal{F}_i, i \in \{1, 2\}$. Alice and Bob want to find the value of the function $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ on (G_1, G_2). Notice that, when $\mathcal{F}_1$ is the family of cliques and $\mathcal{F}_2$ is the family of independent sets, then $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}(C, I) = 1$ if and only if $\mathsf{CIS}_G(C, I) = 0$. A trivial protocol for computing $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ is as follows: Alice sends a bit vector of $V(G_1)$ to Bob and Bob checks whether it intersects with the vertex set $V(G_2)$; the number of bits communicated in this protocol is n. One of our main theorems characterizes pairs of graph families for which the trivial protocol is the best one.

▶ **Theorem 1.1.** *For any two hereditary families of graphs $\mathcal{F}_1$ and $\mathcal{F}_2$, for any $n \in \mathbb{N}$, there is an n-vertex graph G such that $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}) = \Omega(n)$ if and only if $\mathcal{F}_1 \cap \mathcal{F}_2$ is an infinite family.*

We give a sketch of the proof for this Theorem. We observe that when a pair of hereditary graph families have finitely many graphs in their intersection, they have the following property: In one family, all graphs have their independence number (the maximum size of an independent set) bounded by a constant, while in the other family, all graphs have their clique number (the maximum size of a clique) bounded by some other constant. Thus, we consider the communication complexity for computing $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ when $\mathcal{F}_1$ and $\mathcal{F}_2$ are specific families. Let $\mathcal{C}_r$ be the family of graphs such that the independence number is at most r, and $\mathcal{I}_\ell$ be the family of graphs such that the clique number is at most ℓ. Such pairs of families were considered in the study made in [6]. Deriving from Theorem 5 in [10], we show that $D(\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}) = \mathcal{O}(\log^2 n)$ and, therefore, conclude the hypothesis of Theorem 1.1.

One of our main motivation, to carry out the study done in this article, was to introduce ideas from parameterized complexity in the study of communication complexity and vice versa. Parameterized complexity theory is a framework for a refined analysis of primarily hard (NP-hard) problems. Here, every input instance I of a problem Π is accompanied with an integer parameter k, and the running time is measured in terms of the associated parameter k and the input size. The main idea of parameterized algorithms is to *measure* the running time in terms of both input size as well as a parameter that captures structural properties of the input instance. Using ideas from parameterized complexity, we obtain the following nuanced upper bounds on the communication complexity of the function $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$. A pair $(\mathcal{F}_1,\mathcal{F}_2)$ of hereditary graph families where $\mathcal{F}_1 \cap \mathcal{F}_2$ is finite, will be referred to as a *good pair of graph families*.

▶ **Theorem 1.2.** *Let G be an n-vertex graph and $(\mathcal{F}_1,\mathcal{F}_2)$ be a good pair of graph families. Let $\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}$ be the size of a minimum set S of vertices such that $V(G)\setminus S$ has a bipartition (V_1,V_2) with $G[V_1]\in\mathcal{F}_1$ and $G[V_2]\in\mathcal{F}_2$. Then there is a protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ that has $\mathcal{O}(\log^2(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2})+\log n)$ communication complexity.*

For the special case of Clique vs Independent Set problem we get a protocol that has $\mathcal{O}(\log^2(\mathsf{opt}^G_{\mathcal{C}_1,\mathcal{I}_1})+\log n)$ communication complexity. We would like to mention that the protocol used to show that $D(\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell})=\mathcal{O}(\log^2 n)$ uses the full computational power of Alice and Bob. In the protocol, both players are able to compute the communication matrix of the function $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$. In contrast, we design a protocol to study communication complexity in terms of the degeneracy of graphs in the given family, where all the computations of both players are polynomial time operations. In particular, we consider the pair of families $(\mathcal{C}_1,\mathcal{D}_\ell)$, where $\mathcal{D}_\ell$ is the set of all ℓ-degenerate graphs and $\mathcal{C}_1$ is the set of all complete graphs. Note that $\mathcal{D}_0$ is the family of independent sets. Hence, this is still a generalization of the Clique vs Independent Set problem. We prove the following theorem regarding the communication complexity of $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{D}_\ell}$, with the help of a protocol where both the players only execute polynomial time computations. This will be utilized later.

▶ **Theorem 1.3.** *For any constant $\ell\in\mathbb{N}$ and an n-vertex graph G, there is a deterministic protocol that computes the function $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{D}_\ell}$ using $\mathcal{O}(\ell\log^2 n)$ bits and where both players have polynomial computational power.*

Separating Families

The main motivation for Yannakakis to introduce the Clique vs Independent Set problem was to study the number of constraints in the linear programming of a vertex packing polytope. As a spin-off of this study, he provided relations between the Clique vs Independent set problem and a CI-separating family (Clique-Independent set separating family): for a graph G, a family $\mathcal{F}$, of subsets of $V(G)$, is called a CI-separating family if for any disjoint clique C and independent set I in G, there is a set $F\in\mathcal{F}$ such that $C\subseteq F$ and $I\cap F=\emptyset$. He showed that the co-nondeterministic communication complexity of CIS_G is $\log q(G)$, where $q(G)$ is the cardinality of a CI-separating family of G. Yannakakis also provided a polynomial sized CI-separating family on comparability graphs and their complements, chordal graphs and their complements, and asked whether there is a polynomial sized family on general graphs, or even on perfect graphs. Lovász [14] extended the work of Yannakakis to t-perfect graphs and gave a polynomial sized CI-separating family on t-perfect graphs. Bousquet et al. [2] proved the existance of polynomial sized CI-separating families for the following class of graphs: random graphs, split-free graphs (here the graph does not have a fixed split graph

as an induced subgraph), graphs with no induced path P_k on k vertices nor its complement (here k is a constant), and graphs with no induced P_5. But, a result of Göös [7], that shows that the co-nondeterministic communication complexity of CIS_G is $\Omega(\log^{1.128} n)$, implies that the cardinality of CI-separating family on general graphs is super polynomial in the number of vertices.

The communication complexity of CIS_G, $D(\mathsf{CIS}_G) = \mathcal{O}(\log^2 n)$ implies that there is a CI-separating family of cardinality $n^{\mathcal{O}(\log n)}$ (See [13]). We would like to remark that the existence of a CI-separating family does not imply that such a family can be enumerated in time polynomial in the size of the family. The best known bound on the cardinality of a enumerable CI-separating family on general graphs is $\mathcal{O}(n^{\frac{\log n}{2}})$, by Hajnal (unpublished, cited in [14]). Cygan et. al. [4] also enumerated a CI-separating family, of cardinality $n^{\mathcal{O}(\log n)}$, in time $n^{\mathcal{O}(\log n)}$. In the special case of finding a CI-separating family, one can use the communication protocol of CIS_G, given in [14], to enumerate such a family in time $n^{\mathcal{O}(\log n)}$. To generalize from the definition of CI-separating families, for a graph G, and a pair of families $\mathcal{F}_1$ and $\mathcal{F}_2$, a notion of $(\mathcal{F}_1, \mathcal{F}_2)$-separating family was introduced. A family $\mathcal{P}$ of vertex subsets of $V(G)$ is called a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family if for any two disjoint vertex subsets V_1 and V_2 with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$, there is a set $A \in \mathcal{P}$ such that $V_1 \subseteq A$ and $V_2 \cap A = \emptyset$.

From an observation made in [13], it is implied that a non-deterministic protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ corresponds to a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family. This implies that if $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}) = c$, then there is a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family of size 2^c. Similar to the case of CI-separating families, describing an enumeration algorithm to find the best separating family is a problem of wide interest. In this paper, we show that a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family of size $2^{\mathcal{O}(\log^{r+\ell} n)}$ can be enumerated in time $2^{\mathcal{O}(\log^{r+\ell} n)}$. This, in turn, implies the following theorem.

▶ **Theorem 1.4.** *For any two hereditary families of graphs $\mathcal{F}_1$ and $\mathcal{F}_2$, for each integer $n > 0$, there is an n-vertex graph G such that any $(\mathcal{F}_1, \mathcal{F}_2)$-separating family must be of size $2^{\Omega(n)}$ if and only if $\mathcal{F}_1 \cap \mathcal{F}_2$ is an infinite family.*

Note that although $D(\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}) = \mathcal{O}(\log^2 n)$, we are not able to find a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family of size $2^{\mathcal{O}(\log^2 n)}$ that can be enumerated in time $2^{\mathcal{O}(\log^{r+\ell} n)}$. We also get the following theorem as a "separating family" analogue of Theorem 1.2. This theorem is extremely useful in designing parameterized algorithms.

▶ **Theorem 1.5.** *Let $(\mathcal{F}_1, \mathcal{F}_2)$ be a good pair of graph families and G be an n-vertex graph. Let S be a minimum sized vertex set of G such that $V(G) \setminus S$ has a bipartition (V_1, V_2) with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Let $|S| = \mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}$. Then a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family, for G, of cardinality $2^{\mathcal{O}(\log^c \mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2})} n^{\mathcal{O}(1)}$ can be enumerated in time $2^{\mathcal{O}(\log^c \mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2})} n^{\mathcal{O}(1)}$, where c is a constant.*

Another pair of graph properties (families of graphs) we consider is the family of complete graphs, $\mathcal{C}_1$ and that of ℓ-degenerate graphs, $\mathcal{D}_\ell$. By Theorem 1.3, we already know that $D(\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{D}_\ell})$ is $\mathcal{O}(\ell \log^2 n)$. We also give an algorithm to enumerate a $(\mathcal{C}_1, \mathcal{D}_\ell)$-separating family for an n-vertex graph, of cardinality $n^{\mathcal{O}(\ell \log n)}$, in time $n^{\mathcal{O}(\ell \log n)}$. In other words, we succeed in efficiently enumerating a $(\mathcal{C}_1, \mathcal{D}_\ell)$-separating family of size $n^{\mathcal{O}(\ell \log n)}$, the existence of which results from $D(\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{D}_\ell}) = \mathcal{O}(\ell \log^2 n)$.

Applications

In 2013, Cygan et al. [4] drew a very interesting relation between the field of enumerating separating families and designing algorithms. As mentioned earlier, a CI-separating family of cardinality $n^{\mathcal{O}(\log n)}$ is enumerated in time $n^{\mathcal{O}(\log n)}$, and this family is used to design fast exact and parameterized algorithms. They showed that Split Vertex Deletion, where we want to delete at most k vertices from a given n-vertex graph to get a split graph, can be solved in time $\mathcal{O}(1.2738^k k^{\mathcal{O}(\log k)} + n^3)$. They also showed that all induced split subgraphs of a given n-vertex graph can be listed in time $\mathcal{O}(3^{n/3} n^{\mathcal{O}(\log n)})$ time. This work motivated the last part of our study: designing exact and FPT algorithms. Not only are the enumeration algorithms for separating families interesting combinatorial questions in their own right, but they also help to design fast FPT and exact exponential time algorithms for a class of problems. A generic class of problems for which a separating family based approach works is as follows. Let $\mathcal{G}$ be a family of graphs. Then $\mathcal{G} + kv$ contains all graphs G such that there is a vertex set $S \subseteq V(G)$, of size at most k, with the property that the graph $G \setminus S \in \mathcal{G}$. Given two graph families $\mathcal{F}_1, \mathcal{F}_2$, we consider the following problem in this paper.

$(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition **Parameter:** k
Input: A graph G and a non-negative integer k
Question: Is there a vertex set $S \subseteq V(G)$, of size at most k, such that there is a partition $V_1 \uplus V_2$ of $V(G) \setminus S$ and $G[V_i] \in \mathcal{F}_i$, $i \in \{1, 2\}$?

The optimization version of $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition is denoted by $(\mathcal{F}_1, \mathcal{F}_2)$-Partition. Here, the aim is to find the minimum size of a vertex set S such that $V(G) \setminus S$ has a bipartition (V_1, V_2) with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families. For any positive integer k, let the families $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ have FPT recognition algorithms. That is, there are algorithms which take as input a graph G and an integer k, decide whether $G \in \mathcal{F}_i + kv, i \in \{1, 2\}$ and run in time $f(k)|V(G)|^{\mathcal{O}(1)}$. For ease of notation, if $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families, and the families $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ have FPT recognition algorithms, then we call $(\mathcal{F}_1, \mathcal{F}_2)$ an FPT-*good pair of families*.

▶ **Theorem 1.6.** *Let $(\mathcal{F}_1, \mathcal{F}_2)$ be an* FPT*-good pair of families. Also, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the best recognition algorithms for $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ respectively. For an n-vertex input graph and non-negative integer k, let the running time of $\mathcal{A}_i, i \in \{1, 2\}$, be $T_i(n, k)$. Then $(\mathcal{F}_1, \mathcal{F}_2)$-*p-Partition *on an instance (G, k) can be solved in time $2^{\mathcal{O}(\log^c k)} n^{\mathcal{O}(1)} \cdot \max\{T_1(n, k), T_2(n, k)\}$.*

One could obtain a result analogous to Theorem 1.6 for $(\mathcal{F}_1, \mathcal{F}_2)$-Partition. Some of the problems for which we get faster FPT and exact algorithms are (Clique,Planar)-p-Partition, (Clique,Triangle-free)-p-Partition, (Clique,Forest)-p-Partition and (Clique, Treewidth-t)-p-Partition.

2 Preliminaries

We use $\mathbb{N}$ to denote the set of natural numbers. For $n \in \mathbb{N}$, we use $[n]$ to denote $\{1, \ldots, n\}$. Through out the paper we use n to denote the number of vertices in the graph used in the context. In this paper, the function log is used to denote the logarithm function with *base* 2. We use standard notations from graph theory [5]. The vertex set and edge set of a graph are denoted as $V(G)$ and $E(G)$ respectively. The complement of the graph G is denoted by $\overline{G}$. The *neighbourhood* of a vertex v is represented as $N_G(v)$, or, when the context of the graph is clear, simply as $N(v)$. The *closed neighbourhood* of a vertex v, denoted by $N[v]$, is

subset $N(v) \cup \{v\}$.The *non-neighbourhood* of a vertex v is denoted by $\overline{N}_G(v)$. The *degree* of a vertex v, or the number of neighbours of v, is denoted by $d_G(v)$. Similarly, the *non-degree* of v, or the number of non-neighbours of v, is denoted by $\overline{d}_G(v)$. An *induced subgraph* of G on the vertex set $V' \subseteq V$ is written as $G[V']$. For a vertex subset $V' \subseteq V$, $G[V \setminus V']$ is also denoted as $G - V'$. We denote by $\omega(G)$ the size of a maximum clique in G. Similarly, $\alpha(G)$ denotes the size of a maximum independent set in G. A subgraph G' of G is denoted as $G' \leq_s G$. A complete graph on n vertices is denoted by K_n. A stable graph on n vertices is a graph G with edge set $\emptyset$, and is denoted by $\overline{K}_n$. An *empty graph* is a graph which does not have any vertices, and therefore no edges as well. Given two subgraphs $G_1, G_2 \leq_s G$, $G_1 \cap G_2$ is the induced subgraph $G[V(G_1) \cap V(G_2)]$. Similarly, $G_1 \cup G_2$ denoted the induced subgraph $G[V(G_1) \cup V(G_2)]$. For any positive integers r, ℓ, we use $R(r, \ell)$ to denote the *Ramsey number*. That is, any graph on at least $R(r, \ell)$ vertices will have either a clique of size r or an independent set of size ℓ. A family $\mathcal{F}$ of graphs is said to be *hereditary* if for any graph $G \in \mathcal{F}$, every induced subgraph of G is also contained in $\mathcal{F}$. Let $\mathcal{G}$ be a family of graphs. Then $\mathcal{G} + kv$ contains all graphs G such that there is a vertex set $S \subseteq V(G)$, of size at most k, with the property that the graph $G - S \in \mathcal{G}$.

Informally, a protocol can be thought of as a communication between two players, Alice and Bob. They have decided on some function f and wish to evaluate $f(x, y)$, for some input $x \in X$ and $y \in Y$. The catch is that x is only known to Alice and y is only known to Bob. Now we give a formal definition of a communication protocol.

▶ **Definition 2.1** ([12]). A protocol Π over a domain $X \times Y$ with range Z is a binary tree where each internal node v is labelled either by a function $a_v : X \to \{0, 1\}$ or by a function $b_v : Y \to \{0, 1\}$, and each leaf is labelled with an element $z \in Z$. The value of the protocol Π on an input (x, y) is the label of the leaf reached by starting at the root, and walking along a path in the tree. At each internal node v labelled by a_v, the walk takes left if $a_v(x) = 0$ and right if $a_v(x) = 1$, and at each internal node labelled by b_v, the walk takes left if $b_v(y) = 0$ and right if $b_v(y) = 1$. The cost of the protocol Π on an input (x, y) is the length of the path taken on the input (x, y). The cost of the protocol Π is the height of the binary tree.

▶ **Definition 2.2** ([12]). For a function $f : X \times Y \to Z$, the deterministic communication complexity of f is the minimum cost of Π, over all protocols Π that compute f. We denote the deterministic communication complexity of f by $D(f)$.

For further reading on Communication Complexity, including the concepts of non-deterministic and co-nondeterministic communication complexity of a function, we refer the reader to [12, 13]. One of the first functions, whose communication complexity was studied, is the Disjointness function. For any $x, y \in \{0, 1\}^n$, the function is defined as,

$$\mathsf{DISJ}_n(x, y) = \begin{cases} 0 & \text{if there exists } i \in [n], x[i] = y[i] = 1 \\ 1 & \text{otherwise} \end{cases}$$

▶ **Proposition 2.3** ([12]). $D(\mathit{DISJ}_n) \geq n$

We study a variant of the DISJ_n function, called the Graphic Disjointness function. Let G be a graph on n vertices and m edges. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two hereditary graph families. The following function is defined for the graph G, and the families $\mathcal{F}_1, \mathcal{F}_2$ as follows. For any two vertex subsets V_1 and V_2 such that $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$,

$$\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}(V_1, V_2) = \begin{cases} 1 & \text{if } V_1 \cap V_2 = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

A problem, related to that of computing $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, is the problem of enumerating separating families for two graph families.

▶ **Definition 2.4.** Let G be a graph on n vertices, $\mathcal{F}_1$ and $\mathcal{F}_2$ be two graph families. Suppose $\mathcal{F}$ is a family of subsets of $V(G)$ with the following property: If we take any two vertex disjoint induced subgraphs $G_1, G_2 \leq_s G$, such that $G_1 \in \mathcal{F}_1$ and $G_2 \in \mathcal{F}_2$, there is a set $F \in \mathcal{F}$ such that $V(G_1) \subseteq F$ and $V(G_2) \cap F = \emptyset$. Then $\mathcal{F}$ is called an $(\mathcal{F}_1, \mathcal{F}_2)$-separating family in G. Such a set F is called a separating set for G_1 and G_2.

▶ **Observation 2.5.** *Let G be an n-vertex graph. Let G_1, G_2 be induced subgraphs of G. Suppose for each $v \in V(G_1)$, $\overline{d}_G(v) < n/2$ and for each $w \in V(G_2)$, $d_G(w) < n/2$. Then, $V(G_1) \cap V(G_2) = \emptyset$ and $\{v \mid v \in V(G), \overline{d}_G(v) < n/2\}$ is a separating set for G_1 and G_2.*

3 Communication protocols for pairs of Hereditary graph families

To prove Theorem 1.1, we first need to prove a sublinear communication complexity bound for a specific pair of graph families. More formally, in this section we consider a pair of hereditary families of graphs, $\mathcal{C}_r = \{H \mid \alpha(H) \leq r\}$ and $\mathcal{I}_\ell = \{H \mid \omega(H) \leq \ell\}$. Here, r and ℓ are two positive integers. In this section, we consider the communication complexity of $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$. Using this, we complete the proof of Theorem 1.1. In the later half of this Section, we give upper bounds on the communication complexity of the function $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, in terms of a structural parameter of the graph G. We consider one such structural parameter and design a protocol with the help of this additional parameter.

3.1 Communication Protocol for Families of Sparse and Dense graphs

As a corollary to Theorem 5 of [10], we obtain the following Lemma:

▶ **Lemma 3.1.** *For any $r, \ell \in \mathbb{N}$, $D(GDISJ_{G,\mathcal{C}_r,\mathcal{I}_\ell}) = \mathcal{O}(R(r+1, \ell+1)\log^2 n)$.*

The protocol designed to show that $D(\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}) = \mathcal{O}(R(r+1,\ell+1)\log^2 n)$ heavily relies on the infinite computational power of both the players (See full version). In this section, we describe a protocol, with much worse communication complexity, but where both players resort to polynomial time computations only. The communication complexity of this protocol is still sublinear in $|V(G)|$. This protocol will be very useful when we design enumeration algorithms for $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating families in Section 4.

We will describe a communication protocol for $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$, with complexity $\mathcal{O}(\log^{r+\ell} n)$, for any $r, \ell \in \mathbb{N}$. Here, Alice receives an induced subgraph G_1 of G such that $G_1 \in \mathcal{C}_r$, and Bob receives an induced subgraph G_2 of G such that $G_2 \in \mathcal{I}_\ell$. They have to determine whether $V(G_1) \cap V(G_2) = \emptyset$ or not. Note that both Alice and Bob receive the graph G.

First, we give a protocol $\Pi_{1,2}$ for $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{I}_2}$, with a cost of $\mathcal{O}(\log^3 n)$. This protocol uses a protocol $\Pi_{1,1}$, for $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{I}_1}$, as a sub-protocol. As mentioned earlier, for any pair of induced subgraphs $C, I \in G$, with $C \in \mathcal{C}_1, I \in \mathcal{I}_1$, $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{I}_1}(C, I) = 1$ if and only if $\mathsf{CIS}_G(C, I) = 0$. The function CIS_G has a deterministic protocol of cost $\mathcal{O}(\log^2 n)$ [16]. Therefore, there is a protocol $\Pi_{1,1}$ of cost $\mathcal{O}(\log^2 n)$ for $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{I}_1}$. The protocol for the general case $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$ can be designed in a recursive manner that uses protocols of $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_{\ell-1}}$ and $\mathsf{GDISJ}_{G,\mathcal{C}_{r-1},\mathcal{I}_\ell}$ as subprotocols.

▶ **Lemma 3.2.** *For a graph G, there is a deterministic protocol for computing $GDISJ_{G,\mathcal{C}_1,\mathcal{I}_2}$ using $\mathcal{O}(\log^3 n)$ bits and where both players have polynomial computational power.*

Proof sketch. Let Alice get the induced subgraph G_1 and Bob get the induced subgraph G_2. The following is a protocol $\Pi_{1,2}$ that Alice and Bob will execute. Alice and Bob continue with the protocol till either they detect a vertex in the intersection of $V(G_1)$ and $V(G_2)$,

or one of G, G_1 and G_2 becomes an empty graph. The protocol is executed in top down fashion, i.e., the two players resort to a step only if the previous steps are not applicable.

1. If either G_1 or G_2 is an empty graph, then the players declare that the graphs are disjoint.
2. Alice looks for a vertex $v \in V(G_1)$ with $\overline{d}_G(v) \geq n/2$. She sends the vertex v to Bob. If $v \in V(G_2)$, then Bob lets Alice know and they terminate the protocol. Otherwise, both players delete the vertices of $\overline{N}_G(v) \cup \{v\}$ from the graph G to obtain graph $G' = G - (\overline{N}_G(v) \cup \{v\})$. Alice defines $G'_1 = G_1 - \{v\}$ while Bob defines $G'_2 = G_2 - \overline{N}_G(v)$. Then, they continue the protocol for determining whether $V(G'_1) \cap V(G'_2) = \emptyset$ in G'.
3. Bob looks for a vertex $v \in V(G_2)$ with $d_G(v) \geq n/2$. Bob sends the vertex v to Alice. If $v \in V(G_1)$, then Alice lets Bob know and they terminate the protocol. Otherwise, both players use the protocol $\Pi_{1,1}$ to compute $\mathsf{GDISJ}_{G[N_G(v)],\mathcal{C}_1,\mathcal{I}_1}(G[N_G(v) \cap V(G_1)], G[N_G(v) \cap V(G_2)])$. If the output is 0, then they declare that $V(G_1) \cap V(G_2) \neq \emptyset$ and stop. Otherwise, they delete the vertices of $N_G[v]$ from G to get $G' = G - N_G[v]$. Alice defines $G'_1 = G_1 - N_G[v]$ while Bob defines $G'_2 = G_2 - N_G[v]$. Then, they continue the protocol for determining whether $V(G'_1) \cap V(G'_2) = \emptyset$ in G'.
4. Suppose all the above steps fails, then, both players declare that $V(G_1) \cap V(G_2) = \emptyset$.

The full proof is given in the full version of this paper. ◀

▶ **Corollary 3.3.** *For any graph G, there is a deterministic protocol for $\mathsf{GDISJ}_{G,\mathcal{C}_2,\mathcal{I}_1}$ using $\mathcal{O}(\log^3 n)$ bits, where both players have polynomial computational power.*

We can give a protocol $\Pi_{r,\ell}$, for the problem $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$, of cost $\mathcal{O}(\log^{r+\ell} n)$, using a protocol for $\Pi_{r,\ell-1}$ or $\Pi_{r-1,\ell}$. We use similar arguments as in the protocol $\Pi_{1,2}$, to design the protocol $\Pi_{r,\ell}$. Thus, we can prove the following theorem using induction on $r + \ell$.

▶ **Lemma 3.4.** *For $r, \ell \in \mathbb{N}$ and graph G, there is a deterministic protocol for $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$ using $\mathcal{O}(\log^{r+\ell} n)$ bits and where both players have polynomial computational power.*

3.2 Characterization for Hereditary graph families

We are ready to prove Theorem 1.1. That is, we try to determine $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2})$ for any given pair of hereditary families $\mathcal{F}_1, \mathcal{F}_2$. If one of $\mathcal{F}_1$ or $\mathcal{F}_2$ is finite, then the number of vertices of each graph in one of the families is bounded by a constant. Thus, a trivial protocol would be for the player, who receives the bounded-sized subgraph, to send the full subgraph over to the other player, using $\mathcal{O}(\log n)$ bits. This implies, $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}) = \mathcal{O}(\log n)$. So, the interesting case is when both $\mathcal{F}_1$ and $\mathcal{F}_2$ are infinite. Now we prove Theorem 1.1.

Proof of Theorem 1.1. Without loss of generality we can assume that both $\mathcal{F}_1$ and $\mathcal{F}_2$ are infinite. Suppose the intersection family is finite. This means that there is a constant r such that a complete graph K_r, on r vertices, does not belong to the intersection family, because of finiteness. Since K_r does not belong to $\mathcal{F}_1 \cap \mathcal{F}_2$, it must not belong to at least one of the families. Let this be $\mathcal{F}_1$. Since $\mathcal{F}_1$ is hereditary, no graph in $\mathcal{F}_1$ has K_r as an induced subgraph. This implies that for any graph G in $\mathcal{F}_1$, $\omega(G) \leq r - 1$. Now we show that for any graph G in $\mathcal{F}_2$, $\alpha(G) \leq \ell - 1$ for some constant ℓ. Towards that, we first claim that $\mathcal{F}_1$ contains all stable graphs. Otherwise, since $\mathcal{F}_1$ is a hereditary family, if $\mathcal{F}_1$ does not contain a stable graph on ℓ' vertices, all graphs in $\mathcal{F}_1$ neither have a r-sized clique as an induced subgraph nor an ℓ'-sized independent set as an induced subgraph. However, by Ramsey's theorem, each graph in $\mathcal{F}_1$ has at most $R(r, \ell')$ vertices, thus contradicting the infiniteness of $\mathcal{F}_1$. So far we know that, $\mathcal{F}_1 \cap \mathcal{F}_2$ is finite and $\mathcal{F}_1$ contains all stable graphs. This implies that $\mathcal{F}_2$ does not contain all stable graphs. Let ℓ be an integer such that $\overline{K}_\ell \notin \mathcal{F}_2$. By the

hereditary property of $\mathcal{F}_2$, no graph in $\mathcal{F}_2$ contains $\overline{K}_\ell$ as an induced subgraph. That is, for all graph G in $\mathcal{F}_2$, $\alpha(G) \leq \ell - 1$. Thus, Lemma 3.1 gives us a deterministic protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, with $o(n)$ communication complexity.

For the reverse direction, suppose $\mathcal{F}_1 \cap \mathcal{F}_2$ is an infinite family. To prove $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}) = \Omega(n)$ we give a simple reduction from DISJ_n. In DISJ_n, Alice is given $x \in \{0,1\}^n$ and Bob is given $y \in \{0,1\}^n$ and they want to check whether there is an $i \in [n]$ such that $x[i] = y[i] = 1$. Now we create an instance of $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ as follows. We fix an n-vertex graph $G \in \mathcal{F}_1 \cap \mathcal{F}_2$ (such a graph exists because of hereditary property), with $V(G) = \{v_1, \ldots, v_n\}$. Let $V_x = \{v_i \in V(G) \mid i \in [n], x[i] = 1\}$ and $V_y = \{v_i \in V(G) \mid i \in [n], y[i] = 1\}$. Since $G \in \mathcal{F}_1 \cap \mathcal{F}_2$, $G[V_x] \in \mathcal{F}_1$ and $G[V_y] \in \mathcal{F}_2$. In the $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ problem, Alice gets $G[V_x]$ and Bob gets $G[V_y]$. Clearly $V_x \cap V_y \neq \emptyset$ if and only if there is an $i \in [n]$ such that $x[i] = y[i] = 1$. Hence, by Proposition 2.3, $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}) = \Omega(n)$. ◀

For the rest of this paper, a pair $(\mathcal{F}_1, \mathcal{F}_2)$ of hereditary graph families where $\mathcal{F}_1 \cap \mathcal{F}_2$ is a finite graph family, will be referred to as a *good pair of graph families*. The proof of Theorem 1.1 also gives us the following folklore corollary.

▶ **Corollary 3.5.** *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families. Then, there are constants r and ℓ, such that for any graph $G_1 \in \mathcal{F}_1$ and $G_2 \in \mathcal{F}_2$, $\omega(G_1) \leq r$ and $\alpha(G_2) \leq \ell$.*

Corollary 3.5 and Ramsey theorem leads us to another useful corollary.

▶ **Corollary 3.6.** *Let G be a graph. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families. Then there are constant r and ℓ (same as the constants mentioned in Corollary 3.5) such that, for any pair (G_1, G_2) of induced subgraphs of G, if $G_1 \in \mathcal{F}_1$ and $G_2 \in \mathcal{F}_2$, then $|V(G_1) \cap V(G_2)| < R(r+1, \ell+1)$.*

3.3 A Parameterized approach to designing protocols

In Section 3.1, for each pair of constants r, ℓ, we saw a protocol for $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$ with sublinear communication complexity. We also showed that any good pair $(\mathcal{F}_1, \mathcal{F}_2)$ of graph families must be such that there are constants r, ℓ with $\mathcal{F}_1 \subseteq \mathcal{C}_r, \mathcal{F}_2 \subseteq \mathcal{I}_\ell$. In this section, we give an alternate protocol that uses the structure of the input graph G, to obtain a more refined upper bound on the communication complexity of $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$. For a graph, let $\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}$ denote the size of a minimum set S of vertices such that $V(G) \setminus S$ has a bipartition (V_1, V_2) with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. In this section, for a graph G and a good pair of graph families $(\mathcal{F}_1, \mathcal{F}_2)$, we give a protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ that has $\mathcal{O}(\log^2(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}) + \log n)$ communication complexity.

Proof of Theorem 1.2. We can assume that Alice and Bob both have a minimum vertex set S such that $V(G) \setminus S$ has a bipartition (V_1, V_2) with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Thus $|S| = \mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}$. The players also have a bipartition (V_1, V_2) of $V(G) \setminus S$, such that $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$.

Now let Alice receive the induced subgraph $G_1 \in \mathcal{F}_1$ and Bob receive the induced subgraph $G_2 \in \mathcal{F}_2$. The following is a protocol Π that Alice and Bob will execute. The protocol is executed in top down fashion, i.e., the two players resort to a step only if the previous steps are not applicable.

1. If either G_1 or G_2 is an empty graph, then they declare that the graphs are disjoint.
2. Alice and Bob run the protocol $\Pi_{r,\ell}$ to compute $\mathsf{GDISJ}_{G[S],\mathcal{C}_r,\mathcal{I}_\ell}(G_1[S], G_2[S])$. If there is a vertex intersection between $G_1[S]$ and $G_2[S]$, then they declare that the graphs G_1 and G_2 intersect and stop the protocol.

3. Suppose there is no vertex intersection between $G_1[S]$ and $G_2[S]$. Alice sends the vertices of the subgraph $G_1 \cap G[V_2]$ to Bob. If Bob finds that $V(G_2) \cap V(G_1 \cap G[V_2]) \neq \emptyset$, then Bob lets Alice know and they terminate the protocol.
4. Suppose Bob does not find any vertex common to both $V(G_1 \cap G[V_2])$ and $V(G_2)$. Then Bob sends the vertices of the subgraph $G_2 \cap G[V_1]$ to Alice. If Alice finds that $V(G_1) \cap V(G_2 \cap G[V_2]) \neq \emptyset$, then Alice lets Bob know and they terminate the protocol.

Otherwise, they declare that the two graphs G_1 and G_2 do not intersect on any vertex. If $V(G_1[S]) \cap V(G_2[S]) \neq \emptyset$, then the subprotocol $\Pi_{r,\ell}$ correctly detects the intersection in step 2. Otherwise, $V(G_1)$ and $V(G_2)$ can intersect either in V_1 or in V_2 and they detect in step 3 or step 4. If neither of the above cases happen, then $V(G_1) \cap V(G_2) = \emptyset$.

Next, we show the bound on the communication complexity. Following from Lemma 3.1, $\mathsf{GDISJ}_{G[S],\mathcal{C}_r,\mathcal{I}_\ell}(G_1[S], G_2[S])$ can be computed with the communication of $\mathcal{O}(\log^2 \mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2})$ bits. By definition, $G_1 \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Then, by Corollary 3.6, $|V(G_1) \cap V(G[V_2])| < R(r+1, \ell+1)$. Thus, in step 3, Alice sends at most $R(r+1,\ell+1)\log n$ bits to Bob. By a similar argument, in step 4, Bob sends at most $R(r+1,\ell+1)\log n$ bits to Alice. Therefore, the communication complexity of Π is $\mathcal{O}(\log^{r+\ell}(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}) + \log n)$. ◀

Suppose $(\mathcal{F}_1, \mathcal{F}_2)$ is a pair of hereditary graph families that are not good. By Theorem 1.1, for an n-vertex graph G, any protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ must have communication complexity $\Omega(n)$. This gives us the following corollary.

▶ **Corollary 3.7.** *Let $(\mathcal{F}_1, \mathcal{F}_2)$ be a pair of hereditary graph families that have infinitely many graphs in their intersection. Then, for each integer $n > 0$, there is a graph G such that for any computable function f, there cannot be a protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, that has communication complexity $f(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}) + o(n)$.*

4 Separating families

In this section, we design enumeration algorithms for separating families for a good pair of graph families. It was stated in [13] that a non-deterministic protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ corresponds to a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family. This means that if $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ has deterministic, and hence non-deterministic, complexity c, then there exists a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family of size 2^c. From Corollary 3.5 and Lemma 3.1, this means that, for an n-vertex graph, there exists a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family of size $2^{\mathcal{O}(\log^2 n)}$, for any constants r, ℓ. However, since the protocols use players with unlimited power of computation, this does not mean that there is an enumeration algorithm that finds such a separating family in time $2^{\mathcal{O}(\log^2 n)} n^{\mathcal{O}(1)}$. First, for an n-vertex graph G, we design an algorithm to enumerate a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family of size $2^{\mathcal{O}(\log^{r+\ell} n)}$, in time $2^{\mathcal{O}(\log^{r+\ell} n)} n^{\mathcal{O}(1)}$. This uses ideas from the protocol given in Lemma 3.4. Then, for a good pair $(\mathcal{F}_1, \mathcal{F}_2)$ of graph families, we utilize the structure of G, for a different approach to design enumeration algorithms for $(\mathcal{F}_1, \mathcal{F}_2)$-separating families.

▶ **Lemma 4.1.** *For any $r, \ell \in \mathbb{N}$, every graph with n vertices has a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family of cardinality $2^{\mathcal{O}(\log^{r+\ell} n)}$. Moreover, such a family can be enumerated in time $2^{\mathcal{O}(\log^{r+\ell} n)}$.*

Lemma 4.1 and Corollary 3.5 gives us the following Corollary.

▶ **Corollary 4.2.** *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families. Then, there are constants r and ℓ, such that every n-vertex graph has a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family of cardinality $2^{\mathcal{O}(\log^{r+\ell} n)}$ and it can be enumerated in time $2^{\mathcal{O}(\log^{r+\ell} n)}$.*

In fact, we obtain Theorem 1.4 from Lemma 4.1 and Corollary 3.5.

Separating families and parameterization

We give the proof of Theorem 1.5. We show that the upper bound obtained due to Corollary 4.2 can be improved if we use ideas from Parameterized Complexity, as we did for Theorem 1.2. This is extremely useful for designing FPT algorithms. Again, the ideas from the protocol of Lemma 3.4 comes to more use than the protocol of Lemma 3.1. To show this, we first prove the following lemma.

▶ **Lemma 4.3.** *Let $(\mathcal{F}_1, \mathcal{F}_2)$ be a good pair of graph families. Given, as input, $G, S \subseteq V(G)$ and partition $V_1 \uplus V_2$ of $V(G) \setminus S$ such that $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$, there is an algorithm to enumerate $(\mathcal{F}_1, \mathcal{F}_2)$-separating family $\mathcal{S}$ for G of cardinality $2^{\mathcal{O}(\log^c(|S|))}n^{\mathcal{O}(1)}$ in time $2^{\mathcal{O}(\log^c(|S|))}n^{\mathcal{O}(1)}$, where c is a constant.*

Proof. We know that $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Since $(\mathcal{F}_1, \mathcal{F}_2)$ is a good pair of graph families, by Corollary 3.5, we know that there are constants r, ℓ, such that for any $G_1 \in \mathcal{F}_1$ and $G_2 \in \mathcal{F}_2$, $\omega(G_1) \leq r$ and $\alpha(G_2) \leq \ell$. Let us define $c = r + \ell$. By Lemma 4.1, the graph $G[S]$ has a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family $\mathcal{S}'$ of cardinality $2^{\mathcal{O}(\log^{r+\ell}|S|)}$. Moreover, such a family can be enumerated in time $2^{\mathcal{O}(\log^{r+\ell}(|S|))}$. Now consider the family.

$$\mathcal{S} = \left\{A \cup (V_1 \setminus S_1) \cup S_2 \mid A \in \mathcal{S}', S_1 \subseteq V_1, S_2 \subseteq V_2, |S_1| < R(r+1, \ell+1), |S_2| < R(r+1, \ell+1)\right\}$$

The cardinality of $\mathcal{S}$ is bounded by $2^{\mathcal{O}(\log^c(|S|))}n^{\mathcal{O}(2R(r+1,\ell+1))}$ and it can be enumerated in time $2^{\mathcal{O}(\log^c(|S|))}n^{\mathcal{O}(2R(r+1,\ell+1))}$. We show that $\mathcal{S}$ is indeed a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family for G. Consider any disjoint vertex subsets U_1 and U_2 of $V(G)$ such that $G[U_1] \in \mathcal{F}_1$ and $G[U_2] \in \mathcal{F}_2$. We need to show that there is a set $T \in \mathcal{S}$ such that $U_1 \subseteq T$ and $T \cap U_2 = \emptyset$. Since the two families $\mathcal{F}_1$ and $\mathcal{F}_2$ are hereditary, $G[U_1 \cap S] \in \mathcal{F}_1$ and $G[U_2 \cap S] \in \mathcal{F}_2$. Since $\mathcal{S}'$ is a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family for $G[S]$ there is a set $A \in \mathcal{S}'$ such that $S \cap U_1 \subseteq A$ and $(S \cap U_2) \cap A = \emptyset$. Since $G[U_1], G[V_1] \in \mathcal{F}_1$ and $G[U_2], G[V_2] \in \mathcal{F}_2$, by Corollary 3.6, we know that $|U_1 \cap V_2| < R(r+1, \ell+1)$ and $|U_2 \cap V_1| < R(r+1, \ell+1)$. Now consider the set $T = A \cup (V_1 \setminus (U_2 \cap V_1)) \cup (U_1 \cap V_2)$. Since $|U_1 \cap V_2| < R(r+1, \ell+1)$ and $|U_2 \cap V_1| < R(r+1, \ell+1)$, by the definition of $\mathcal{S}$, $T \in \mathcal{S}$. Notice that $U_1 \subseteq T$ and $U_2 \cap T = \emptyset$. Hence, we are done. ◀

Lemma 4.3 gives us Theorem 1.5. Suppose there was an approximation algorithm $\mathcal{A}$ for $(\mathcal{F}_1, \mathcal{F}_2)$-Partition, where the approximation factor is defined by a computable function f depending only on the size of an optimal solution, and let the running time of $\mathcal{A}$ be $T(n)$ on an n-vertex input graph. Then, a $(\mathcal{F}_1, \mathcal{F}_2)$-seperating family, for G, of cardinality $2^{\mathcal{O}(\log^c f(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}))}n^{\mathcal{O}(1)}$ can be enumerated in time $2^{\mathcal{O}(\log^c f(\mathsf{opt}))}n^{\mathcal{O}(1)}$, where c is the same constant as in Theorem 1.5, which is at most $r + \ell$.

5 Applications in Parameterized and Exact Algorithms

In this section we relate the results obtained in previous sections to exact and FPT algorithms. The main result of this section is to show that the $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition problem is FPT. In fact, we propose an algorithm strategy that might result in faster running times than that of the best known algorithms for certain pairs $(\mathcal{F}_1, \mathcal{F}_2)$. We also provide combinatorial bounds on the number of maximal induced subgraphs that have a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$. We also give a strategy to design an enumeration algorithm for all such maximal induced subgraphs. Similarly, we can find the maximum(minimum) size of such an induced subgraph. These results and their corollaries can be found in the full version of the paper.

Parameterized Algorithms

The question of what is the maximum size of an induced subgraph, that has a vertex bipartition (A, B) with $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$, brings us to the question of how 'far' a graph is from becoming a graph with the desired bipartition. The $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition problem addresses this question. In this part, we look at this problem and a technique to solve this problem, when the pair of families are a good pair of families. The technique we use is an adaptation of the popular iterative compression technique.

▶ **Observation 5.1.** *If an instance (G, k) is a* YES *instance of $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition, then for any induced subgraph $G' \leq_s G$, (G', k) is also a* YES *instance of $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition.*

Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families, and for any positive integer k, the families $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ have FPT recognition algorithms, that is, there are algorithms which take as input a graph G and an integer k, decides whether $G \in \mathcal{F}_i + kv, i \in \{1, 2\}$ and runs in time $f(k)|V(G)|^{\mathcal{O}(1)}$. For ease of notation, if $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families, and the families $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ have FPT recognition algorithms, then we call $(\mathcal{F}_1, \mathcal{F}_2)$ an FPT-*good pair of families*.

We obtain a fast FPT algorithm for $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition, as claimed in Theorem 1.6 by incorporating the iterative compression technique. For more details about the algorithmic technique of iterative compression we refer to the book (chapter 4 [3]).

Proof Sketch for Theorem 1.6. Let (G, k) be an input instance. The algorithm is based on the iterative compression technique. Due to Observation 5.1, the iterative compression technique is meaningful for this problem. The iteration step is exactly as described in [3](chapter 4). The description of the compression problem and an algorithm to solve the same is given below.

The input of the compression problem is a graph G' and a vertex set $S \subseteq V(G')$, of size at most $k + 1$. The set S satisfies the property that there is a partition $V_1 \uplus V_2$ of $V(G) \setminus S$ such that $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. The compression problem outputs YES if there is a vertex set S' of size at most k such that there is a partition $V_1' \uplus V_2'$ of $V(G) \setminus S'$ with $G[V_1'] \in \mathcal{F}_1$ and $G[V_2'] \in \mathcal{F}_2$. Otherwise, the output is NO. Lemma 4.3 can be used to solve the compression problem in time $2^{\mathcal{O}(\log^c k)} n^{\mathcal{O}(1)} \cdot 2k \max\{T_1(n, k), T_2(n, k)\}$.

By Lemma 4.3, we know that there is an enumeration algorithm which outputs a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family $\mathcal{S}$ of cardinality $2^{\mathcal{O}(\log^c |S|)} n^{\mathcal{O}(1)}$ in time $2^{\mathcal{O}(\log^c |S|)} n^{\mathcal{O}(1)}$. Now for each $S \in \mathcal{S}$ and each pair of non-negative integers k_1, k_2 such that $k_1 + k_2 \leq k$, we run $\mathcal{A}_1$ on $(G[S], k_1)$ and $\mathcal{A}_2$ on $(G - S, k_2)$. We output YES if both $\mathcal{A}_1$ and $\mathcal{A}_2$ outputs YES. Otherwise our algorithm will output NO. The full proof is in the full version of the paper. ◀

There are several corollaries of Theorem 1.6, to be found in the full version.

6 Conclusion

In this paper, we studied the parameterized communication complexity of the function $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$. We also obtained separating families for good pairs of families, and used them to give combinatorial bounds, exact algorithms and FPT algorithms. An important question here is to see if the lower bounds for CIS_G can be used to obtain non-trivial lower bounds for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, when $(\mathcal{F}_1, \mathcal{F}_2)$ is a good pair of graph families. Also, it would be interesting to study the communication complexity of these functions in terms of other parameters of the input. We would like to thank Pranabendu Misra and an anonymous reviewer for insightful comments.

References

1. Kazuyuki Amano. Some improved bounds on communication complexity via new decomposition of cliques. *Discrete Applied Mathematics*, 166:249–254, 2014. doi:10.1016/j.dam.2013.09.015.
2. Nicolas Bousquet, Aurélie Lagoutte, and Stéphan Thomassé. Clique versus independent set. *Eur. J. Comb.*, 40:73–92, 2014. doi:10.1016/j.ejc.2014.02.003.
3. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
4. Marek Cygan and Marcin Pilipczuk. Split vertex deletion meets vertex cover: New fixed-parameter and exact exponential-time algorithms. *Inf. Process. Lett.*, 113(5-6):179–182, 2013. doi:10.1016/j.ipl.2013.01.001.
5. R. Diestel. *Graph Theory*. Springer, Berlin, second ed., electronic edition, February 2000.
6. Tomas Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. Complexity of graph partition problems. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC'99, pages 464–472, New York, NY, USA, 1999. ACM. doi:10.1145/301250.301373.
7. Mika Göös. Lower bounds for clique vs. independent set. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1066–1076, 2015. doi:10.1109/FOCS.2015.69.
8. Mika Göös, T. S. Jayram, Toniann Pitassi, and Thomas Watson. Randomized communication vs. partition number. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:169, 2015. URL: http://eccc.hpi-web.de/report/2015/169.
9. Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1077–1088, 2015. doi:10.1109/FOCS.2015.70.
10. Vince Grolmusz and Gábor Tardos. A note on non-deterministic communication complexity with few witnesses. *Theory Comput. Syst.*, 36(4):387–391, 2003. doi:10.1007/s00224-003-1158-7.
11. Hao Huang and Benny Sudakov. A counterexample to the Alon-Saks-Seymour conjecture and related problems. *Combinatorica*, 32(2):205–219, 2012. doi:10.1007/s00493-012-2746-4.
12. Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, New York, NY, USA, 1997.
13. László Lovász. Communication complexity: a survey. *Paths, flows, and VLSI-layout*, pages 235–265, 1990.
14. László Lovász. Stable sets and polynomials. *Discrete Mathematics*, 124(1-3):137–153, 1994. doi:10.1016/0012-365X(92)00057-X.
15. Manami Shigeta and Kazuyuki Amano. Ordered biclique partitions and communication complexity problems. *Discrete Applied Mathematics*, 184:248–252, 2015. doi:10.1016/j.dam.2014.10.029.
16. Mihalis Yannakakis. Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43(3):441–466, 1991. doi:10.1016/0022-0000(91)90024-Y.
17. Andrew Chi-Chih Yao. Some complexity questions related to distributive computing(preliminary report). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC'79, pages 209–213, New York, NY, USA, 1979. ACM. doi:10.1145/800135.804414.

Monitor Logics for Quantitative Monitor Automata*

Erik Paul

Institute of Computer Science, Leipzig University, Leipzig, Germany
epaul@informatik.uni-leipzig.de

Abstract

We introduce a new logic called Monitor Logic and show that it is expressively equivalent to Quantitative Monitor Automata.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Quantitative Monitor Automata, Nested Weighted Automata, Monitor Logics, Weighted Logics

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.14

1 Introduction

In the last years, there has been increasing interest in quantitative features of the specification and analysis of systems. Such quantitative aspects include the consumption of a certain resource or the output of a benefit. Both weighted automata and weighted logics [7] are means to achieve this quantitative description of systems. They can be employed for both finite and infinite input.

Very recently, Chatterjee et al. introduced a new automaton model operating on infinite words [4]. *Quantitative Monitor Automata* are equipped with a finite number of monitor counters. At each transition, a counter can be started, terminated, or the value of the counter can be increased or decreased. The term "monitor" stems from the fact that the values of the counters do not influence the behavior of the automaton. The values of the counters when they are terminated provide an infinite sequence of weights, which is evaluated into a single weight using a *valuation function*.

Quantitative Monitor Automata possess several interesting features. They are expressively equivalent to a subclass of *Nested Weighted Automata* [3], an automaton model which for many valuation functions has decidable emptiness and universality problems. Quantitative Monitor Automata are also very expressive. As an example, imagine a storehouse with a resource which is restocked at regular intervals. Between restocks, demands can remove one unit of this resource at a time. Such a succession of restocks and demands can be modeled as an infinite sequence over the alphabet $\{\mathsf{restock}, \mathsf{demand}\}$. Interesting quantitative properties of such a sequence include the long-term average demand, the minimum demand and the maximum demand between restocks. These properties can be described using Quantitative Monitor Automata. At every restock a counter is started, counting the number of demands until the next restock. An appropriate valuation function then computes the desired property. For the average demand, this can be achieved with the *Cesàro mean* which was introduced to automata theory by Chatterjee et. al in [2]. Note that behaviors like these cannot be

* This work was supported by Deutsche Forschungsgemeinschaft (DFG), Graduiertenkolleg 1763 (QuantLA).

modeled using *weighted Büchi-automata* [11, 12] or their extension with valuation functions [8]. In the latter model, the Cesàro mean of any sequence is bounded by the largest transition weight in the automaton. This is not the case for Quantitative Monitor Automata.

In this paper, we develop a logic which is expressively equivalent to Quantitative Monitor Automata. Our main results are the following.

- We introduce a new logic which we call *Monitor Logic*.
- We show that this Monitor Logic is expressively equivalent to Quantitative Monitor Automata.
- We show various closure properties of Quantitative Monitor Automata and prove that Muller and Büchi acceptance conditions provide the same expressive power.

The relationship between automata and logics plays a large role in specification and verification. Statements are often easier to formulate in the form of a logic formula rather than directly as an automaton. Consequently, the fundamental Büchi-Elgot-Trakhtenbrot Theorem [1, 10, 17], which established the coincidence of regular languages with languages definable in monadic second order logic, has found use in many areas of application. An extension to semiring weighted automata was later given by Droste and Gastin [6].

Our logic is equipped with three quantifiers. A sum quantifier to handle the computations on the counters, a valuation quantifier to handle the valuation, and a third quantifier to combine the weights of all runs on a word. Our biggest challenge was to find appropriate restrictions on the use of the quantifiers. Without any restrictions the logic would be too powerful, which we also formally prove using counter examples. The most important result of our considerations is that the computations of the sum quantifier should depend on an MSO-definable condition.

We note that our constructions are effective. Given a formula from our logic, we can effectively construct a Quantitative Monitor Automaton describing this formula. Conversely, for every automaton we can effectively construct a formula with the automaton's behavior.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ denote the natural numbers, $\mathbb{Z}$ the integers and $\mathbb{R}$ the reals. An alphabet Σ is a finite set. An infinite word over Σ is a sequence $w = a_0 a_1 a_2 \ldots$ from Σ. The set of infinite words over Σ is denoted by Σ^ω. The set of finite words Σ^* over Σ is defined as the set of finite sequences $a_0 a_1 \ldots a_n$ from Σ. The empty word is denoted by ε. A mapping $\Sigma^\omega \to \mathbb{R} \cup \{\infty\}$ is called a *series*.

A *(nondeterministic) Muller automaton over* Σ (NMA) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \mathcal{F}, \delta)$ where (1) Σ is an alphabet, (2) Q is a finite set of states, (3) $q_0 \in Q$ is the initial state, (4) $\mathcal{F} \subseteq \mathcal{P}(Q)$ is the set of final sets and (5) $\delta \subseteq Q \times \Sigma \times Q$ is the set of transitions.

Let $a_0 a_1 \ldots \in \Sigma^\omega$ be an infinite word. A *run of* $\mathcal{A}$ *over* w is an infinite sequence of transitions $r_w = (t_i)_{i \geq 0}$ so that $t_i = (q_i, a_i, q_{i+1}) \in \delta$ for all $i \geq 0$. We denote by $\mathrm{In}^Q(r_w)$ the set of states which appear infinitely many times in r_w, i.e.

$$\mathrm{In}^Q(r_w) = \{q \in Q \mid \forall i \exists j \geq i : t_j = (q, a_j, q_{j+1})\}.$$

A run r_w of $\mathcal{A}$ over $w \in \Sigma^\omega$ is called *accepting* if $\mathrm{In}^Q(r_w) \in \mathcal{F}$, that is, if the states which appear infinitely many times in r_w form a set in $\mathcal{F}$. In this case we say that w is *recognized (accepted) by* $\mathcal{A}$. The set of accepting runs over a word $w \in \Sigma^\omega$ is denoted by $\mathrm{Acc}_{\mathcal{A}}(w)$. The *infinitary language* of $\mathcal{A}$, denoted by $\mathcal{L}_\omega(\mathcal{A})$, is the set of all infinite words that are accepted by $\mathcal{A}$. A language $L \subseteq \Sigma^\omega$ is called ω-recognizable if there is a Muller automaton $\mathcal{A}$ so that $L = \mathcal{L}_\omega(\mathcal{A})$.

A Muller automaton $\mathcal{A} = (\Sigma, Q, q_0, \mathcal{F}, \delta)$ is called *deterministic* if the set of transitions δ can be interpreted as a function $Q \times A \to Q$, i.e. if for every $(p, a) \in Q \times A$ there exists exactly one $q \in Q$ with $(p, a, q) \in \delta$. It is well known and will be important to us that for each Muller automaton $\mathcal{A}$, we can effectively construct a deterministic Muller automaton with the same language [15, 16].

3 Quantitative Monitor Automata

An *ω-valuation function* is a mapping $\mathrm{Val}\colon \mathbb{Z}^{\mathbb{N}} \to \mathbb{R} \cup \{\infty\}$ that assigns real values or ∞ to infinite sequences of integers. Typical examples of such functions are the Cesàro mean

$$\textsc{Ces}((z_i)_{i\geq 0}) = \begin{cases} \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} z_i & \text{if this limit exists} \\ \infty & \text{otherwise,} \end{cases}$$

the supremum $\textsc{Sup}((z_i)_{i\geq 0}) = \sup_{i\geq 0} z_i$, the infimum $\textsc{Inf}((z_i)_{i\geq 0}) = \inf_{i\geq 0} z_i$, the limit superior $\textsc{LimSup}((z_i)_{i\geq 0}) = \lim_{n\to\infty} \sup_{i\geq n} z_i$ and the limit inferior $\textsc{LimInf}((z_i)_{i\geq 0}) = \lim_{n\to\infty} \inf_{i\geq n} z_i$.

For a new symbol $\mathbb{1}$ and an ω-valuation function Val, we extend the domain of Val to sequences $\mathbf{z} = (z_i)_{i\geq 0}$ from $\mathbb{Z} \cup \{\mathbb{1}\}$ as follows. If at some point $\mathbf{z}$ becomes constantly $\mathbb{1}$, we let $\mathrm{Val}(\mathbf{z}) = \infty$. Otherwise we let $\mathbf{z}'$ be the subsequence of $\mathbf{z}$ which contains all elements that are not $\mathbb{1}$ and define $\mathrm{Val}(\mathbf{z}) = \mathrm{Val}(\mathbf{z}')$.

We now define Quantitative Monitor Automata. We use a different name, however, in order to distinguish between Büchi and Muller acceptance conditions.

A *Büchi automaton with monitor counters* (BMCA) $\mathcal{A}$ is a tuple $(\Sigma, Q, I, F, \delta, n, \mathrm{Val})$ where (1) Σ is the alphabet, (2) Q is a finite set of states, (3) $I \subseteq Q$ is the set of initial states, (4) F is the set of accepting states, (5) δ is a finite subset of $Q \times \Sigma \times Q \times (\mathbb{Z} \cup \{s, t\})^n$, called the transition relation, such that for every $(q, a, q', \mathbf{u}) \in \delta$ at most one component of $\mathbf{u}$ contains s, (6) n is the number of counters and (7) Val is an ω-valuation function.

Intuitively, the meaning of a transition $(q, a, q', \mathbf{u})$ is that if the automaton is in state q and reads an a, it can move to state q' and start counter j if $u^j = s$, add u^j to the current value of counter j if this counter is activated and $u^j \in \mathbb{Z}$, or stop counter j if $u^j = t$. Initially, all counters are inactive. We will also call $\mathcal{A}$ an n-BMCA or a Val-BMCA, thereby stressing the number of counters or the ω-valuation function used.

Let $a_0 a_1 \ldots \in \Sigma^\omega$ be an infinite word. A *run of $\mathcal{A}$ over w* is an infinite sequence of transitions $r_w = (t_i)_{i\geq 0}$ so that $t_i = (q_i, a_i, q_{i+1}, \mathbf{u}_i) \in \delta$ for all $i \geq 0$.

A run r_w of $\mathcal{A}$ over $w \in \Sigma^\omega$ is called *accepting* if (1) $q_0 \in I$, (2) $\mathrm{In}^Q(r_w) \cap F \neq \emptyset$, (3) if $u_i^j = s$ for some $i \geq 0$, then there exists $l > i$ such that $u_l^j = t$ and for all $k \in \{i+1, \ldots, l-1\}$ we have $u_k^j \in \mathbb{Z}$, (4) if $u_i^j = t$ for some $i \geq 0$, then there exists $l < i$ such that $u_l^j = s$ and for all $k \in \{l+1, \ldots, i-1\}$ we have $u_k^j \in \mathbb{Z}$ and (5) infinitely often some counter is activated, i.e.

$$\{i \geq 0 \mid u_i^j = s \text{ for some } j\}$$

is an infinite set. The set of accepting runs over a word $w \in \Sigma^\omega$ is denoted by $\mathrm{Acc}_{\mathcal{A}}(w)$.

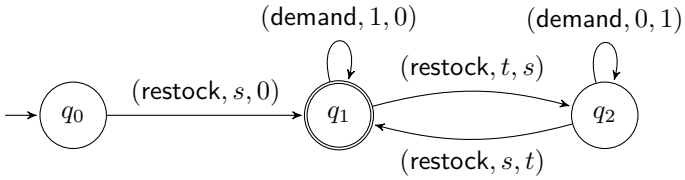
An accepting run r_w defines a sequence $\mathbf{z} = (z_i)_{i\geq 0}$ from $\mathbb{Z} \cup \{\mathbb{1}\}$ as follows. If $u_i^j = s$ for some $j \in \{1, \ldots, n\}$ and $l > i$ is such that $u_l^j = t$ and for all $k \in \{i+1, \ldots, l-1\}$ we have $u_k^j \in \mathbb{Z}$, then $z_i = \sum_{k=i+1}^{l-1} u_k^j$. If $u_i^j \neq s$ for all $j \in \{1, \ldots, n\}$, then $z_i = \mathbb{1}$. We also call $\mathbf{z}$ the *weight-sequence associated to r_w*. The weight of the run r_w is defined as $\mathrm{Val}(r_w) = \mathrm{Val}(\mathbf{z})$. The behavior of the automaton $\mathcal{A}$ is the series $[\![\mathcal{A}]\!]\colon \Sigma^\omega \to \mathbb{R} \cup \{\infty\}$

defined by $[\![\mathcal{A}]\!](w) = \inf_{r_w \in \mathrm{Acc}(w)} \mathrm{Val}(r_w)$, where the infimum over the empty set is defined as ∞. A series $\Sigma^\omega \to \mathbb{R} \cup \{\infty\}$ is called *MC-recognizable* if there exists a BMCA $\mathcal{A}$ such that $[\![\mathcal{A}]\!] = S$. The notions of *n-MC-recognizable* and Val-*MC-recognizable* are defined likewise.

A *Muller automaton with monitor counters* (MMCA) is defined like a BMCA, but instead of a set of accepting states we have a set of accepting sets $\mathcal{F} \subseteq \mathcal{P}(Q)$. The condition (2) for a run r_w on a word $w \in \Sigma^\omega$ to be accepting is then replaced by $\mathrm{In}^Q(r_w) \in \mathcal{F}$, i.e. a Muller acceptance condition.

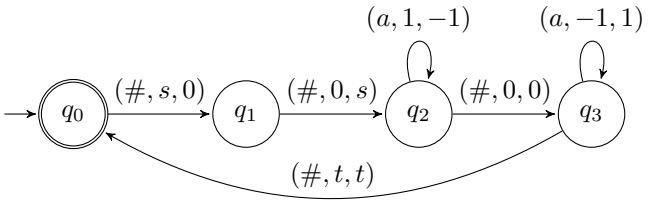
Büchi automata with monitor counters use a Büchi acceptance condition, i.e. at least one accepting state has to appear infinitely often. Lemma 3 shows that using a Muller acceptance condition does not influence the expressive power.

▶ **Example 1.** Consider the the alphabet $\Sigma = \{\mathsf{demand}, \mathsf{restock}\}$ with the ω-valuation function Val = Ces. We model a storehouse with some sort of supply that is restocked whenever restock is encountered, and one unit of the supply is removed at every demand. Given an infinite sequence of restocks and demands, we are interested in the long-time average demand between restocks. Under the assumption that every such sequence starts with a restock, this behavior is modeled by the following automaton with two monitor counters.



When for the valuation function we take Inf or Sup, the automaton above describes the lowest or highest demand ever encountered, for the latter assuming that the demands are bounded.

▶ **Example 2** ([4])**.** Consider the alphabet $\Sigma = \{a, \#\}$ and the language L consisting of words $(\#^2 a^* \# a^* \#)^\omega$. On these words, we consider the quantitative property "the maximal block-length difference between even and odd positions", i.e. the value of the word $\#\#a^{m_1}\#a^{m_2}\#\#\#\ldots$ shall be $\sup_{i \geq 1} |m_{2i-1} - m_{2i}|$. With the choice Val = Sup, a BMCA realizing this behavior is the following.



Each $(\#^2 a^{m_1} \# a^{m_2} \#)$-block is processed by starting both counters on the first two #'s, accumulating m_1 into the first counter and accumulating $-m_1$ into the second, reading #, then accumulating $m_1 - m_2$ into the first counter and $-m_1 + m_2$ into the second, and finally terminating both counters on the last #. Thus, the associated weight-sequence for only this block is $(m_1 - m_2, -m_1 + m_2, \mathbb{1}, \ldots, \mathbb{1})$. Clearly, the final value of counter one is always the negative of the final value in counter two. Since our ω-valuation function is Sup, only the positive counter value actually plays a role in the value assigned to the whole word, and this positive value is $|m_1 - m_2|$.

In the rest of this section, we prove various closure properties for automata with monitor counters and that BMCA and MMCA have the same expressive power.

▶ **Lemma 3.** *Büchi automata with monitor counters are expressively equivalent to Muller automata with monitor counters.*

Proof. The proof is similar to the standard construction to show that Büchi automata are expressively equivalent to Muller automata, see for example [9]. ◀

The next lemma shows that MC-recognizable series are closed under projections and their preimage. Given two alphabets Σ and Γ and a mapping $h\colon \Sigma \to \Gamma$ and thus a homomorphism $h\colon \Sigma^\omega \to \Gamma^\omega$, we define for every $S\colon \Sigma^\omega \to \mathbb{R} \cup \{\infty\}$ the *projection* $h(S)\colon \Gamma^\omega \to \mathbb{R} \cup \{\infty\}$ by

$$h(S)(w) = \inf\{S(v) \mid h(v) = w\}$$

for every $w \in \Gamma^\omega$. Moreover, if $S'\colon \Gamma^\omega \to \mathbb{R} \cup \{\infty\}$, then we put $h^{-1}(S') = S' \circ h$, i.e. $h^{-1}(S')\colon \Sigma^\omega \to \mathbb{R} \cup \{\infty\},\ w \mapsto S'(h(w))$.

▶ **Lemma 4.** *Let Σ and Γ be two alphabets, $h\colon \Sigma \to \Gamma$ be a mapping and* Val *be an ω-valuation function.*

(i) *If $S\colon \Sigma^\omega \to \mathbb{R} \cup \{\infty\}$ is* Val*-MC-recognizable, then the projection $h(S)\colon \Gamma^\omega \to \mathbb{R} \cup \{\infty\}$ is also* Val*-MC-recognizable.*

(ii) *If $S'\colon \Gamma^\omega \to \mathbb{R} \cup \{\infty\}$ is* Val*-MC-recognizable, then $h^{-1}(S')\colon \Sigma^\omega \to \mathbb{R} \cup \{\infty\}$ is also* Val*-MC-recognizable.*

For two series $S_1, S_2\colon \Sigma^\omega \to \mathbb{R} \cup \{\infty\}$, the minimum $\min(S_1, S_2)$ of S_1 and S_2 is defined pointwise, i.e.

$$\min(S_1, S_2)(w) = \min\{S_1(w), S_2(w)\}.$$

Applying the usual union construction for automata, we can show that the minimum of two MC-recognizable series is MC-recognizable as well.

▶ **Lemma 5.** *For any given ω-valuation function* Val*, the* Val*-MC-recognizable series are closed under minimum.*

Let $L \subseteq \Sigma^\omega$ and $S\colon \Sigma^\omega \to \mathbb{R} \cup \{\infty\}$. The *intersection of L and S* is the series $L \cap S\colon \Sigma^\omega \to \mathbb{R} \cup \{\infty\}$ defined for $w \in \Sigma^\omega$ by

$$L \cap S(w) = \begin{cases} S(w) & \text{if } w \in L \\ \infty & \text{otherwise.} \end{cases}$$

The intersection of a recognizable language with an MC-recognizable series is MC-recognizable as well.

▶ **Lemma 6.** *Let* Val *be an ω-valuation function, let $L \subseteq \Sigma^\omega$ be ω-recognizable and $S\colon \Sigma^\omega \to \mathbb{R} \cup \{\infty\}$ be* Val*-MC-recognizable. Then $L \cap S$ is also* Val*-MC-recognizable.*

Proof. The proof is similar to the standard product construction to show that recognizable languages are closed under intersection. ◀

4 Monitor MSO logic

We first want to give a motivation for the quantifiers and restrictions we use in our logic. We are looking for a logic which is expressively equivalent to automata with monitor counters. It is clear that we need a valuation quantifier in order to model the valuation done by the automata. The question is which types of formulas should be allowed in the scope of the valuation quantifier. From [8] it follows that allowing only *almost Boolean* formulas (see below) is too weak. We would only describe Muller automata over valuation monoids, and these are strictly weaker than automata with monitor counters [4].

We therefore have to allow at least some other quantifier in the scope of the valuation quantifier. Taking into account the automaton model we want to describe, this should be a sum quantifier. Most weighted logics [6, 9, 8, 13, 14, 5] use quantifiers that act unconditionally on the whole input, i.e. on the whole word, tree or picture. However, in Lemma 11 we will see that in our case, an unrestricted sum quantifier quickly gets out of hand.

The intention of the sum quantifier as we define it here is to have a sum quantifier which acts on infinite words, but still computes only finite sums on a given word. The computation of the sum quantifier depends on a first order variable x and a second order variable X provided to it. The variable X serves as a "list" of start and stop positions, and the variable x indicates where the summation on the infinite word should take place. Simply put, the sum is evaluated to $\mathbb{1}$ if x does not point to a position in X or there is no successor of x in X. Otherwise, if y is x's successor in X, the sum is taken from $x+1$ to $y-1$.

Intuitively, each sum quantifier corresponds to a counter. In a run of an automaton with monitor counters, not more than one counter can be started at each letter of the given word. Therefore, we use Boolean formulas to choose which counter to use. We combine these choices between counters into so-called *x-summing* formulas, where x is the first order variable provided to each sum quantifier in the formula.

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 a three step logic over an alphabet Σ according to the following grammars.

$$\begin{aligned}
\beta &::= P_a(x) \mid x \leq y \mid x \in X \mid \neg\beta \mid \beta \vee \beta \mid \exists x.\beta \mid \exists X.\beta \\
\psi &::= k \mid \beta \,?\, \psi : \psi \\
\zeta_x &::= \mathbb{1} \mid \beta \,?\, \zeta_x : \zeta_x \mid \bigoplus\nolimits^{x,X} y.\psi \\
\varphi &::= \beta \,?\, \varphi : \varphi \mid \min(\varphi,\varphi) \mid \inf x.\varphi \mid \inf X.\varphi \mid \operatorname{Val} x.\zeta_x
\end{aligned}$$

where $x, y, X \in \mathcal{V}$, $a \in \Sigma$ and $k \in \mathbb{Z}$. The formulas β are called *Boolean* or *MSO* formulas, the formulas ψ *almost Boolean* formulas, the formulas ζ_x *x-summing* formulas and the formulas φ *monitor MSO* (mMSO) formulas. We remark that within an x-summing formula, the first order variable provided to each sum quantifier is always x. This restriction is not imposed on the second order quantifiers, i.e. $\beta \,?\, \bigoplus^{x,X} y.\psi_1 : \bigoplus^{x,Z} y.\psi_2$ is an x-summing formula, but $\beta \,?\, \bigoplus^{x,X} y.\psi_1 : \bigoplus^{z,Z} y.\psi_2$ is neither an x-summing nor a z-summing formula. Also note that the x-summing formulas are only auxiliary formulas, see Remark 7 later on.

The set of free variables $\operatorname{Free}(\varphi)$ is defined as usual, i.e. $\exists, \inf$ and Val bind variables, and in $\bigoplus^{x,X} y.\psi$ the variable y is bound. A formula without free variables is called a *sentence*.

Let $w \in \Sigma^\omega$. We put $\operatorname{dom}(w) = \{0, 1, 2, \ldots\}$ and denote the i-th letter of w by w_i, i.e. $w = w_0 w_1 w_2 \ldots$. Let $\mathcal{V}$ be a finite set of first and second order variables with $\operatorname{Free}(\varphi) \subseteq \mathcal{V}$. A $(\mathcal{V}, w)$-assignment is a mapping $\rho\colon \mathcal{V} \to \operatorname{dom}(w) \cup \mathcal{P}(\operatorname{dom}(w))$ where every first order

variable is mapped to an element of $\mathrm{dom}(w)$ and every second order variable is mapped to a subset of $\mathrm{dom}(w)$. The update $\rho[x \to i]$ for $i \in \mathrm{dom}(w)$ is defined as $\rho[x \to i](x) = i$ and $\rho[x \to i](\mathcal{X}) = \rho(\mathcal{X})$ for all $\mathcal{X} \in \mathcal{V} \setminus \{x\}$. The update $\rho[X \to I]$ for $I \subseteq \mathrm{dom}(w)$ is defined similarly. We encode $(\mathcal{V}, w)$-assignments as usual with an extended alphabet $\Sigma_\mathcal{V} = \Sigma \times \{0,1\}^\mathcal{V}$. Here, we refer to a word over the alphabet $\Sigma_\mathcal{V}$ by (w, ρ), where w is the projection to Σ and ρ is the projection to $\{0,1\}^\mathcal{V}$. A word over $\Sigma_\mathcal{V}$ represents an assignment if and only if for every first order variable the respective row in the extended word contains exactly one 1, in which case (w, ρ) is called *valid*.

It is not difficult to see that the set

$$N_\mathcal{V} = \{(w, \rho) \in \Sigma_\mathcal{V}^\omega \mid (w, \rho) \text{ is valid}\}$$

is ω-recognizable. Let $(w, \rho) \in \Sigma_\mathcal{V}^\omega$. For a Boolean formula β we define the satisfaction relation $(w, \rho) \models \beta$ as usual: if (w, ρ) is not valid, then $(w, \rho) \models \beta$ does not hold; otherwise we define it as follows.

$$\begin{array}{lcl}
(w,\rho) \models P_a(x) & \iff & w_{\rho(x)} = a \\
(w,\rho) \models x \le y & \iff & \rho(x) \le \rho(y) \\
(w,\rho) \models x \in X & \iff & \rho(x) \in \rho(X) \\
(w,\rho) \models \neg\beta & \iff & (w,\rho) \models \beta \text{ does not hold} \\
(w,\rho) \models \beta_1 \vee \beta_2 & \iff & (w,\rho) \models \beta_1 \text{ or } (w,\rho) \models \beta_2 \\
(w,\rho) \models \exists x.\beta & \iff & (w,\rho[x \to i]) \models \beta \text{ for some } i \in \mathrm{dom}(w) \\
(w,\rho) \models \exists X.\beta & \iff & (w,\rho[X \to I]) \models \beta \text{ for some } I \subseteq \mathrm{dom}(w).
\end{array}$$

Let β be an MSO formula. We will write Σ_β for $\Sigma_{\mathrm{Free}(\beta)}$ and N_β for $N_{\mathrm{Free}(\beta)}$. We recall the fundamental Büchi Theorem [1], namely that for $\mathrm{Free}(\beta) \subseteq \mathcal{V}$ the language

$$\mathcal{L}_\mathcal{V}(\beta) = \{(w, \rho) \in N_\mathcal{V} \mid (w, \rho) \models \beta\}$$

defined by β over $\Sigma_\mathcal{V}$ is ω-recognizable. We abbreviate $\mathcal{L}(\beta) = \mathcal{L}_{\mathrm{Free}(\beta)}(\beta)$.

Conversely, every ω-recognizable language $L \subseteq \Sigma^\omega$ is definable by an MSO sentence β, i.e. $L = \mathcal{L}(\beta)$.

We now come to the semantics of the remaining formulas. Let Val be an ω-valuation function. For an almost Boolean, x-summing or monitor MSO formula η we define the *semantics* $[\![\eta]\!]_\mathcal{V}(w, \rho)$ *of* η *under the* $(\mathcal{V}, w)$*-assignment* ρ as follows: if (w, ρ) is not valid, then $[\![\eta]\!]_\mathcal{V}(w, \rho) = \infty$; otherwise the semantics are defined as follows.

$$\begin{array}{rcl}
[\![k]\!]_\mathcal{V}(w,\rho) & = & k \\
[\![\beta \,?\, \psi_1 : \psi_2]\!]_\mathcal{V}(w,\rho) & = & \begin{cases} [\![\psi_1]\!]_\mathcal{V}(w,\rho) & \text{if } (w,\rho) \models \beta \\ [\![\psi_2]\!]_\mathcal{V}(w,\rho) & \text{otherwise} \end{cases} \\
[\![\bigoplus^{x,X} y.\psi]\!]_\mathcal{V}(w,\rho) & = & \begin{cases} \displaystyle\sum_{i=\rho(x)+1}^{\min\{j \in \rho(X) \mid j > \rho(x)\}-1} [\![\psi]\!]_\mathcal{V}(w, \rho[y \to i]) & \text{if } \rho(x) \in \rho(X) \text{ and} \\ & \{j \in \rho(X) \mid j > \rho(x)\} \neq \emptyset \\ \mathbb{1} & \text{otherwise.} \end{cases} \\
[\![\min(\varphi_1, \varphi_2)]\!]_\mathcal{V}(w,\rho) & = & \min\{[\![\varphi_1]\!]_\mathcal{V}(w,\rho), [\![\varphi_2]\!]_\mathcal{V}(w,\rho)\} \\
[\![\inf x.\varphi]\!]_\mathcal{V}(w,\rho) & = & \inf_{i \in \mathrm{dom}(w)} [\![\varphi]\!]_\mathcal{V}(w, \rho[x \to i]) \\
[\![\inf X.\varphi]\!]_\mathcal{V}(w,\rho) & = & \inf_{I \subseteq \mathrm{dom}(w)} [\![\varphi]\!]_\mathcal{V}(w, \rho[X \to I]) \\
[\![\mathrm{Val}\, x.\zeta_x]\!]_\mathcal{V}(w,\rho) & = & \mathrm{Val}((([\![\zeta_x]\!]_\mathcal{V}(w, \rho[x \to i]))_{i \in \mathrm{dom}(w)}).
\end{array}$$

We write $[\![\eta]\!]$ for $[\![\eta]\!]_{\mathrm{Free}(\eta)}$. To indicate the ω-valuation function Val or the alphabet Σ used, we may denote the set of monitor MSO formulas by $\mathrm{mMSO}(\Sigma, \mathrm{Val})$.

▶ Remark 7. From the semantics defined here it is clear that any x-summing sentence ζ_x is semantically equivalent to $\mathbb{1}$. In this sense, the x-summing formulas constitute no meaningful fragment of our logic, and are only auxiliary formulas for the construction of monitor MSO formulas.

Note also that for (w, ρ) valid, we have $[\![\mathrm{Val}\, x.\mathbb{1}]\!](w, \rho) = \infty$. By abuse of notation, we can thus define the abbreviation $\infty = \mathrm{Val}\, x.\mathbb{1}$.

▶ Remark 8. The condition used in the definition of the sum quantifier is definable by the MSO formula

$$\mathrm{notLast}(x, X) = x \in X \wedge \exists y.(y \in X \wedge x < y),$$

where $x < y$ is an abbreviation for $x \leq y \wedge \neg(y \leq x)$. We can therefore also write

$$[\![\bigoplus\nolimits^{x,X} y.\psi]\!](w, \rho) = \begin{cases} \sum_{i=\rho(x)+1}^{\min\{j \in \rho(X) \mid j > \rho(x)\}-1} [\![\psi]\!](w, \rho[y \to i]) & \text{if } (w, \rho) \models \mathrm{notLast}(x, X) \\ \mathbb{1} & \text{otherwise.} \end{cases}$$

In Lemma 12 we will see that the first order variable x is necessarily also the variable which is quantified by Val. If we define an unrestricted sum quantifier $\bigoplus y.\psi$ by

$$[\![\bigoplus y.\psi]\!](w, \rho) = \textstyle\sum_{i \in \mathrm{dom}(w)} [\![\psi]\!](w, \rho[y \to i]),$$

we can write our restricted sum quantifier as

$$[\![\bigoplus\nolimits^{x,X} y.\psi]\!](w, \rho) =$$
$$[\![\mathrm{notLast}(x, X)\ ?\ \textstyle\bigoplus y.(x < y \wedge \forall z.((x < z \wedge z \leq y) \to \neg z \in X)\ ?\ \psi : 0) : \mathbb{1}]\!](w, \rho).$$

▶ **Example 9.** Consider Example 1 again, i.e. the alphabet $\Sigma = \{\mathsf{demand}, \mathsf{restock}\}$ with the ω-valuation function $\mathrm{Val} = \textsc{Ces}$. Then the formula

$$\varphi = \inf X.\left(\forall z.(z \in X \leftrightarrow P_{\mathsf{restock}}(z))\ ?\ \mathrm{Val}\, x.\bigoplus\nolimits^{x,X} y.1 : \infty\right)$$

describes the average total demand between two restocks. We recall that ∞ is simply an abbreviation for the formula $\mathrm{Val}\, x.\mathbb{1}$. As in Example 1, if we take Inf or Sup for the valuation function, the formula above describes the lowest or highest demand ever encountered.

▶ **Lemma 10** (Consistency Lemma). *Let $\varphi \in \mathrm{mMSO}(\Sigma, \mathrm{Val})$ and $\mathcal{V}$ be a finite set of variables with $\mathcal{V} \supseteq \mathrm{Free}(\varphi)$.*

(i) *For any valid $(w, \rho) \in \Sigma_{\mathcal{V}}^{\omega}$ we have $[\![\varphi]\!]_{\mathcal{V}}(w, \rho) = [\![\varphi]\!](w, \rho{\restriction_{\mathrm{Free}(\varphi)}})$.*

(ii) *$[\![\varphi]\!]$ is MC-recognizable if and only if $[\![\varphi]\!]_{\mathcal{V}}$ is MC-recognizable.*

The following lemma shows that the use of an unrestricted sum quantifier leads to not MC-recognizable series.

▶ **Lemma 11.** *Consider the unrestricted sum quantifier from Remark 8*

$$[\![\bigoplus y.\psi]\!](w, \rho) = \textstyle\sum_{i \in \mathrm{dom}(w)} [\![\psi]\!](w, \rho[y \to i]),$$

the ω-valuation function Val *defined by*

$$\mathrm{Val}((z_i)_{i \geq 0}) = \begin{cases} \sum_{i=0}^{\infty} z_i & \textit{if this sum converges} \\ \infty & \textit{otherwise} \end{cases}$$

and the alphabet $\Sigma = \{a, b\}$. Then for the almost Boolean formula

$$\psi = y \leq x \wedge \forall z.(z \leq x \to P_a(z))\ ?\ -1 : 0,$$

the formula $\varphi = \mathrm{Val}\, x.\bigoplus y.\psi$ is not Val*-MC-recognizable.*

Proof (sketch). One easily checks that

$$[\![\varphi]\!](w) = \begin{cases} -\frac{m(m+1)}{2} & \text{if } w = a^m b w' \text{ for some } w' \in \Sigma^\omega \\ \infty & \text{if } w = a^\omega. \end{cases}$$

The idea is now that with only finitely many transitions, and therefore only finitely many different weights, this quadratic growth cannot be realized if only transitions up to the first b in each word influence the weight of the runs. But once the automaton has read this first b, it cannot distinguish between the words anymore. Under appropriate assumptions, we can therefore combine runs from different words to obtain a contradiction. ◀

The next lemma shows that the first order variable x provided to the sum quantifier is necessarily the variable that Val quantifies.

▶ **Lemma 12.** *Consider the ω-valuation function* Val *defined by*

$$\mathrm{Val}((z_i)_{i\geq 0}) = \begin{cases} \frac{1}{z_0} & \text{if } 0 < z_0 = z_1 = z_2 = \dots \\ -1 & \text{otherwise.} \end{cases}$$

and the alphabet $\Sigma = \{a\}$. We define the abbreviation

$$(y = x + 1) = x \leq y \wedge \neg(y \leq x) \wedge \forall z.(z \leq x \vee y \leq z).$$

Then for the Boolean formula

$$\beta(X) = \forall x_1.\forall x_2.((x_1 \in X \wedge x_2 = x_1 + 1) \to \neg(x_2 \in X)),$$

the formula $\varphi = \inf X.\inf z.\left(\beta(X)\ ?\ \mathrm{Val}\, x.\bigoplus^{z,X} y.1 : \infty\right)$ is not Val*-MC-recognizable.*

Proof (sketch). One can check that $[\![\varphi]\!](a^\omega) = 0$. For a BMCA $\mathcal{A}$ realizing this series, the weight-sequence associated to each run has to be constant, and there must be a sequence of runs such that this constant grows arbitrarily large. The latter fact can be exploited to show that there must be a run whose associated weight-sequence is not constant, which leads to the contradiction $[\![\mathcal{A}]\!](a^\omega) = -1$. ◀

5 The main result

In this section, we want to show that the MC-recognizable series coincide with the series definable by monitor MSO formulas from our logic. In Lemma 14, we show how a given MMCA can be described by a monitor MSO formula. To show that every series definable by a monitor MSO formula is also MC-recognizable, we show by induction on the structure of the formula how to construct an MMCA with the same behavior as the formula. We first formulate our main theorem.

▶ **Theorem 13.** *Let Σ be an alphabet and* Val *an ω-valuation function. A series $S\colon \Sigma^\omega \to \mathbb{R} \cup \{\infty\}$ is* Val*-MC-recognizable if and only if there is a monitor MSO sentence $\varphi \in \mathrm{mMSO}(\Sigma, \mathrm{Val})$ with $[\![\varphi]\!] = S$.*

In the following lemma, we show the first direction, namely how to obtain a formula for a given MMCA.

▶ **Lemma 14.** *For every* Val*-MMCA $\mathcal{A}$, there exists a sentence $\varphi \in \mathrm{mMSO}(\Sigma, \mathrm{Val})$ with $[\![\mathcal{A}]\!] = [\![\varphi]\!]$.*

Proof. For first order variables x and y and second order variables $X_1, \dots, X_k$ we define the MSO formulas

$$\begin{aligned}
\mathrm{first}(x) &= \forall y.x \leq y \\
x < y &= x \leq y \wedge \neg(y \leq x) \\
(y = x+1) &= x < y \wedge \forall z.(z \leq x \vee y \leq z) \\
\mathrm{partition}(X_1, \dots, X_k) &= \forall x. \bigvee_{i=1}^{k} \left(x \in X_i \wedge \bigwedge_{j \neq i} \neg(x \in X_j) \right).
\end{aligned}$$

Now let $\mathcal{A} = (\Sigma, Q, I, \mathcal{F}, \delta, n, \mathrm{Val})$ be an n-MMCA. For every $(p, a, q, \mathbf{u}) \in \delta$ we choose a second order variable $X_{(p,a,q,\mathbf{u})}$ and with $k = |\delta|$ we fix a bijection $v\colon \{1, \dots, k\} \to \delta$. For $i \in \{1, \dots, k\}$ we write X_i for $X_{v(i)}$ and $\bar{X}$ for $(X_1, \dots, X_k)$. Furthermore, we fix second order variables $Y_1, \dots, Y_n$ and write $\bar{Y}$ for $(Y_1, \dots, Y_n)$. For $j \in \{1, \dots, n\}$ and $\star \in \{s, t\}$ we abbreviate

$$(u^j(x) = \star) = \bigvee_{\substack{(p,a,q,\mathbf{u}) \in \delta \\ u^j = \star}} x \in X_{(p,a,q,\mathbf{u})}.$$

Intuitively, we use the variables $\bar{X}$ to encode runs, i.e. by assigning the transition $v(i)$ to every position in X_i. The variables $\bar{Y}$ are used to mark the starts and stops of the counters in $\bar{X}$. In the following, we define the MSO formula $\mathrm{muller}(\bar{X})$ which checks that $\bar{X}$ encodes a run of $\mathcal{A}$ satisfying the Muller acceptance condition, and the MSO formula $\mathrm{valid}(\bar{X})$ which checks that $\bar{X}$ encodes an accepting run. The MSO formula $\mathrm{valid}^*(\bar{X}, \bar{Y})$ asserts that the positions in $\bar{Y}$ conform to the starts and stops of the counters in $\bar{X}$. The precise formulas are as follows.

$$\begin{aligned}
&\mathrm{matched}(\bar{X}) = \bigwedge_{(p,a,q,\mathbf{u}) \in \delta} \forall x. \big(x \in X_{(p,a,q,\mathbf{u})} \to P_a(x)\big) \\
&\wedge \forall x. \forall y. \left(y = x+1 \to \bigvee_{q \in Q} \left(\bigvee_{(p,a,q,\mathbf{u}),(q,a',p',\mathbf{u}') \in \delta} (x \in X_{(p,a,q,\mathbf{u})} \wedge y \in X_{(q,a',p',\mathbf{u}')}) \right) \right) \\
&\mathrm{muller}(\bar{X}) = \mathrm{partition}(\bar{X}) \wedge \mathrm{matched}(\bar{X}) \wedge \exists x. \left(\mathrm{first}(x) \wedge \bigvee_{\substack{(p,a,q,\mathbf{u}) \in \delta \\ p \in I}} x \in X_{(p,a,q,\mathbf{u})} \right) \\
&\wedge \bigvee_{F \in \mathcal{F}} \left(\exists x. \forall y. x \leq y \to \left(\left(\bigvee_{\substack{(p,a,q,\mathbf{u}) \in \delta \\ q \in F}} y \in X_{(p,a,q,\mathbf{u})} \right) \right. \right. \\
&\qquad \left. \left. \wedge \bigwedge_{q \in F} \exists z. \left(y \leq z \wedge \bigvee_{(p,a,q,\mathbf{u}) \in \delta} z \in X_{(p,a,q,\mathbf{u})} \right) \right) \right) \\
&\mathrm{valid}(\bar{X}) = \mathrm{muller}(\bar{X}) \wedge \forall x. \exists y. (x \leq y \wedge \bigvee_{j=1}^{n} u^j(y) = s) \wedge \\
&\bigwedge_{j=1}^{n} \forall x. \Big(\big((u^j(x) = s) \to \exists y. (x < y \wedge u^j(y) = t \wedge \forall z. ((x < z \wedge z < y) \to \neg(u^j(z) = s))) \big) \\
&\qquad \wedge \big((u^j(x) = t) \to \exists y. (y < x \wedge u^j(y) = s \wedge \forall z. ((y < z \wedge z < x) \to \neg(u^j(z) = t))) \big) \Big)
\end{aligned}$$

$$\mathrm{valid}^*(\bar{X},\bar{Y}) = \mathrm{valid}(\bar{X}) \wedge \bigwedge_{j=1}^{n} \forall x.(x \in Y_j \leftrightarrow (u^j(x) = s \vee u^j(x) = t)).$$

For $(p,a,q,\mathbf{u}) \in \delta$ we let $\mathrm{wt}_j(p,a,q,\mathbf{u}) = u^j$ and for $i \in \{1,\dots,k-2\}$ and $j \in \{1,\dots,n\}$ define inductively

$$\begin{aligned}
\psi^j_{k-1} &= (y \in X_{k-1} \,?\, \mathrm{wt}_j(v(k-1)) : \mathrm{wt}_j(v(k))) \\
\psi^j_i &= \left(y \in X_i \,?\, \mathrm{wt}_j(v(i)) : \psi^j_{i+1}\right) \\
\zeta_{n+1} &= \mathbb{1} \\
\zeta_j &= \left((u^j(x) = s) \,?\, \bigoplus\nolimits^{x,Y_j} y.\psi^j_1 : \zeta_{j+1}\right).
\end{aligned}$$

Then with $\varphi = \inf \bar{X}.\inf \bar{Y}.(\mathrm{valid}^*(\bar{X},\bar{Y}) \,?\, \mathrm{Val}\, x.\zeta_1 : \infty)$, we have $[\![\mathcal{A}]\!] = [\![\varphi]\!]$. The formula ψ^j_1 evaluates to the weight for counter j in the transition at position y, i.e. it is $\mathrm{wt}_j(v(i))$ iff y is in X_i. The formula ζ_1 evaluates to the output of the counter started at position x in the run encoded by $\bar{X}$. More precisely, ζ_1 evaluates to $\bigoplus^{x,Y_j} y.\psi^j_1$ if counter j is started at position x, and to $\mathbb{1}$ if no counter is started at x. Finally, the formula φ takes the infimum over the weights of all "runs" $\bar{X}$, in the sense that assignments to $\bar{X}$ and $\bar{Y}$ only influence the value of φ if $\bar{X}$ encodes an accepting run and $\bar{Y}$ mirrors its counter starts and stops. ◀

The remainder of this section is dedicated to show the converse, namely that for every monitor MSO formula there is an MMCA with the same behavior as the formula.

▶ **Lemma 15.** *Let β be an MSO formula and $\varphi_1,\varphi_2 \in \mathrm{mMSO}(\Sigma,\mathrm{Val})$ such that $[\![\varphi_1]\!]$ and $[\![\varphi_2]\!]$ are MC-recognizable. Then with $\varphi = \beta \,?\, \varphi_1 : \varphi_2$, the series $[\![\varphi]\!]$ is also MC-recognizable.*

Proof. Let $\mathcal{V} = \mathrm{Free}(\varphi)$. Then we have $\mathrm{Free}(\varphi_1) \subseteq \mathcal{V}$ and $\mathrm{Free}(\varphi_2) \subseteq \mathcal{V}$ and hence by Lemma 10 $[\![\varphi_1]\!]_\mathcal{V}$ and $[\![\varphi_2]\!]_\mathcal{V}$ are MC-recognizable. Due to $\mathrm{Free}(\beta) \subseteq \mathcal{V}$, the classical Büchi theorem tells us that both $\mathcal{L}_\mathcal{V}(\beta)$ and $\mathcal{L}_\mathcal{V}(\neg\beta)$ are ω-recognizable. Hence by Lemma 5 and Lemma 6, $[\![\varphi]\!] = \min(\mathcal{L}_\mathcal{V}(\beta) \cap [\![\varphi_1]\!]_\mathcal{V}, \mathcal{L}_\mathcal{V}(\neg\beta) \cap [\![\varphi_2]\!]_\mathcal{V})$ is also MC-recognizable. ◀

▶ **Lemma 16.** *Let $\varphi_1,\varphi_2 \in \mathrm{mMSO}(\Sigma,\mathrm{Val})$ be such that $[\![\varphi_1]\!]$ and $[\![\varphi_2]\!]$ are MC-recognizable. Then $\varphi = \min(\varphi_1,\varphi_2)$, the series $[\![\varphi]\!]$ is also MC-recognizable.*

Proof. Let $\mathcal{V} = \mathrm{Free}(\varphi_1) \cup \mathrm{Free}(\varphi_2)$, then by Lemma 10, $[\![\varphi_1]\!]_\mathcal{V}$ and $[\![\varphi_2]\!]_\mathcal{V}$ are also MC-recognizable. Hence by Lemma 5, $[\![\varphi]\!] = \min([\![\varphi_1]\!]_\mathcal{V}, [\![\varphi_2]\!]_\mathcal{V})$ is also MC-recognizable. ◀

▶ **Lemma 17.** *Let $\varphi \in \mathrm{mMSO}(\Sigma,\mathrm{Val})$ such that $[\![\varphi]\!]$ is MC-recognizable. Then $\inf x.\varphi$ and $\inf X.\varphi$ are also MC-recognizable.*

Proof. We show the lemma for $\inf x.\varphi$. The proof for $\inf X.\varphi$ is similar. Let $\mathcal{V} = \mathrm{Free}(\inf x.\varphi)$, then $x \notin \mathcal{V}$. We now consider the homomorphism $h\colon \Sigma^\omega_{\mathcal{V}\cup\{x\}} \to \Sigma^\omega_\mathcal{V}$, which erases the x-row. Then for any $(w,\rho) \in \Sigma^\omega_\mathcal{V}$, we have that

$$[\![\inf x.\varphi]\!]_\mathcal{V}(w,\rho) = \inf\{[\![\varphi]\!]_{\mathcal{V}\cup\{x\}}(w,\rho[x \to i]) \mid i \geq 0\} = h([\![\varphi]\!]_{\mathcal{V}\cup\{x\}})(w,\rho).$$

As $\mathrm{Free}(\varphi) \subseteq \mathcal{V} \cup \{x\}$, Lemma 10 shows that $[\![\varphi]\!]_{\mathcal{V}\cup\{x\}}$ is MC-recognizable and therefore by Lemma 4 (i) the series $[\![\inf x.\varphi]\!]_\mathcal{V}$ is MC-recognizable as well. ◀

▶ **Lemma 18.** *Let ψ be an almost Boolean formula and $\mathcal{V} \supseteq \mathrm{Free}(\psi)$. Then there are MSO formulas $\beta_1,\dots,\beta_n$ and weights $z_1,\dots,z_n \in \mathbb{Z}$ such that $\mathrm{Free}(\psi) = \bigcup_{i=1}^n \mathrm{Free}(\beta_i)$, $N_\mathcal{V} = \bigcup_{i=1}^n \mathcal{L}_\mathcal{V}(\beta_i)$, for $i \neq j$ we have $\mathcal{L}_\mathcal{V}(\beta_i) \cap \mathcal{L}_\mathcal{V}(\beta_j) = \emptyset$ and for $(w,\rho) \in N_\mathcal{V}$ we have $[\![\psi]\!]_\mathcal{V}(w,\rho) = z_i$ if and only if $(w,\rho) \in \mathcal{L}_\mathcal{V}(\beta_i)$.*

Proof. For $\psi = k$ with $k \in \mathbb{Z}$, we choose β_1 as any tautology, for example $\beta_1 = \exists x.x \leq x$, and $z_1 = k$.

For $\psi = \beta \mathbin{?} \psi_1 : \psi_2$ we assume that the lemma is true for ψ_1 with $\beta_1^{(1)}, \dots, \beta_{n_1}^{(1)}$ and $z_1^{(1)}, \dots, z_{n_1}^{(1)}$ and for ψ_2 with $\beta_1^{(2)}, \dots, \beta_{n_2}^{(2)}$ and $z_1^{(2)}, \dots, z_{n_2}^{(2)}$. Then for ψ we choose $\beta_1, \dots, \beta_{n_1+n_2}$ and $z_1, \dots, z_{n_1+n_2}$ as follows. For $i \in \{1, \dots, n_1\}$ we set $\beta_i = \beta \wedge \beta_i^{(1)}$ and $z_i = z_i^{(1)}$ and for $i \in \{1, \dots, n_2\}$ we set $\beta_{n_1+i} = \neg\beta \wedge \beta_i^{(2)}$ and $z_{n_1+i} = z_i^{(2)}$. ◀

▶ **Lemma 19.** *Let ζ be an x-summing formula and $\mathcal{V} \supseteq \mathrm{Free}(\zeta)$. Then there are MSO formulas $\beta_1, \dots, \beta_n$ and formulas $\zeta_1, \dots, \zeta_n$ with $\zeta_i = \bigoplus^{x,Y_i} y.\psi_i$ for some almost Boolean formula ψ_i such that $\mathrm{Free}(\zeta) = \bigcup_{i=1}^n \mathrm{Free}(\beta_i) \cup \mathrm{Free}(\zeta_i)$, for $i \neq j$ we have $\mathcal{L}_\mathcal{V}(\beta_i) \cap \mathcal{L}_\mathcal{V}(\beta_j) = \emptyset$, for $(w, \rho) \in N_\mathcal{V}$ we have $[\![\zeta]\!]_\mathcal{V}(w, \rho) = [\![\zeta_i]\!]_\mathcal{V}(w, \rho)$ if and only if $(w, \rho) \in \mathcal{L}_\mathcal{V}(\beta_i)$ and if $(w, \rho) \notin \bigcup_{i=1}^n \mathcal{L}_\mathcal{V}(\beta_i)$ then $[\![\zeta]\!](w, \rho) = \mathbb{1}$. We can assume the variables Y_i to be pairwise distinct.*

▶ **Theorem 20.** *Let ζ be an x-summing formula. Then $[\![\mathrm{Val}\, x.\zeta]\!]$ is MC-recognizable.*

Proof (sketch). We adapt and expand an idea from [9]. Let $\beta_1, \dots, \beta_n$ and $\zeta_1, \dots, \zeta_n$ be the formulas we can find for ζ according to Lemma 19. We write $\zeta_i = \bigoplus^{x,Y_i} y.\psi_i$. Then for each $i \in \{1, \dots, n\}$, let $\beta_{i1}, \dots, \beta_{in_i}$ and $z_{i1}, \dots, z_{in_i}$ be the formulas and weights we can find for ψ_i according to Lemma 18.

The proof idea is as follows. For $\mathcal{V} = \mathrm{Free}(\mathrm{Val}\, x.\zeta)$, the mapping $[\![\mathrm{Val}\, x.\zeta]\!]$ assigns values to words from $\Sigma_\mathcal{V}^\omega$. Consider $(w, \rho) \in \Sigma_\mathcal{V}^\omega$. We can interpret each ζ_i as a counter which is stopped and then restarted at the k-th letter of w depending on whether $(w, \rho[x \to k])$ satisfies β_i. As our automata cannot stop and start a single counter at the same time, each counter i will correspond to two counters i and i' in the automaton we construct. The computations of counter i depend on $\beta_{i1}, \dots, \beta_{in_i}$. We extend the alphabet $\Sigma_\mathcal{V}$ by adding two entries for each counter to each letter in $\Sigma_\mathcal{V}$. The entries for counter i can contain an s to indicate the start of the counter, a t to indicate a stop, a number $j \in \{1, \dots, n_i\}$ to indicate that the counter is active and should add z_{ij} to its current value, or a $\perp$ to indicate that the counter is inactive. Let $\tilde{\Sigma}_\mathcal{V}$ be this new alphabet. We show that we can define an ω-recognizable language L over $\tilde{\Sigma}_\mathcal{V}$ which has all information about the counter operations encoded in the word. For example, if $(w, \rho[x \to k]) \models \beta_i$, then in the word $(w, \rho, v) \in \tilde{\Sigma}_\mathcal{V}^\omega$ corresponding to (w, ρ) the entry for counter i in the k-th letter should contain an s. Then if $(w, \rho[x \to k, y \to k+1]) \models \beta_{ij}$, the i-entry of the $k+1$-th letter should contain a j. The precise formulation of this is involved.

When we have shown that the language L is recognizable, we can construct a deterministic Muller automaton $\tilde{\mathcal{A}}$ which recognizes L. Turning $\tilde{\mathcal{A}}$ into an MMCA and applying a projection, we finally obtain the recognizability of $[\![\mathrm{Val}\, x.\zeta]\!]$. ◀

This concludes our induction, and thus the proof of Theorem 13.

6 Conclusion

We introduced a new logic which is expressively equivalent to Quantitative Monitor Automata. Since our proofs are constructive, we immediately obtain the possibility to reduce the satisfiability and equivalence problems of our logic to the emptiness and equivalence problems of Quantitative Monitor Automata. Future work could therefore focus on the investigation of this automaton model, and on the related model of Nested Weighted Automata [3].

References

1 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

2 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. In Michael Kaminski and Simone Martini, editors, *Proc. CSL*, volume 5213 of *LNCS*, pages 385–400. Springer, 2008.

3 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested weighted automata. In *Proc. LICS*, pages 725–737, 2015.

4 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative monitor automata. In Xavier Rival, editor, *Proc. SAS*, volume 9837 of *LNCS*. Springer, 2016.

5 Manfred Droste and Stefan Dück. Weighted automata and logics on graphs. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald T. Sannella, editors, *Proc. MFCS*, volume 9234 of *LNCS*, pages 192–204. Springer, 2015.

6 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380:69–86, 2007.

7 Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata*. Monogr. Theoret. Comput. Sci. EATCS Ser. Springer, 2009.

8 Manfred Droste and Ingmar Meinecke. Weighted automata and weighted MSO logics for average and long-time behaviors. *Inform. Comput.*, 220–221:44–59, 2012.

9 Manfred Droste and George Rahonis. Weighted automata and weighted logics on infinite words. In *Proc. DLT*, volume 4036 of *LNCS*, pages 49–58. Springer, 2006.

10 Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Am. Math. Soc.*, 98(1):21–51, 1961.

11 Zoltán Ésik and Werner Kuich. A semiring-semimodule generalization of ω-regular languages I. *Journal of Automata, Languages and Combinatorics*, 10(2/3):203–242, 2005.

12 Zoltán Ésik and Werner Kuich. A semiring-semimodule generalization of ω-regular languages II. *Journal of Automata, Languages and Combinatorics*, 10(2/3):243–264, 2005.

13 Ina Fichtner. Weighted picture automata and weighted logics. *Theor. Comput. Syst.*, 48(1):48–78, 2011.

14 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.

15 Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*, pages 133–191. Elsevier Science, 1990.

16 Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages (Vol. 3)*, pages 389–455. Springer, 1997.

17 Boris Avraamovich Trakhtenbrot. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 140:326–329, 1961. In Russian.

The Complexity of Quantum Disjointness*

Hartmut Klauck

Centre for Quantum Technologies and Nanyang Technological University, Singapore
hklauck@gmail.com

Abstract

We introduce the communication problem QNDISJ, short for *Quantum (Unique) Non-Disjointness*, and study its complexity under different modes of communication complexity. The main motivation for the problem is that it is a candidate for the separation of the quantum communication complexity classes QMA and QCMA. The problem generalizes the Vector-in-Subspace and Non-Disjointness problems. We give tight bounds for the QMA, quantum, randomized communication complexities of the problem. We show polynomially related upper and lower bounds for the MA complexity. We also show an upper bound for QCMA protocols, and show that the bound is tight for a natural class of QCMA protocols for the problem. The latter lower bound is based on a geometric lemma, that states that every subset of the n-dimensional sphere of measure 2^{-p} must contain an ortho-normal set of points of size $\Omega(n/p)$.

We also study a "small-spaces" version of the problem, and give upper and lower bounds for its randomized complexity that show that the QNDISJ problem is harder than Non-disjointness for randomized protocols. Interestingly, for quantum modes the complexity depends only on the dimension of the smaller space, whereas for classical modes the dimension of the larger space matters.

1998 ACM Subject Classification F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes

Keywords and phrases Communication Complexity, Quantum Proof Systems

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.15

1 Introduction

Communication complexity [30, 23] is a central area in computational complexity and the source of many lower bounds for other computational (nonuniform) models. Because of this much of the research in communication complexity is focused on lower bounds. Most of these lower bound applications employ the lower bound to the Disjointness problem shown by Kalyanasundaram and Schnitger [14] (see also [26, 7] for simpler proofs), or, in the quantum case the lower bound by Razborov [27] (see also [28]).

The present paper is mainly motivated by the following open problem. The complexity class QMA (in the Turing machine world) is the quantum analogue of NP (or rather of MA), namely the class of problems, that can be verified efficiently given a (non-interactive) quantum proof (by a quantum verifier). Similarly, QCMA is the class of problems where a *classical* proof can be verified efficiently by a quantum verifier. Obviously, the relationship between the two classes is highly interesting. This relates to the problem of whether more

* This work is funded by the Singapore Ministry of Education (partly through the Academic Research Fund Tier 3 MOE2012-T3-1-009) and by the Singapore National Research Foundation. Also supported by Majulab UMI 3654.

information can be present in a quantum state than in a classical state (of the same size), in this case based on the capacity to function as a proof. This problem was first suggested by Aharonov and Naveh [5].

Aharonov and Naveh have conjectured that $QCMA = QMA$, because the local Hamiltonian problem is complete for QMA (Turing machine model, [15]), and hence in some sense only a constant number of qubits in a quantum proof need to be touched by the verifier at once, so only localized entanglement seems necessary. Nevertheless such a result has not been established since. On the other hand, obviously we are far away from proving such separations as $QCMA \neq QMA$ for the Turing machine model (one implication would be $P \neq PSPACE$). Aaronson and Kuperberg [3] have shown a separation of the classes relative to a *quantum* oracle, a result that does not imply the usual relativization obstacle to showing that the classes are equal. It remains open whether $QCMA \neq QMA$ relative to some *classical oracle* (which would imply that anyone who wants to show that $QMA = QCMA$ must use non-relativizing techniques). Aaronson and Wigderson [4] have proposed the stronger *algebrization* obstacle to showing complexity theoretic results, and give many examples of such results requiring non-algebrizing techniques. One of their methods is to use separations of complexity classes in communication complexity (see [6]) in order to show algebrization separations. This motivates trying to separate QCMA and QMA in communication complexity (besides the power of quantum proofs being interesting in any model). The main reason why this is preferable over trying to solve the (usually easier) separation in the query complexity model (which would give an oracle separation) is that we have a proper candidate problem for the separation, namely the problem Linear-Space-Distance (LSD) introduced by Raz and Shpilka [25], who show that LSD is QMA-complete (for the communication complexity model).

Due the QMA-completeness of LSD, if there is a separation of QMA from QCMA (in communication complexity), then there is one using LSD. However, the problem is awkward in the sense that the 1-inputs (resp. the 0-inputs) do not form a manifold. This complicates reasoning about the problem, which is of a geometrical nature, and is best studied in its non-discretized version. We propose a subproblem of LSD that has the following nice properties: its input sets are Riemannian manifolds and there are nontrivial protocols for various modes of communication for the problem, exhibiting limits on lower bounds that can be shown, guiding our intuition.

The problem we propose is called Quantum (Unique) Non-Disjointness (short QNDISJ). Informally, the 0-inputs of QNDISJ are pairs (W_A, W_B) of subspaces of $\mathbb{R}^n$ that are orthogonal to each other, while the 1-inputs are pairs such that $W_A \cap W_B$ has dimension 1, and the spaces are orthogonal outside of their intersection.

We view QNDISJ as a natural quantum analogue of the Disjointness problem. This works as follows: For a fixed ortho-normal basis of $\mathbb{R}^n$ a subset $x \subseteq \{1, \ldots, n\}$ can be identified with a subspace (by taking the span of the basis vectors indexed by x). Hence, if Alice and Bob have sets of size s, t respectively that are disjoint, then their inputs correspond to two orthogonal subspaces, i.e., a 0-inputs of QNDISJ. If the sets intersect on 1 element, then the two subspaces have a 1-dimensional intersection, and they are (also) a 1-input. The difference between (the complement) of Disjointness and QNDISJ is then that there is no fixed basis for the latter problem, but rather that Alice and Bob know their own subspaces, but no good basis of the whole space, in which the intersecting (unit) vector is a basis vector.

Another problem of which QNDISJ is a generalization is the Vector-in-Subspace problem [22, 24, 16], which is the same problem, only that Alice has a 1-dimensional subspace, and Bob an $n/2$-dimensional subspace. For this problem Klartag and Regev give a $\Omega(n^{1/3})$ lower

bound on the randomized communication complexity and Raz gives a randomized $O(\sqrt{n})$ upper bound, with the upper bound likely tight. The quantum complexity of this problem is $O(\log n)$.

We explore the communication complexity of QNDISJ for the following modes of communication: QMA, QCMA, MA, randomized, quantum. We also consider the version of the problem, where the dimensions of the subspaces are s, t with $s \leq t \leq n/2$. We give (almost) matching upper and lower bounds for randomized, quantum, QMA-protocols in the case $s = t = n/4$. We give non-trivial bounds for smaller spaces as well (see the table later on). One interesting conclusion is that for the quantum modes (Q, QCMA, QMA) the complexity depends (up to small factors) only on s, the dimension of the smaller space, whereas for the classical modes (R, MA) the complexity of the larger space matters. We also give a lower bound for QCMA protocols for QNDISJ under a restriction on the protocols. This restriction is that the proof (sent by Merlin, who sees both subspaces) depends on the intersection of the subspaces *only* (and can hence be viewed as an (arbitrary) subset of the sphere). We prove a geometric lemma about large subsets of the sphere that allows us to reduce a smaller instance of Disjointness to the "leftover" problem of QNDISJ, once one of the classical proofs is fixed (namely the problem of accepting all 1-inputs for which this proof is good, while rejecting all 0-inputs).

Our restriction seems natural, because it is difficult to imagine how other information from Merlin could be useful to Alice (who knows her space, just not the intersection) or Bob. So our conditional bound can be seen as some evidence that quantum proofs are really more powerful than classical proofs. We note that a separation between QMA- and QCMA-communication complexity in the one-way model is known [21], but that result has no bearing on algebrization and the analogous problem for Turing machines.

2 Organization of the Paper

This is an extended abstract. In the next section we give a formal definition of $QNDISJ_{s,t}$. In Section 4 we have a table describing our results and define the property under which our conditional QCMA lower bound holds. In Section 5 we have preliminaries, and in Section 6 a rough overview of our main techniques. See the full paper for more formal statements and for proofs.

3 Definition of the Problem

Denote by $S^{n-1} = \{x \in \mathbb{R}^n : ||x|| = 1\}$ the sphere. The Grassmannian is $G_{n,m} = \{V : V \subseteq \mathbb{R}^n$ and V is an m-dimensional subspace$\}$. We define our main problem on two manifolds. Throughout the paper we will fix n as the dimension of the underlying space when talking about our problem. s, t are the dimensions of Alice's and Bob's subspace and we will always have $s \leq t \leq n/2$.

▶ **Definition 1.** The *orthogonal manifold* $O_{s,t}$ is the set of pairs of subspaces W_A, W_B, where W_A is an s-dimensional subspace of $\mathbb{R}^n$, W_B a t-dimensional subspace of $\mathbb{R}^n$, and $W_A \perp W_B$.

▶ **Definition 2.** The *intersection manifold* $I_{s,t}$ is the set of pairs of subspaces W_A, W_B, where W_B is an s-dimensional subspace of $\mathbb{R}^n$, W_B a t-dimensional subspace of $\mathbb{R}^n$, and W_A, W_B intersect in a 1-dimensional subspace spanned by some vector z. Furthermore $(W_A \cap z^{\perp}) \perp (W_B \cap z^{\perp})$.

Table 1 Complexity of $QNDISJ_{s,t}$ for $s \leq t$.

Mode	Upper Bound	Lower Bound
QMA	$O(\log n)$	$\Omega(\sqrt{\log t})$ 1)
R	$O(\min\{s\sqrt{t}, n \log n\})$	$\Omega(s(t/s)^{1/3})$
Q	$O(\sqrt{s} \log n)$	$\Omega(\sqrt{s})$
MA	$O(\sqrt{s\sqrt{t}})$	$\Omega(t^{1/6})$
QCMA	$O(s^{1/3} \log n)$	$\Omega(s^{1/3})$ (*)

1) The lower bound becomes $\Omega(\log t)$, if Alice and Bob do not share entanglement.
(*) This lower bound is conditional on assumption (*) about protocols (see below).

▶ **Definition 3.** The problem $QNDISJ_{s,t}$ is a partial function. The set of 1-inputs is $I_{s,t}$. The set of 0-inputs is $O_{s,t}$. For all other pairs of subspaces the function is undefined.

When we don't indicate s, t, then $s = t = \frac{n}{4}$. We leave n, the dimension of the underlying space, implicit.

To provide some insight as to the name of the problem, consider Razborov's hard distribution for the Disjointness problem [26]: disjoint inputs are disjoint pairs of sets of size $n/4$, intersecting inputs are pairs of sets of size $n/4$ that have intersection size 1. If we fix an ortho-normal basis of $\mathbb{R}^n$, we can view each set as picking $n/4$ basis vectors and hence consider Alice and Bob's inputs as subspaces of dimension $n/4$. The subspaces are orthogonal for disjoint sets and are in the intersection manifold for intersecting inputs.

The difference between Non-disjointness and $QNDISJ$ is that Alice and Bob *do not know a good basis* for the whole space. Alice knows her space, and she can find a basis for her space, but the intersecting vector is a basis vector only in a hidden basis neither she nor Bob know. The situation is as if someone would take the Non-disjointness example above, and apply a secret unitary transformation to the bases of the subspaces, and then only hand the transformed basis of W_A to Alice and only the transformed basis of W_B to Bob.

The Vector-in-Subspace problem is $QNDISJ_{1,n/2}$. Similar to the reduction from Non-Disjointness above, there is a reduction from $INDEX_n$ (see Section 5.1) to this problem. Furthermore note that $QNDISJ_{1,1}$ is a somewhat natural real version of the Equality problem.

We don't explicitly consider discretized versions of QNDISJ in this paper. Obviously one can easily encode an approximation of the problem by providing Alice and Bob with a basis of their subspaces rounded to precision $1/poly(n)$.

4 Results

Our results concerning the communication complexity of $QNDISJ_{s,t}$ are collected in the following table. Alice receives the s-dimensional subspace, Bob the t-dimensional subspace, and $s \leq t$. The bounds for QMA, Q, R are tight up to logarithmic factors in the case $s = t = n/4$. Note that unless mentioned, we allow entanglement shared by Alice and Bob for the lower bounds (but don't use entanglement in our protocols).

We now explain the lower bound for QCMA protocols which holds under a certain assumption on the protocols. It is natural to assume that the prover should send information about the intersection to one of the players. The intersection (in the case of a 1-input) is a 1-dimensional subspace, and hence the prover should probably send some information about

the (normalized) vector that spans it. A message from the prover would then correspond to a subset of the unit sphere, e.g. could be a spherical cap (or something else).

Our assumption about Merlin's proof is hence that the prover sends messages that correspond to subsets P of the unit sphere. All 1-inputs, where $W_A \cap W_B \in P$ should be accepted on such a proof with high probability.

What this means is that the prover communicates arbitrary information about the intersection, *but nothing more*. One more point is: which sphere? There are 3 possibilities: the sphere in $\mathbb{R}^n$, the sphere in W_A, and the sphere in W_B. Each of these is fine regarding our assumption. Indeed in the case of $s = t = n/4$ this difference does not matter much. For smaller spaces it is more convenient to use the sphere in W_A (assuming that Merlin sends his message to Alice).

We now make the assumption formal. in a QCMA-protocol the prover Merlin sends a classical message (the proof) to Alice, after which Alice and Bob verify the proof, using quantum communication. See Section 5.1 for the definition. We can identify Merlin's proof message with the subset of 1-inputs, which will be accepted with probability at least 2/3 by the verifier(s) when given this proof. So besides the actual message, we also refer to said subset of the 1-inputs as a proof. Hence, with a proof length of p we get a set of at most 2^p proofs that cover the set of 1-inputs. In particular, under any given distribution, the average proof must have measure at least 2^{-p} on the 1-inputs. For QNDISJ, the set of 1-inputs is the intersection manifold $I_{n/4,n/4}$.

A proof that satisfies our assumption also corresponds to a subset of the sphere S^{n-1}.

▶ **Definition 4.** A subset P' of the intersection manifold $I_{n/4,n/4}$ is called *intersectional*, if there is a subset $P \subseteq S^{n-1}$ such that $P' = \{(W_A, W_B) : (W_A, W_B) \in I_{n/4,n/4}$ and $(W_A \cap W_B)$ is spanned by some $z \in P\}$.

▶ **Definition 5.** A QCMA-protocol for a function f satisfies assumption (*), if it is a valid QCMA protocol, and if there is a strategy for Merlin, in which he can convince Alice and Bob to accept with probability at least 2/3 for every 1-input by using intersectional proofs only.

5 Preliminaries

5.1 Communication Complexity

We assume familiarity with the standard modes of communication complexity, and use $R(f)$ to denote the randomized communication complexity (for simplicity we choose public coin randomness), and $Q(f)$ to denote the (entanglement assisted) quantum communication complexity, with error 1/3 in each case. For details we refer to [29].

Proof systems have been introduced into communication complexity in [6], and studied further in e.g. [18, 25, 1, 4, 19, 20, 10, 12]. We now define the main models involving a prover that we use.

▶ **Definition 6.** In a Merlin Arthur protocol a prover (Merlin) sends a string to Alice, who then communicates with Bob. Merlin sees both inputs x, y while Alice sees only x and Bob only y. The goal is to compute a Boolean function $f(x, y)$. Alice and Bob have shared randomness that is invisible to Merlin. Such a protocol is *sound*, if all 0-inputs are accepted with probability at most 1/3 given any message of Merlin, and *complete*, if all 1-inputs are accepted with probability at least 2/3 for some message of Merlin.

The cost of the protocol is the total communication length used, in the worst case, i.e., the total length of the messages sent by Merlin, Alice and Bob. The Merlin Arthur

communication complexity of f is the minimum complexity over all sound and complete protocols for f. It is denoted by $MA(f)$.

If we fix the proof length to some parameter p, then it is natural to only count the length of the communication among Alice and Bob. We denote the MA complexity with fixed proof length p by $MA^p(f)$.

It is easy to see that for all $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ we have $MA^n(f) = O(1)$. One can also establish easily by a counting argument that for most such f we have $MA(f) = \Theta(n)$. It is open to prove a larger lower bound than $\Omega(\sqrt{n})$ for any explicit function.

We now turn to quantum versions of this model.

▶ **Definition 7.** In a QCMA protocol Merlin sends a classical message to Alice, while Alice and Bob can communicate using quantum messages and may hold parts of an arbitrary entangled state (not accessible by Merlin). The remaining description is as for MA-protocols. The QCMA-complexity of f is denoted by $QCMA(f)$. If we restrict the length of the proof to p, then we count only the length of the communication between Alice and Bob. The corresponding complexity measure is denoted by $QCMA^p(f)$.

In a QMA protocol Merlin may send a quantum message to Alice. Otherwise the definition is as above. The notations are $QMA(f)$ and $QMA^p(f)$.

Finally, we consider the model where Merlin, Alice, Bob additionally share a classical public coin. Merlin then sends a quantum message to Alice, and Alice and Bob communicate with quantum messages (but have no shared entanglement). This can be called Arthur Quantum Merlin Arthur, because the shared public coin could be seen as a challenge to Merlin. The complexity is denoted $AQMA(f)$.

We define AQMA protocols because we can precisely capture the complexity of $QNDISJ_{s,t}$ in this model (see the full paper, the bound is $\Theta(\log t)$).

Besides the problem $QNDISJ_{s,t}$, and Vector-in-Subspace= $QNDISJ_{1,n/2}$ we also consider the following problems:

▶ **Definition 8.** The disjointness problem $DISJ_{s,t}$, or short DISJ in case $s = t = n/4$, is the problem where Alice gets $x \in \{0,1\}^n$, Bob gets $y \in \{0,1\}^n$, and they should accept if and only if $\vee_i(x_i \wedge y_i) = 0$ (and we have $|x| = s$ and $|y| = t$). The complement of this problem is NDISJ.

In the problem $INDEX_n$ Alice receives $x \in \{0,1\}^n$, Bob receives $i \in [n]$, and the required output is x_i. This is (more or less) equivalent to $NDISJ_{n/2,1}$.

In the problem IP_n Alice and Bob receive $x, y \in \{0,1\}^n$, and the required output is $\bigoplus_i x_i \wedge y_i$.

One of the main techniques of quantum computing is amplitude amplification, a generalization of Grover search [8, 13].

▶ **Fact 9.** *Suppose we are given a quantum protocol that, depending on the input x, y, either accepts with probability δ, or never accepts, with communication c, and hence computes a function $f(x, y)$ with large, but one-sided error.*

Then there is a quantum protocol for f with communication $O(\sqrt{1/\delta} \cdot c)$ and constant (one-sided) error.

▶ **Definition 10.** A *reduction* from a problem $f : X \times Y \to \{0,1\}$ to a problem $g : U \times V \to \{0,1\}$ consists of two mappings $\rho : X \to U$ and $\tau : Y \to V$ such that

$g(\rho(x), \tau(y)) = f(x, y)$ for all x, y.

Clearly, when there is a reduction from f to g, then for every mode of communication complexity g is at least as hard as f, because Alice and Bob can perform arbitrary local computations for free.

Finally, we note that by standard techniques we may assume that all amplitudes in all our quantum protocols are real.

5.2 Spherical Caps

Let S^{n-1} denote the $(n-1)$-dimensional sphere (i.e., the set of unit vectors in $\mathbb{R}^n$). By μ we usually denote the uniform distribution on a manifold, i.e., the Haar measure. A *spherical cap* centered on a unit vector c is the set $C_\epsilon^c = \{w \in S^{n-1} : \langle v, c\rangle \geq \epsilon\}$, where we leave n implicit and $\epsilon \geq 0$. If we care only about the area or measure of a cap, we drop the center c, because the caps are isomorphic to each other.

We are interested in the measure $\mu(C_\epsilon)$. For this we should know the area of both the sphere and a spherical cap. Let A_{n-1} denote the area of S^{n-1}. An explicit formula is $A_{n-1} = \frac{2\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2})}$. This is maximized (over integers) at $n = 7$. It can be shown that $A_{n-2}/A_{n-1} \geq \frac{\sqrt{n}}{4}$ for all $n \geq 2$.

Next we state upper and lower bounds on the area or measure of a cap C_ϵ in S^{n-1}.

▶ **Lemma 11.** *Let $\epsilon \leq 1/2$.*

1. *The measure $\mu(C_\epsilon)$ is at most $e^{-n\epsilon^2/2}$.*
2. *The area of C_ϵ is at least $A_{n-2}e^{-n\epsilon^2}/(8n\epsilon)$.*
3. *$\mu(C_\epsilon) \geq e^{-n\epsilon^2}/(32\epsilon\sqrt{n})$.*
4. *If $v \in S^{n-1}$ is a fixed vector, and w is randomly drawn from S^{n-1} under μ, then the probability that $\langle v, w\rangle^2 \geq \frac{k}{n}$ is $\leq 2e^{-k/2}$ and $\geq e^{-k}/(16\sqrt{k})$ for all $1 \leq k \leq n/4$.*

5.3 Concentration of Measure

We now consider projecting random unit vectors on larger subspaces. Unsurprisingly, the larger the subspace gets, the better the concentration of measure is. First note, that when a random unit vector from $\mathbb{R}^n$ is projected onto a fixed subspace of dimension t, then the expected squared projection length is t/n. The following bounds can be found in [11].

▶ **Fact 12.** *Let v be a uniformly random vector from S^{n-1}, W a fixed t-dimensional subspace of $\mathbb{R}^n$, and L denote $||Proj_W v||^2$.*

1. *For $0 < \beta < 1$: $Prob(L \leq (1-\beta)\frac{t}{n}) \leq e^{-t\beta^2/4}$.*
2. *For $0 < \beta < 1$: $Prob(L \geq (1+\beta)\frac{t}{n}) \leq e^{-t\beta^2/8}$.*

We also state a version of the Johnson-Lindenstrauss Theorem, for inner products, see [11].

▶ **Fact 13.** *Let $0 < \epsilon < 0.5$, $n, m > 0$ integers and k such that $k \geq 64/\epsilon^2 \cdot \ln m$. Then for any set $\{v_1, \ldots v_m\} \subseteq \mathbb{R}^n \cap S^{n-1}$ a random projection $g : \mathbb{R}^n \to \mathbb{R}^k$ plus re-normalization (together a mapping f) has the property that for all i, j*

$$\langle v_i|v_j\rangle - \epsilon \leq \langle f(v_i)|f(v_j)\rangle \leq \langle v_i|v_g\rangle + \epsilon.$$

5.4 Sampling by Equators

This is the core technical result from [16].

▶ **Fact 14.** *Let $A \subseteq S^{n-1}$ be a set of measure $\mu(A) \geq 2^{-p}$. Let $v \in S^{n-1}$ be a uniformly random vector from the unit sphere, and $v^{\perp} \subseteq \mathbb{R}^n$ the corresponding uniformly random subspace of dimension $n-1$ orthogonal to v. For any $\frac{p+1}{n} < k < 1$ we have*

$$Prob[|\frac{\mu_{v^{\perp}}(v^{\perp} \cap A)}{2^{-p}} - 1| \geq k] \leq e^{-\gamma n k/(p+1)},$$

where $\gamma > 0$ is a constant, and $\mu_{v^{\perp}}$ is the uniform measure on $S^{n-1} \cap v^{\perp}$.

5.5 Nets on the Sphere

A reasonable short proof of intersection for QNDISJ is the nearest center (to the intersection) of a cap in an ϵ-net consisting of spherical caps on the sphere. For us ϵ (usually the maximum distance between any vector and the nearest cap center) will be much larger than in the standard literature about ϵ-nets, i.e., ϵ will be close to $\sqrt{2}$. Therefore we prefer to simply call a set of vectors such that the union of caps around them covers the sphere a *net*. For the matter of quantum measurements, one can also allow the union of caps and corresponding anti-caps as elements of a net. An anti-cap is simple the set $\{-v : v \in C_{\epsilon}^{c}\}$. Recall that for C_{ϵ} we use the inner product between the cap center and the vectors in the cap as the defining closeness parameter.

▶ **Lemma 15.** *For $1 \leq p \leq n/4$ there is a set M of $20e^{2p}n^2$ vectors such that for every vector $v \in S^{n-1}$ there is a vector $w \in M$ with $\langle v, w \rangle^2 \geq \frac{p}{n}$.*

6 Techniques

Here we briefly sketch the main ideas in the paper.

6.1 QMA

There is a simple protocol with complexity $O(\log n)$: For a 1-input (W_A, W_B) Merlin sends a unit vector in the intersection $W_A \cap W_B$ as a quantum state to Alice. Alice measures this with an observable containing W_A. If the measurement does not yield W_A as the result, she rejects. Otherwise she sends the measured state to Bob, who measures with an observable containing W_B, and accepts iff the result is W_B.

We explore lower bounds, and are able to show a lower bound of $\Omega(\sqrt{\log t})$ via a reduction from the inner product function $IP_{\log t}$ to $QNDISJ_{1,t}$. We can get rid of the square root if we don't allow entanglement between Alice and Bob via a reduction from a random function $f : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$ for $m = \log t$ together with a proof that $QMA(f)$ is $\Theta(m)$ with high probability.

The $\log t$ upper bound is tight, if we allow a public coin to be shared by Alice, Bob, and Merlin. This holds via dimension reduction with the Johnson-Lindenstrauss Theorem (Fact 13). Without the public coin it remains open whether there is a better upper bound than $\log n$. For very small s, t we can use the randomized protocol (described below) to beat this bound.

6.2 Randomized

For $s = t = n/4$ the complexity is $\widetilde{\Theta}(n)$, with the lower bound inherited from the standard disjointness problem DISJ and the upper bound by Alice sending a uniformly random unit

vector $v \in W_A \cap S^{n-1}$ encoded with additive error $1/poly(n)$ for each position. Bob then checks if this vector has projection $\approx 4/n$ or only $1/poly(n)$ onto W_B.

For smaller $s \leq t$ things become interesting. We give a protocol of complexity $O(s\sqrt{t})$ extending the protocol of [24] for the Vector-in-Subspace problem. In that protocol one tries to communicate a vector by using a public coin containing a lot of random unit vectors, and indicating which of them has the largest inner product with the vector one tries to communicate.

The extension is to do this for vectors that have a small overlap with the desired vector only, and to the case of differently sized spaces. This extension is like trying to run the mentioned protocol twice, and a careful analysis is needed using concentration of measure on the sphere and for random projections on subspaces. Basically, Alice has a vector with a given projection onto Bob's space, and tries to communicate the overlap, by pointing out the random vector (in the public coin) that has the best overlap with her vector. For the part of her vector that is in Bob's space to be 'visible' it must be larger than the 'noise', namely the usual deviation of a random vector from its expected projection onto the space. Furthermore it is also important for larger values of s, t to communicate the inner product between her vector and the chosen random vector, because otherwise the noise makes the signal useless. We note that for $s\sqrt{t} \geq n \log n$ our protocol becomes useless.

We also show a lower bound. This builds on the lower bound for the Vector-in-Subspace problem in [16]. The idea is to use a direct sum argument. The conditional external information cost [7] has a direct sum property for the OR of s instances of a problem. It is easy to embed an OR of s instances of $QNDISJ_{1,t/s}$ into one instance of $QNDISJ_{s,t}$. We then extend the result of [16] about $QNDISJ_{1,n/2}$ to conditional external information cost. Originally, this result uses the rectangle/corruption bound. The difficulty is that for the direct sum argument we must lower bound the *conditional* information cost. For this we define a partition (a random subspace V of dimension $n/3$ is drawn, then $W_A \in S^{n-1} \cap V$ and $W_B \subseteq V^{\perp}$ are chosen randomly and independently). We then have to bound the information cost conditioned on V.

Overall we get a lower bond of $\Omega(s(t/s)^{1/3})$. This approach might be improved to $\Omega(\sqrt{st})$ by improving the lower bound for $QNDISJ_{1,t}$.

We note some special cases in the following corollary.

▶ **Corollary 16.**

1. $R(QNDISJ)$ *is between* $\Omega(n)$ *and* $O(n \log n)$.
2. $R(QNDISJ_{\sqrt{n},n/2})$ *is between* $\Omega(n^{2/3})$ *and* $O(n)$.
3. $R(QNDISJ_{\sqrt{n},\sqrt{n}})$ *is between* $\Omega(n^{1/2})$ *and* $O(n^{3/4})$.
4. $R(QNDISJ_{O(1),O(1)})$ *is* $\Theta(1)$.

6.3 Quantum

The upper bound $O(\sqrt{s} \log n)$ is by amplitude amplification (see Fact 9): if Alice sends the uniform superposition over a basis of her space W_A to Bob, who measures with an observable containing W_B as an element, then for 1-inputs the probability of acceptance is $1/s$. Note that for 0-inputs this protocol never accepts.

The lower bound of $\Omega(\sqrt{s})$ is by reduction from the classical disjointness problem DISJ and Razborov's lower bound for the latter [27].

It remains open, whether the log-factor can be shaved off of the upper bound (compare [2]).

The protocol in our upper bound uses many rounds. Round-dependant lower bounds can be derived from the corresponding Disjointness lower bounds, see [9]. In particular, the

one-way quantum complexity of $QNDISJ_{s,t}$ (Alice to Bob) is $\Omega(s)$ (by a reduction from $INDEX_s$, see [17]).

▶ **Theorem 17.** *There is a quantum protocol with k rounds (Alice starting) that computes $QNDISJ_{s,t}$ with communication $O(s/k \cdot \log n)$ as long as $k \leq \sqrt{s}$. The protocol is optimal up to poly-logarithmic factors.*

Proof. The lower bound is by reduction from Disjointness and the main result in [9]. For the upper bounds we use amplitude amplification on the following protocol: Alice sends s/k^2 copies of the state used in our quantum protocol above. Bob measures those copies, and accepts with probability $1/k^2$. ◀

6.4 QCMA

We give a protocol of complexity $O(s^{1/3} \log n)$ in which Merlin can use caps on the sphere are his proofs. The verification is via amplitude amplification.

Merlin and Alice agree beforehand on a net of spherical caps on the sphere in W_A for all subspaces W_A of dimension s. This net has 2^p centers. On a 1-input (W_A, W_B) Merlin sends Alice the closest center to an intersecting unit vector in $W_A \cap W_B$ from the agreed upon net. Alice and Bob then use the same amplitude amplification protocol as in the prover-less case. Since the cap-center is better than a uniform superposition we get a better upper bound. The reason is that the cap center $|c\rangle$ satisfies $\langle c|x\rangle^2 \geq p/s$ for the intersection $|x\rangle$, whereas a uniform superposition $|u\rangle$ over some ortho-normal basis guarantees only $\langle u|x\rangle^2 \geq 1/s$.

▶ **Theorem 18.**
1. *For all $\log s \leq p \leq s$: $QCMA^p(QNDISJ_{s,t}) \leq O(\sqrt{s/p} \log n)$.*
2. *$QCMA(QNDISJ_{s,t}) \leq O(s^{1/3} \log n)$*

We give a conditional lower bound, for protocols with property (*). Such protocols need communication $\Omega(s^{1/3})$. It is enough to show that $QCMA(QNDISJ) = \Omega(n^{1/3})$ by padding.

The idea is that under the condition (*) an (intersectional) proof corresponds to a large subset of the sphere, and we can then, by a new geometrical lemma, find an ortho-normal set of size $\Omega(n/p)$ in any subset of the sphere of measure 2^{-p}. This result can be used to give a reduction from $DISJ_{n/p,n/p}$ to the subfunction of $QNDISJ$ that accepts all 1-inputs in the proof and rejects all 0-inputs.

For this we fix one large, intersectional proof. We then have a quantum protocol that accepts all 1-inputs in the proof, while rejecting all 0-inputs. We find our large ortho-normal set inside the proof, and then embed the classical $DISJ_{n/p,n/p}$ instance. The lower bound follows via the quantum lower bound for $DISJ_{n/p,n/p}$ [27].

This is the geometric lemma mentioned above.

▶ **Lemma 19.** *Let $A \subseteq S^{n-1}$ be a set of measure at least $\mu(A) \geq 2^{-p}$ for $o(\sqrt{n}) \geq p \geq \omega(1)$. Then A contains a set of $\ell = n/(40p)$ vectors $v_1, \ldots, v_\ell$ such the x_i form an ortho-normal system (i.e., every v_i is orthogonal to the span of the other vectors).*

The lower bound statement is as follows.

▶ **Theorem 20.** *Under the condition (*)*
1. *$QCMA^p(QNDISJ_{n/4,n/4}) \geq \Omega(\sqrt{n/p})$ for $p \leq o(\sqrt{n})$.*
2. *$QCMA(QNDISJ_{s,t}) \geq \Omega(s^{1/3})$.*

6.5 MA

We "Merlinize" our randomized protocol (proofs are still spherical caps as in the QCMA case). The result is an upper bound that is the square root of the randomized upper bound.

▶ **Theorem 21.**
1. *For all* $\log s \leq p \leq s$: $MA^p(QNDISJ_{s,t}) \leq O(s\sqrt{t}/p)$.
2. $MA(QNDISJ_{s,t}) \leq O(\sqrt{s\sqrt{t}})$.

Lower bounds for MA-communication complexity can be shown by using the rectangle bound [18]. [16] give such a lower bound, and we get a lower bound that depends polynomially on t. Sadly, no direct sum result is known for the rectangle bound, and our bound is simply a lower bound for $QNDISJ_{1,t}$. We note that the Grassmannian manifold is much harder to handle than the sphere, and so going for an improved rectangle bound heads on seems difficult.

▶ **Fact 22.** $MA(QNDISJ_{1,t}) = \Omega(t^{1/6})$.

It is interesting that this lower bound depends polynomially on the dimension of the *larger* subspace, whereas our QCMA upper bound depends only on the dimension of the smaller subspace.

7 Open Problems

We list a number of interesting open problems.
1. Show an unconditional, large lower bound on $QCMA(QNDISJ)$.
2. Give better bounds for the randomized and MA complexities of $QNDISJ_{s,t}$.
3. Since the randomized and MA protocols we give are one-way protocols, it might be interesting to also get one-way lower bounds.
4. Is $Q(QNDISJ_{s,t}) = O(\sqrt{s})$?
5. Our QMA upper and lower bounds are not close for small dimensional subspaces. For instance we only know that $QMA(QNDISJ_{\log n,\log n})$ is between $\sqrt{\log\log n}$ and $\log n$.
6. Raz and Shpilka [25] show that QMA protocols can be made one-way, but in general only at a polynomial blowup in communication. Can a gap be shown (for instance for Disjointness)?
7. We show that $AQMA(INDEX_n) \geq \Omega(\log n)$. Larger lower bounds for any explicit functions for even AM-complexity are wide open.
8. There is still a gap between the best lower and upper bound known for $QMA(DISJ)$ [20].
9. What is the QMA communication complexity (with entanglement) of a random function?
10. It would be nice if applications of our bounds could be found. Most applications of communication complexity employ Disjointness, so it is quite likely that the 'hidden basis" version of the problem (in particular also the Vector-in-Subspace problem) has interesting applications, e.g. in data-streaming.

References

1 S. Aaronson. Qma/qpoly ⊆ pspace/poly: De-merlinizing quantum protocols. In *Proceedings of 21st IEEE Conference on Computational Complexity*, 2006.
2 S. Aaronson and A. Ambainis. Quantum search of spatial regions. In *Proceedings of 44th IEEE FOCS*, pages 200–209, 2003.

3 S. Aaronson and G. Kuperberg. Quantum versus classical proofs and advice. *Theory of Computing*, 3(1):129–157, 2007.

4 S. Aaronson and A. Wigderson. Algebrization: A New Barrier in Complexity Theory. *ACM Transactions on Computation Theory*, 1(1), 2009.

5 D. Aharonov and T. Naveh. Quantum np - a survey. quant-ph/0210077, 2002.

6 L. Babai, P. Frankl, and J. Simon. Complexity classes in communication complexity theory. In *Proceedings of 27th IEEE FOCS*, pages 337–347, 1986.

7 Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. Information theory methods in communication complexity. In *Proceedings of 17th IEEE Conference on Computational Complexity*, pages 93–102, 2002.

8 G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information: A Millennium Volume*, volume 305 of *AMS Contemporary Mathematics Series*, pages 53–74. AMS, 2002. quant-ph/0005055.

9 M. Braverman, A. Garg, Young K.K., J. Mao, and D. Touchette. Near-optimal bounds on bounded-round quantum communication complexity of disjointness. In *IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 773–791, 2015.

10 A. Chakrabarti, G. Cormode, A. McGregor, J. Thaler, and S. Venkatasubramanian. Verifiable stream computation and arthur-merlin communication. In *30th Conference on Computational Complexity*, pages 217–243, 2015.

11 S. Dasgupta and A. Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

12 M. Göös, T. Pitassi, and T. Watson. Zero-information protocols and unambiguity in arthur-merlin communication. *Algorithmica*, 76(3):684–719, 2016.

13 L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM STOC*, pages 212–219, 1996.

14 B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992. Earlier version in Structures'87.

15 A. Yu. Kitaev. Quantum NP, January 1999. Talk given at AQIP'99, DePaul University, Chicago.

16 B. Klartag and O. Regev. Quantum one-way communication is exponentially stronger than classical communication. In *Proceedings of 43rd ACM STOC*, 2011.

17 H. Klauck. On quantum and probabilistic communication: Las Vegas and one-way protocols. In *Proceedings of 32nd ACM STOC*, pages 644–651, 2000.

18 H. Klauck. Rectangle size bounds and threshold covers in communication complexity. In *18th Annual IEEE Conference on Computational Complexity*, pages 118–134, 2003.

19 H. Klauck. A strong direct product theorem for disjointness. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC*, pages 77–86, 2010.

20 H. Klauck. On arthur merlin games in communication complexity. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity*, pages 189–199, 2011.

21 H. Klauck and S. Podder. Two Results about Quantum Messages. In *Proceedings of MFCS*, 2014.

22 I. Kremer. Quantum communication. Master's thesis, Hebrew University, Computer Science Department, 1995.

23 E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

24 R. Raz. Exponential separation of quantum and classical communication complexity. In *Proceedings of 31st ACM STOC*, pages 358–367, 1999.

25 R. Raz and A. Shpilka. On the power of quantum proofs. In *19th Annual IEEE Conference on Computational Complexity*, pages 260–274, 2004.

26 A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992.

27 A. Razborov. Quantum communication complexity of symmetric predicates. *Izvestiya of the Russian Academy of Sciences, mathematics*, 67(1):159–176, 2003. quant-ph/0204025.

28 A. Sherstov. The pattern matrix method for lower bounds on quantum communication. In *Proceedings of 40th ACM STOC*, pages 85–94, 2008.

29 R. de Wolf. Quantum communication and complexity. *Theoretical Computer Science*, 287(1):337–353, 2002.

30 A. C-C. Yao. Some complexity questions related to distributive computing. In *Proceedings of 11th ACM STOC*, pages 209–213, 1979.

Smoothed and Average-Case Approximation Ratios of Mechanisms: Beyond the Worst-Case Analysis*

Xiaotie Deng[1], Yansong Gao[2], and Jie Zhang[†3]

1 **Shanghai Jiao Tong University, China**
deng-xt@cs.sjtu.edu.cn
2 **Shanghai Jiao Tong University, China**
799@sjtu.edu.cn
3 **University of Southampton, United Kingdom**
jie.zhang@soton.ac.uk

Abstract

The approximation ratio has become one of the dominant measures in mechanism design problems. In light of analysis of algorithms, we define the *smoothed approximation ratio* to compare the performance of the optimal mechanism and a truthful mechanism when the inputs are subject to random perturbations of the worst-case inputs, and define the *average-case approximation ratio* to compare the performance of these two mechanisms when the inputs follow a distribution. For the one-sided matching problem, Filos-Ratsikas et al. [21] show that, amongst all truthful mechanisms, *random priority* achieves the tight approximation ratio bound of $\Theta(\sqrt{n})$. We prove that, despite of this worst-case bound, random priority has a *constant smoothed approximation ratio*. This is, to our limited knowledge, the first work that asymptotically differentiates the smoothed approximation ratio from the worst-case approximation ratio for mechanism design problems. For the average-case, we show that our approximation ratio can be improved to $1 + e$. These results partially explain why random priority has been successfully used in practice, although in the worst case the optimal social welfare is $\Theta(\sqrt{n})$ times of what random priority achieves. These results also pave the way for further studies of smoothed and average-case analysis for approximate mechanism design problems, beyond the worst-case analysis.

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

Keywords and phrases Mechanism Design, Approximation Ratio, Smoothed Analysis, Average-case Analysis

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.16

1 Introduction

Algorithmic mechanism design [35, 36] deals with optimization problems where the input is provided by self-interested agents that participate in the mechanism by reporting their private information. If it best serves their purpose, they might have incentives to report false information. The goal of the designer is to motivate agents to always report truthfully. At the same time, the mechanism designer aims to optimize some objective function over

* Research results reported in this work are partially supported by the National Natural Science Foundation of China (Grant No.61632017).
† Part of this work was done when Jie Zhang was at the University of Oxford and was visiting Shanghai Jiao Tong University. Jie Zhang was supported by the ERC Advanced Grant 321171 (ALGAME).

the agents' reports, subject to a polynomial-time implementability constraint. Examples of applications include scheduling problems [17, 28, 29], facility location problems [22, 39], kidney exchange problems [6], assignment problems [19], one-sided matching [21], resource allocation [24], and auction design [3, 4, 26, 30, 34]. For a more detailed investigation, we refer the reader to [37]. The canonical measure of evaluating how well a truthful mechanism approximately optimizes the objective is the *approximation ratio* [39]. Given any instance, the approximation ratio compares the performance of the optimal mechanism (which always outputs an optimal solution but is not necessarily truthful) against the performance of a truthful mechanism. The worst-case ratio is the largest value of this ratio, amongst all possible inputs.

The difference of mechanism design to algorithm design is the additional constraint of motivating agents to act truthfully in the mechanism. In algorithm design problems, in contrast, the inputs are not controlled by rational agents, and the *worst-case time complexity* is one of the dominant measures to evaluate the performance of algorithms. However, this is a very pessimistic measure. On one hand, if it is possible to obtain a small worst-case complexity, then it is a very strong guarantee on the performance of the algorithm no matter what input is given. On the other hand, there are algorithms that perform well in practice but have a high worst-case complexity bound. To address this disparity, the *average-case* time complexity of an algorithm is an alternative measure to the worst-case complexity; it measures the time complexity of the algorithm, averaged over all possible inputs when they follow a certain distribution. The main motivation of studying average-case complexity is that some algorithms may have to run for a high-order polynomial-time or even exponential time in the worst case, but the input for this to happen may rarely or never occur in practice. So instead of only focusing on unrealistic worst-case instances, researchers consider the performance of the algorithm on average. One criticism to average-case complexity, however, is that it requires assumptions about the distribution of inputs, and these are not guaranteed to hold in practice. Even for the same algorithm, when it is applied to different application areas, the real-world distribution may vary. In light of this, *smoothed complexity* is a hybrid of the worst-case and average-case analysis that inherits advantages of both. Specifically, it measures the expected performance of algorithms under slight random perturbations of the worst-case inputs. If the smoothed complexity of an algorithm is low, then it is unlikely that the algorithm will take a long time to solve practical instances whose data are subject to slight noise and imprecisions. Although average-case and smoothed analysis are usually more complex than the worst-case analysis, they provide different measures from the worst-case complexity.

In this paper, we extend the classical worst-case approximation ratio analysis of mechanisms to the smoothed approximation ratio and average-case approximation ratio analysis. *Average-case approximation ratio*, on average, measures the performance of a truthful mechanism in approximately maximizing social welfare (or minimizing social cost) against the performance of the optimal mechanism; the *smoothed approximation ratio* compares the performance of these two mechanisms when the inputs are subject to random perturbations of worst-case inputs. Therefore, the central questions in smoothed analysis and average-case analysis in mechanism design framework are:

Given a mechanism design problem, in case the worst-case approximation ratio is asymptotically large, are there any mechanisms that achieve a constant smoothed approximation ratio and a constant average-case approximation ratio? Given a concrete mechanism for the problem, does it have a constant smoothed approximation ratio and a constant average-case approximation ratio?

As a first step of extending the worst-case analysis to the smoothed and average-case analysis in mechanism design, we study the problem of approximate social welfare maximization (without money) in the *one-sided matching settings* (also referred to as the *house allocation problem*). These settings consider the fundamental resource allocation problem of assigning items to agents, such that each agent receives exactly one item. It has numerous applications, such as assigning workers to shifts, students to courses, and patients to doctor appointments. In this problem, agents are asked to provide their preferences over items. In game-theoretic terms, these are the agents' von Neumann-Morgenstern utilities [49, 50]. Social welfare is the sum of all agents' utilities. It is easy to see that agents, as self-interested identities, have an incentive to misreport their preferences if they can benefit from this behavior. The *random priority* mechanism, apart from being truthful, also satisfies the desirable properties of anonymity and ex-post Pareto efficiency. In term of social welfare maximization, Filos-Ratsikas et al. [21] show that amongst all truthful mechanisms, random priority achieves the worst-case approximation ratio tight bound of $\Theta(\sqrt{n})$. That is, random priority can guarantee an upper bound of $O(\sqrt{n})$ and there is a worst-case instance on which no mechanism can do better than $\Omega(\sqrt{n})$. Nevertheless, the tight bound instance has a very unique structure such that it is very unlikely to happen in practice. Therefore, we are in great request to understand how well random priority performs on average, when the instances are sampled from a certain distribution. How well it performs under some random perturbations of the worst-case inputs? We address these questions in this paper.

1.1 Our contribution

To the best of our knowledge, this is the first work that asymptotically differentiates the smoothed approximation ratio from the worst-case approximation ratio for mechanism design problems. In particular, we show the following results:

- The random priority mechanism has a constant smoothed approximation ratio.
- The average-case approximation ratio of random priority is upper bounded by a constant $1 + e$, when agents' valuations are drawn from the uniform distribution $\mathrm{U}(0, 1)$.

Our results imply that, although in the worst-case the optimal social welfare is $\Theta(\sqrt{n})$ times of the social welfare attainable by random priority, under polynomial small perturbation around the worst-case inputs and on average, random priority achieves a constant factor of the optimal social welfare.

In [21], the tight bound examples for the worst-case approximation ratio have a unique structure where the preferences of all agents over the items have the same ordering, and the values are all close to either 1 or 0. From the average-case perspective, these examples rarely happen if the valuations are independently and identically drawn from a uniform distribution. From the smoothed analysis perspective, this unanimous ordering would break up after any random perturbation. This is the high-level intuition behind why the smoothed and average-case approximation ratios could be asymptotically different from the worst-case approximation ratio.

1.2 Related work

Spielman and Teng [43, 44] first propose the methodology of smoothed analysis of algorithms with the attempt to explain why the simplex algorithms usually run in polynomial time in practice. They start with the shadow-vertex pivot rule and show its polynomial smoothed complexity. Since then, smoothed analysis has been studied on a variety of different problems

and algorithms, including linear programming [11, 40, 44], online and approximation algorithms [9, 10, 41], searching and sorting [7, 23, 31], game theory [14, 16], local search [5, 20], clustering and knapsack problem [32]. A comprehensive survey can be found in [45].

For average-case analysis of algorithms, different results have been obtained in, for example, quicksort for sorting problem [18] and the simplex algorithm for solving linear programming [13]. We refer the reader to [48] for a comprehensive survey.

In the presence of incentives, the one-sided matching problem was originally defined by Hylland and Zeckhauser [27], and has been studied extensively ever since [12, 19, 33, 46, 51]. We refer the interested reader to surveys on this problem [2, 42]. The *random priority* mechanism, also known as *random serial dictatorship*, has been extensively studied [1, 12, 47]. It has also been widely used in practice, for example, the supplementary round of school student assignment mechanism in New York City, is shown to be equivalent to a random priority mechanism [38].

In the Bayesian auction design literature [15, 25], the focus is on how well a truthful mechanism can approximately maximize the expected revenue, when instances are taken from the entire input space. More specifically, the dominant approach in the study of Bayesian auction design is the *ratio of expectations*. One disadvantage of this approach is that it does not directly compare the performance of the two mechanisms on specific inputs. To address this problem, in what follows, we present a different approach, the *expectation of the ratio*, and compare them in more detail.

2 Preliminaries

We study the one-sided matching problem that consists of n agents and n indivisible items. All the agents are endowed with von Neumann - Morgenstern utilities over the items. The VNM utility theorem states that any rational agent whose preference satisfies four axioms, namely *completeness*, *transitivity*, *continuity*, and *independence*, is endowed with a utility function to represent its preference. These agents report their preferences to a mediator; based on their report, the problem is to allocate items to agents, according to a random permutation such that every agent receives exactly one item.

In this paper, we adopt the *unit-range* canonical representation of agents' valuation [8, 51]. That is, for the utility a_{ij} of agent i receiving item j, we have $\max_j\{a_{ij}\} = 1, \min_j\{a_{ij}\} = 0$, for any $i \in [n]$ [1]. Following this, a valuation profile (or instance) of agents' preferences can be represented by a matrix $\mathrm{A} = [a_{ij}]_{n\times n}$, where row vector $\mathbf{a_i} = (a_{i1}, \ldots, a_{in})$ indicates the valuation of agent i's preference. An *allocation* is an assignment of items to agents. We denote an allocation by a matrix $\mathrm{X} = [x_{ij}]_{n\times n}$, where x_{ij} indicates the probability of agent i receiving item j. Given any preferences of the agents as input, a *mechanism* is a mapping from the input to an allocation X as output.

We denote the set of all possible instances by $\mathcal{A}$ and denote the set of all possible allocation by $\mathcal{X}$. Given a mechanism M and a valuation profile $\mathrm{A} \in \mathcal{A}$, as well as its allocation $\mathrm{X(A)} \in \mathcal{X}$, we denote the utility of agent i by $u_i(\mathrm{X(A)}) = \sum_j a_{ij}x_{ij}$ and denote the social welfare by $SW_{\mathrm{M}}(\mathrm{X(A)}) = \sum_i u_i(\mathrm{X(A)})$. When the context is clear, we drop the allocation notation and simplely refer them by $u_i(\mathrm{A})$ and $SW_{\mathrm{M}}(\mathrm{A})$. We note that there is another interpretation

[1] We note here that our model would be more general and some calculations would be simpler if we drop the constraint $\max_j\{a_{ij}\} = 1$ and $\min_j\{a_{ij}\} = 0$, but only require that $0 \le a_{ij} \le 1$, for all $i, j \in [n]$. The only reason to have such a constraint is to follow the *unit-range* canonical representation of agents' valuation studied in literature.

of our one-sided matching problem: items are divisible and x_{ij} is the fraction of agent i receiving item j. Since the number of agents is equal to the number of items, and every agent receives exactly one item, the allocation matrix X is a doubly stochastic matrix, i.e., $\sum_j x_{ij} = 1$ for any i, and $\sum_i x_{ij} = 1$ for any j. According to the Birkhoff - von Neumann theorem, every doubly stochastic matrix can be decomposed into a convex combination of some permutation matrices. Therefore, $u_i(\mathrm{A})$ and $SW_{\mathrm{M}}(\mathrm{A})$ can be interpreted as expected utilities and expected social welfare in the indivisible items setting, and can be interpreted as exact utilities and exact social welfare in the divisible items setting.

Agents are self-interested and look to maximize their utilities by giving a mendacious preference to the mechanism as part of the input. In approximate mechanism design, we restrict our interest to the class of *truthful mechanisms*, i.e., the mechanisms in which agents cannot improve their utilities by misreporting. The canonical measure of efficiency of a truthful mechanism M is the *worst-case approximation ratio*,

$$r_{\mathrm{worst}}(\mathrm{M}) = \sup_{\mathrm{A}\in\mathcal{A}} \frac{SW_{\mathrm{OPT}}(\mathrm{A})}{SW_{\mathrm{M}}(\mathrm{A})},$$

where $SW_{\mathrm{OPT}}(\mathrm{A}) = \max_{\mathrm{X}\in\mathcal{X}} \sum_{i=1}^{n} u_i(\mathrm{X})$ is the optimal social welfare which is equal to the value of the maximum weight matching between agents and items. This ratio compares social welfare of the optimal allocation against social welfare of a truthful mechanism M. Note that the ratio is no less than 1.

Random priority mechanism fixes an ordering of the agents uniformly at random and then lets them pick their most preferred item from the set of available items based on this ordering. It is shown in [21] that random priority achieves the matching approximation ratio bound of $\Theta(\sqrt{n})$. This result implies that random priority is asymptotically the best truthful mechanism.

2.1 Smoothed and average-case approximation ratios

Analogously to the definition of smoothed complexity of algorithms, we define the *smoothed approximation ratio* as follows:

$$r_{\mathrm{smoothed}}(\mathrm{M}) = \max_{\mathrm{A}} \mathop{\mathbb{E}}_{g_{ij}\sim\mathrm{N}(0,1)} \left[\frac{SW_{\mathrm{OPT}}(\mathrm{A}+\sigma||\mathrm{A}||\mathrm{G})}{SW_{\mathrm{M}}(\mathrm{A}+\sigma||\mathrm{A}||\mathrm{G})} \right],$$

where the parameter σ is the size of the perturbation and $\mathrm{G} = [g_{ij}]_{n\times n}$. That is to say, $\sigma||\mathrm{A}||\mathrm{G}$ is a matrix of independent Gaussian variables of mean 0 and standard deviation $\sigma||\mathrm{A}||$. We multiply by $||\mathrm{A}||$ to relate the magnitude of the perturbation to the magnitude of the input A. We say that a mechanism has a *constant smoothed approximation ratio* if its smoothed approximation ratio is polynomial only in $1/\sigma$, when the size of the input n approaches infinity [2]. Following the natural of the unit-range representation, we consider the *property-preserving perturbation* by restricting a natural perturbation model to preserve

[2] We will show that the smoothed approximation ratio of random priority is polynomial only in $1/\sigma$ but not in n. This is in contrast to its worst-case approximation ratio $\Theta(\sqrt{n})$. For this reason our result asymptotically differentiates the smoothed approximation ratio from the worst-case ratio. It is a scenery analogously to the analysis of algorithms, where in some problems the smoothed complexity is polynomial in n and $1/\sigma$ but the worst-case complexity is exponential in n. In both studies, the point of smoothed analysis is to show that although the mechanism (algorithm) may perform poorly in the worst case, it performs well under slight random perturbations of worst-case inputs.

certain properties of the input. Specifically, the perturbation preserves agents' 0 and 1 valuation and all valuations stay in the interval $[0, 1]$ [3].

Similarly, we define the *average-case approximation ratio* of mechanisms as follows:

$$r_{\text{average}}(\text{M}) = \mathop{\mathbb{E}}_{a_{ij}\sim \text{U}} \left[\frac{SW_{\text{OPT}}(\text{A})}{SW_{\text{M}}(\text{A})} \right],$$

where the elements a_{ij} of input A is chosen from a distribution U. In this paper, we consider the case that agent's values a_{ij} are independent variables and follow the uniform distribution $\text{U}(0, 1)$. For any agent i, since $a_{ij}, j = 1, \ldots, n$, are independent and identically distributed $\text{U}(0, 1)$ random variables, the sum $\sum_{j=1}^{n} a_{ij}$ follows the *Irwin-Hall distribution*. So,

$$\Pr\left[\sum_{j=1}^{n} a_{ij} \le x\right] = \frac{1}{n!} \sum_{k=0}^{\lfloor x-1 \rfloor} (-1)^k \binom{n}{k} (x-1-k)^n,$$

where $\lfloor \cdot \rfloor$ is the floor function.

In the following, we contrast this approach to the established literature on Bayesian mechanism design.

2.2 Bayesian mechanism design approach

In Bayesian mechanism design [15, 25], there is also a prior distribution from which the agent types come from, but the objective is to characterize the maximum ratio (for some given distribution of the agent types) of the expected social welfare of a truthful mechanism over the expected social welfare of the optimal mechanism, i.e., the ratio of expectations. That is, the objective is to characterize the ratio r in the following formula,

$$\mathbb{E}\left[SW_{\text{OPT}}(\text{A})\right] \le r \cdot \mathbb{E}\left[SW_{\text{M}}(\text{A})\right].$$

Thus, the key difference to our approach is that this measurement does not directly compare the performance of the two mechanisms on specific inputs. We discuss this in more detail in the following.

2.3 Comparison of the two approaches

We note that the worst-case approximation ratio compares the performance of the two mechanisms on a *case-by-case basis*. In addition, the smoothed analysis of algorithms is defined as the performance of the algorithm on the worst-case input when it is subject to a slight random perturbation. Therefore, it measures the averaged performance of the algorithm in a small neighbourhood area of *an individual input*. In essence, both metrics consider a specific input (or some small noise around a specific input). Indeed, our approach is informed by these two metrics.

Average-case approximation ratio and the Bayesian approach each has their strength in measuring how good a truthful mechanism approximates the optimal mechanism. The random priority mechanism for the one-sided matching problem is used for example, in the supplementary round of school student assignment in the New York City once in a while. In

[3] We note that the choice of Gaussian perturbation is standard in classical smoothed analysis of algorithms, and restriction of property-preserving perturbations is necessary and meaningful even in average-case analysis of algorithms.

this case, we are interested in how likely a truthful mechanism performs well on a particular instance. Therefore, the average-case approximation ratio, which is the expectation of ratio, fits in this need. On the other hand, online auction mechanisms, as the revenue source for Internet businesses, are used one a daily base. In that case, we are interested in how well a truthful mechanism performs comparing to the optimal mechanism when instances are sampled throughout the entire input space. Therefore, the Bayesian approach, which is the ratio of expectations, is more suitable.

We note that our definitions, the expectation of ratio, introduce more technical challenges. As we are interested in characterizing the expectation of a non-linear function (ratio of two variables), we can no longer handle expectations of two different variables separately. Let us take for example when agents' valuations are drawn independently and identically from the uniform distribution $\mathrm{U}(0,1)$. In the Bayesian approach, it is trivial to see that the expected social welfare of the completely random mechanism (by ignoring agents' valuations and allocate items totally random) is $n/2$, and the expected social welfare of the optimal mechanism is less than n. So the ratio of two expectations is less than 2. However, for the case-by-case comparison in the average-case approximation ratio, it is not directly clear what the ratio would be. In this paper, as a first step, we study the uniform distribution for the average-case ratio and the smoothed ratio. We suspect that different distributions would result in different ratios. In contrast, by the linearity of the ratio in two expectations, in the well-studied Bayesian auction design literature, it is common that positive results hold for a class of distributions, such as monotone hazard distributions [25].

3 Smoothed Analysis

In this section we show that random priority has a constant smoothed approximation ratio.

Firstly, let us understand the structure of the profile space $\mathcal{A}$ in the unit-range setting. For each agent i with valuations $\mathbf{a}_i = (a_{i1}, \ldots, a_{in})$, it has utility 0 on its least preferred item and has utility 1 on its most preferred item. So there remains $n-2$ elements which are random variables following the uniform distribution $\mathrm{U}(0,1)$. Note that there are $n(n-1)$ possible choices to select $n-2$ out of n elements to be random variables, while the other two elements being either 0 or 1. So there are $(n(n-1))^n$ possible configuration of such random variables. Therefore, we can decompose the sample space $\mathcal{A}$ into sets $\mathrm{S_k}, k = 1, \ldots, (n(n-1))^n$, where each set $\mathrm{S_k}$ contains the instances A that the specific two out of n values of $a_{ij}, j \in [n]$ are fixed for each agent i.

Given any valuation profile A, its 0 and 1 entries are fixed so it must belong to exactly one of the set S_k. Let $\mathrm{S(A)} \in \{\mathrm{S}_k\}_{k\in[(n(n-1))^n]}$ denote the set where A belongs to. Let $\mathrm{H(A)} = \{(i,j) : a_{ij} = 0 \text{ or } a_{ij} = 1\}$, clearly $\mathrm{H(A)}$ has $2n$ elements. Given any set $\mathrm{S_k}$ and any two instances A_1 and A_2 chosen from $\mathrm{S_k}$, it is easy to see that $\mathrm{H(A}_1) = \mathrm{H(A}_2)$.

For convenience, denote $\Gamma = (\gamma_{ij})_{n\times n} = \sigma||\mathrm{A}||\mathrm{G}$. The probability density function of Γ is

$$f(\Gamma) = K \cdot e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}} = K \cdot e^{-\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\gamma_{ij}^2}{2\sigma^2||\mathrm{A}||^2}},$$

where K is a constant, $|\Gamma|^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}\gamma_{ij}^2$. Therefore, our problem of characterizing the smoothed approximation ratio of random priority is reduced to computing the upper bound of

$$\max_{\mathrm{A}\in\mathcal{A}} \int_{\mathrm{A}+\Gamma\in\mathrm{S(A)}} \frac{1}{K'} \cdot e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}} \cdot \frac{SW_{OPT}(\mathrm{A}+\Gamma)}{SW_{RP}(\mathrm{A}+\Gamma)} d\Gamma,$$

where $K' = \int_{\mathrm{A}+\Gamma\in\mathrm{S(A)}} e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}} d\Gamma$. As we clarified in the Preliminaries, we consider the property-preserving perturbation that preserves agents' 0 and 1 valuation and all valuations stay in the interval $[0,1]$. Therefore, our integral is taken over the space $\mathrm{A}+\Gamma \in \mathrm{S(A)}$. We would need the following auxiliary lemma. The proof is omitted due to the page limit.

▶ **Lemma 1.** *For any valuation profile* $\mathrm{A} \in \mathcal{A}$ *and standard vector norm such as Euclidean norm, p-norm and maximum norm, we have* $||\mathrm{A}|| \geq 1$.

Our main result of this section is the following.

▶ **Theorem 2.** *Random priority has a constant smoothed approximation ratio. That is to say, its smoothed approximation ratio is polynomial in* $1/\sigma$*, when the input size* n *approaches infinity.*

Obviously, when $\frac{1}{\sigma} \geq \sqrt{n}$, since the smoothed approximation ratio is upper bounded by worst-case approximation ratio $O(\sqrt{n})$, there $\exists M > 0$, such that

$$r_{\text{smoothed}} \leq r_{\text{worst}} \leq O(\sqrt{n}) \leq M \cdot \frac{1}{\sigma}$$

In the following we focus on the case when $\frac{1}{\sigma} < \sqrt{n}$. We further divide our analysis into two cases, depending on the size of the perturbation σ and the magnitude of A.

Case 1: $\sigma||\mathrm{A}|| \leq 1$.

Firstly, the following lemma lower bounds the constant K'.

▶ **Lemma 3.** *When* $\sigma||\mathrm{A}|| \leq 1$*, for any instance* A *and Gaussian perturbation* Γ*, we have*

$$K' = \int_{\mathrm{A}+\Gamma\in\mathrm{S(A)}} e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}} d\Gamma \geq \left(e^{-\frac{1}{2}} \cdot \sigma||\mathrm{A}||\right)^{n(n-2)}$$

Proof.

$$\begin{aligned}
K' &= \int_{\mathrm{A}+\Gamma\in\mathrm{S(A)}} e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}} d\Gamma = \prod_{(i,j)\in\mathrm{H(A)}} e^{-\frac{0}{2\sigma^2||\mathrm{A}||^2}} \prod_{(i,j)\notin\mathrm{H(A)}} \int_{a_{ij}+\gamma_{ij}\in[0,1]} e^{-\frac{|\gamma_{ij}|^2}{2\sigma^2||\mathrm{A}||^2}} d\gamma_{ij} \\
&= 1 \cdot \prod_{(i,j)\notin\mathrm{H(A)}} \int_{-a_{ij}}^{1-a_{ij}} e^{-\frac{|\gamma_{ij}|^2}{2\sigma^2||\mathrm{A}||^2}} d\gamma_{ij} \\
&= \prod_{(i,j)\notin\mathrm{H(A)}} \left(\int_{0}^{1-a_{ij}} e^{-\frac{|\gamma_{ij}|^2}{2\sigma^2||\mathrm{A}||^2}} d\gamma_{ij} + \int_{0}^{a_{ij}} e^{-\frac{|\gamma_{ij}|^2}{2\sigma^2||\mathrm{A}||^2}} d\gamma_{ij} \right) \\
&\geq \prod_{(i,j)\notin\mathrm{H(A)}} \left(\int_{0}^{1-a_{ij}} e^{-\frac{|\gamma_{ij}|^2}{2\sigma^2||\mathrm{A}||^2}} d\gamma_{ij} + \int_{1-a_{ij}}^{1} e^{-\frac{|\gamma_{ij}|^2}{2\sigma^2||\mathrm{A}||^2}} d\gamma_{ij} \right) = \prod_{(i,j)\notin\mathrm{H(A)}} \int_{0}^{1} e^{-\frac{|\gamma_{ij}|^2}{2\sigma^2||\mathrm{A}||^2}} d\gamma_{ij} \\
&= \left(\int_{0}^{1} e^{-\frac{|\gamma_{ij}|^2}{2\sigma^2||\mathrm{A}||^2}} d\gamma_{ij} \right)^{n(n-2)} \geq \left(\int_{0}^{\sigma||\mathrm{A}||} e^{-\frac{|\gamma_{ij}|^2}{2\sigma^2||\mathrm{A}||^2}} d\gamma_{ij} \right)^{n(n-2)} \\
&\geq \left(\int_{0}^{\sigma||\mathrm{A}||} e^{-\frac{1}{2}} d\gamma_{ij} \right)^{n(n-2)} = \left(e^{-\frac{1}{2}} \cdot \sigma||\mathrm{A}||\right)^{n(n-2)}
\end{aligned}$$

◀

Secondly, given any instance A and its associated set S(A), we further partition the set S(A) into two subsets, according to the value of social welfare of random priority on its elements, namely $\mathrm{S}_c(\mathrm{A}) = \{\mathrm{B} \in \mathrm{S(A)} : SW_{RP(\mathrm{B})} \leq n^c\}$ and the residual $\mathrm{S(A)}/\mathrm{S}_c(\mathrm{A}) = \{\mathrm{B} \in \mathrm{S(A)} :$

$SW_{RP(\mathrm{B})} > n^c\}$, where $0 < c < 1$ is a parameter that will facilitate us to prove our main theorem. Let $\mathrm{V}(\mathrm{S_c(A)})$ and $\mathrm{V}(\mathrm{S(A)})$ be the volume of $\mathrm{S}_c(\mathrm{A})$ and $\mathrm{S(A)}$, respectively. We upper bound the fraction of the elements of the set $\mathrm{S(A)}$ for which the social welfare of random priority on its elements is no more than n^c. A useful observation here is that, this objective is equivalent to computing the probability that the social welfare of random priority on any instance A is no more than n^c, when the agents' values a_{ij} are independently and identically drawn from the uniform distribution $\mathrm{U}(0,1)$.

▶ **Lemma 4.** *Given any n, for every $0 < c < 1$, we have*

$$\frac{\mathrm{V}(\mathrm{S_c(A)})}{\mathrm{V}(\mathrm{S(A)})} \le \frac{\mathrm{e}^{2n}}{\sqrt{2\pi n}} \cdot \left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n(n-2)}.$$

Proof. We note that random priority is a truthful mechanism, and it fixes an ordering of agents uniformly at random. Every agent i has a probability of $1/n$ to be selected first to choose an item, a probability of $2/n$ to be selected first or second to choose an item, and so on. That is to say, for each agent i, if we sort the items in decreasing order according to agent i's preference, then the allocation vector $\mathbf{x}_i = (x_{ij})$ obtained by random priority would stochastically dominates the vector $(\frac{1}{n}, \ldots, \frac{1}{n})$. I.e., for every $k = 1, \ldots, n$, it holds that $\sum_{j=1}^{k} x_{ij} \ge k/n$.

Therefore, the utility of agent i in any instance A is $u_i(\mathrm{A}) = \sum_{j=1}^{n} a_{ij}x_{ij} \ge \frac{1}{n} \cdot \sum_{j=1}^{n} a_{ij}$. Hence, we have social welfare $\mathrm{SW_{RP}(A)} = \sum_i u_i(\mathrm{A}) \ge \frac{1}{n}\sum_{i,j} a_{ij}$. So, $\mathrm{SW_{RP}(A)} \le n^c$ implies $\sum_{i\in[n],j\in[n]} a_{ij} \le n^{1+c}$. Together with the observation noted above, we utilize the Irwin-Hall distribution to prove the lemma.

$$\begin{aligned}
&\frac{\mathrm{V}(\mathrm{S_c(A)})}{\mathrm{V}(\mathrm{S(A)})} = \Pr\left[\mathrm{SW_{RP}(A)} \le n^c\right] \le \Pr\left[\sum\nolimits_{i\in[n],j\in[n]} a_{ij} \le n^{1+c}\right] \\
&= \Pr\left[\sum\nolimits_{(i,j)\notin \mathrm{H(A)}} a_{ij} \le n^{1+c} - n\right] \\
&= \frac{1}{(n(n-2))!} \sum_{k=0}^{\lfloor n^{1+c}-n\rfloor} (-1)^k \binom{n(n-2)}{k} (n^{1+c} - n - k)^{n(n-2)} \\
&\le \frac{1}{(n(n-2))!} \sum_{k=0}^{n(n-2)} \binom{n(n-2)}{k} n^{(1+c)(n(n-2))} = \frac{1}{(n(n-2))!} \cdot (2n^{1+c})^{n(n-2)} \\
&\le \frac{(2n^{1+c})^{n(n-2)}}{\sqrt{2\pi(n(n-2))} \cdot \left(\frac{n(n-2)}{\mathrm{e}}\right)^{n(n-2)}} = \frac{(2n^{1+c})^{n(n-2)}}{\sqrt{2\pi n} \cdot \left(\frac{n^2}{\mathrm{e}}\right)^{n(n-2)}} \cdot \frac{\sqrt{2\pi n} \cdot \left(\frac{n^2}{\mathrm{e}}\right)^{n(n-2)}}{\sqrt{2\pi(n(n-2))} \cdot \left(\frac{n(n-2)}{\mathrm{e}}\right)^{n(n-2)}} \\
&= \frac{1}{\sqrt{2\pi n}} \cdot \left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n(n-2)} \cdot \left(1 + \frac{2}{n-2}\right)^{n(n-2)+\frac{1}{2}} \\
&\le \frac{1}{\sqrt{2\pi n}} \cdot \left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n(n-2)} \cdot \mathrm{e}^{2n} = \frac{\mathrm{e}^{2n}}{\sqrt{2\pi n}} \cdot \left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n(n-2)}
\end{aligned}$$

◀

Combing these two lemmas we can prove our main theorem under this case.

Proof of Case 1 of Theorem 2. Note that when $\mathrm{A} + \Gamma \notin \mathrm{S}_c(\mathrm{A})$, we can upper bound $\frac{SW_{OPT(\mathrm{A}+\Gamma)}}{SW_{RP(\mathrm{A}+\Gamma)}} \le n^{1-c}$, and in all cases $\frac{SW_{OPT(\mathrm{A}+\Gamma)}}{SW_{RP(\mathrm{A}+\Gamma)}} \le O(\sqrt{n})$ according to the worst-case approximation ratio result [21]. In addition, by Lemma 4 we have,

$$\mathrm{V}(\mathrm{S_c(A)}) \le \frac{\mathrm{e}^{2n}}{\sqrt{2\pi n}} \cdot \left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n(n-2)} \cdot \mathrm{V}(\mathrm{S(A)}) = \frac{\mathrm{e}^{2n}}{\sqrt{2\pi n}} \cdot \left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n(n-2)} \cdot 1$$

Therefore,

$$\mathop{\mathbb{E}}_{g_{ij}\sim \mathrm{N}(0,1)}\left[\frac{SW_{\mathrm{OPT}}(\mathrm{A}+\sigma||\mathrm{A}||\mathrm{G})}{SW_{\mathrm{RP}}(\mathrm{A}+\sigma||\mathrm{A}||\mathrm{G})}\right] = \int_{\mathrm{A}+\Gamma\in\mathrm{S}(\mathrm{A})} \frac{e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}}}{\int_{\mathrm{A}+\Gamma\in\mathrm{S}(\mathrm{A})} e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}}d\Gamma}\cdot\frac{SW_{OPT(\mathrm{A}+\Gamma)}}{SW_{RP(\mathrm{A}+\Gamma)}}d\Gamma$$

$$= \int_{\mathrm{A}+\Gamma\notin\mathrm{S}_c(\mathrm{A})} \frac{e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}}\cdot\frac{SW_{OPT(\mathrm{A}+\Gamma)}}{SW_{RP(\mathrm{A}+\Gamma)}}}{\int_{\mathrm{A}+\Gamma\in\mathrm{S}(\mathrm{A})} e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}}d\Gamma}d\Gamma + \int_{\mathrm{A}+\Gamma\in\mathrm{S}_c(\mathrm{A})} \frac{e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}}\cdot\frac{SW_{OPT(\mathrm{A}+\Gamma)}}{SW_{RP(\mathrm{A}+\Gamma)}}}{\int_{\mathrm{A}+\Gamma\in\mathrm{S}(\mathrm{A})} e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}}d\Gamma}d\Gamma$$

$$\le \int_{\mathrm{A}+\Gamma\notin\mathrm{S}_c(\mathrm{A})} \frac{e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}}\cdot n^{1-c}}{\int_{A+\Gamma\in\mathrm{S}(\mathrm{A})} e^{-\frac{|\Gamma|^2}{2\sigma^2||A||^2}}d\Gamma}d\Gamma + \int_{\mathrm{A}+\Gamma\in\mathrm{S}_c(\mathrm{A})} \frac{e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}}\cdot O(\sqrt{n})}{\int_{\mathrm{A}+\Gamma\in\mathrm{S}(\mathrm{A})} e^{-\frac{|\Gamma|^2}{2\sigma^2||\mathrm{A}||^2}}d\Gamma}d\Gamma$$

$$\le n^{1-c} + \frac{\mathrm{V}(\mathrm{S_c}(\mathrm{A}))\cdot 1\cdot O(\sqrt{n})}{\int_{\mathrm{A}+\Gamma\in\mathrm{S}(\mathrm{A})} e^{-\frac{|\Gamma|^2}{2\sigma^2||A||^2}}d\Gamma} \le n^{1-c} + \frac{\frac{e^{2n}}{\sqrt{2\pi n}}\cdot\left(\frac{2e}{n^{1-c}}\right)^{(n(n-2))}\cdot O(\sqrt{n})}{\left(e^{-\frac{1}{2}}\cdot\sigma||\mathrm{A}||\right)^{n(n-2)}} \quad \text{(by Lemma 3)}$$

$$= n^{1-c} + \frac{e^{2n}}{\sqrt{2\pi n}}\cdot O(\sqrt{n})\cdot\left(\frac{2e^{\frac{3}{2}}}{n^{1-c}\sigma||\mathrm{A}||}\right)^{(n(n-2))}$$

Now let $1-c = \frac{2}{(n-2)\log n} + \frac{\log(2e^{\frac{3}{2}})}{\log n} + \frac{\log(1/\sigma||\mathrm{A}||)}{\log n}$. According to Lemma 1, we have $||\mathrm{A}|| \ge 1$, combining with the assumption $\frac{1}{\sigma} < \sqrt{n}$, we get that $\frac{\log(1/\sigma||\mathrm{A}||)}{\log n} \le \frac{\log(1/\sigma)}{\log n} < \frac{\log(\sqrt{n})}{\log n} = \frac{1}{2}$, which means $0 < 1-c < \frac{1}{2}$, as n approaches infinity. Therefore the value of $1-c$ is feasible. Then,

$$\mathop{\mathbb{E}}_{g_{ij}\sim \mathrm{N}(0,1)}\left[\frac{SW_{\mathrm{OPT}}(\mathrm{A}+\sigma||\mathrm{A}||\mathrm{G})}{SW_{\mathrm{RP}}(\mathrm{A}+\sigma||\mathrm{A}||\mathrm{G})}\right] \le 2e^{\frac{3}{2}}\cdot e^{\frac{2}{n-2}}\cdot\frac{1}{\sigma||\mathrm{A}||} + \frac{O(\sqrt{n})}{\sqrt{2\pi n}} \le 2e^{\frac{3}{2}}\cdot e^{\frac{2}{n-2}}\cdot\frac{1}{\sigma} + \frac{O(\sqrt{n})}{\sqrt{2\pi n}}$$

So, when $\sigma||\mathrm{A}|| \le 1$ and $\frac{1}{\sigma} < \sqrt{n}$, we have $\mathbb{E}_{g_{ij}\sim \mathrm{N}(0,1)}\left[\frac{SW_{\mathrm{OPT}}(\mathrm{A}+\sigma||\mathrm{A}||\mathrm{G})}{SW_{\mathrm{RP}}(\mathrm{A}+\sigma||\mathrm{A}||\mathrm{G})}\right] < 2e^{\frac{3}{2}}\cdot\frac{1}{\sigma}$ as n approaches infinity.

Case 2 $\sigma||\mathrm{A}|| > 1$. The proof is omitted due to the page limit.

Combining these two cases we complete our proof of Theorem 2. ◀

4 Average-case Analysis

In this section we show the average-case approximation ratio of the random priority mechanism can be improved to $1+e$.

Firstly, we upper bound the probability of the social welfare of random priority when it is smaller than a value n^c, for a certain parameter c.

▶ **Lemma 5.** *Given any n, for every $0 < c < 1$, such that $\frac{\mathrm{e}^2}{\sqrt{2\pi n}}\cdot\left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n-2} < 1$, we have*

$$\Pr[SW_{RP}(\mathrm{A}) \le n^c] \le \frac{\mathrm{e}^2}{\sqrt{2\pi}}\cdot\sqrt{n}\cdot\left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n-2}.$$

Proof. For any agent i, since $a_{ij}, j = 1,\ldots,n$, are independent and identically distributed $\mathrm{U}(0,1)$ random variables, the sum $\sum_{j=1}^{n} a_{ij}$ follows the *Irwin-Hall distribution*. In addition, in the unit-range setting, there exist j_1 and j_2 such that $a_{ij_1} = 1, a_{ij_2} = 0$, so $\sum_{j=1}^{n} a_{ij} \le x$ is equivalent to $\sum_{j\ne j_1,j_2} a_{ij} \le x-1$, given any $x > 0$. Therefore,

$$\Pr\left[\sum\nolimits_{j=1}^{n} a_{ij} \le x\right] = \Pr\left[\sum\nolimits_{j\ne j_1,j_2} a_{ij} \le x-1\right] = \frac{1}{(n-2)!}\sum_{k=0}^{\lfloor x-1\rfloor}(-1)^k\binom{n-2}{k}(x-1-k)^{n-2}$$

where $\lfloor\cdot\rfloor$ is the floor function. So,

$$\begin{aligned}
&\Pr\left[\sum_{j=1}^{n} a_{ij} \leqslant n^c\right] = \frac{1}{(n-2)!}\sum_{k=0}^{\lfloor n^c-1\rfloor}(-1)^k\binom{n-2}{k}(n^c-1-k)^{n-2}\\
&\leq \frac{1}{(n-2)!}\sum_{k=0}^{\lfloor n^c-1\rfloor}\binom{n-2}{k}(n^c)^{n-2} \leq \frac{1}{(n-2)!}\sum_{k=0}^{n-2}\binom{n-2}{k}n^{c(n-2)}\\
&= \frac{1}{(n-2)!}\cdot(2n^c)^{n-2} \leq \frac{(2n^c)^{n-2}}{\sqrt{2\pi(n-2)}\cdot\left(\frac{n-2}{\mathrm{e}}\right)^{n-2}} \qquad (1)\\
&= \frac{(2n^c)^{n-2}}{\sqrt{2\pi n}\cdot\left(\frac{n}{\mathrm{e}}\right)^{n-2}}\cdot\frac{\sqrt{2\pi n}\cdot\left(\frac{n}{\mathrm{e}}\right)^{n-2}}{\sqrt{2\pi(n-2)}\cdot\left(\frac{n-2}{\mathrm{e}}\right)^{n-2}}\\
&= \frac{1}{\sqrt{2\pi n}}\cdot\left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n-2}\cdot\left(1+\frac{2}{n-2}\right)^{n-2+\frac{1}{2}}\\
&\leq \frac{1}{\sqrt{2\pi n}}\cdot\left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n-2}\cdot\mathrm{e}^2 = \frac{\mathrm{e}^2}{\sqrt{2\pi n}}\cdot\left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n-2}
\end{aligned}$$

where inequality (1) is due to Stirling's formula.

As we have shown in Lemma 4, in random priority mechanism, agent i's utility $u_i(\mathrm{A}) = \sum_{j=1}^{n} a_{ij}x_{ij} \geq \frac{1}{n}\cdot\sum_{j=1}^{n} a_{ij}$. It implies that $\Pr\left[u_i(\mathrm{A}) > x\right] \geq \Pr\left[\frac{1}{n}\sum_{j=1}^{n} a_{ij} > x\right]$. Therefore,

$$\begin{aligned}
&\Pr\left[SW_{RP}(\mathrm{A}) > n^c\right] = \Pr\left[\sum_{i=1}^{n} u_i(\mathrm{A}) > n^c\right] \geq \prod_{i=1}^{n}\Pr\left[u_i(\mathrm{A}) > \frac{1}{n}\cdot n^c\right]\\
&\geq \prod_{i=1}^{n}\Pr\left[\frac{1}{n}\sum_{j=1}^{n} a_{ij} > \frac{1}{n}\cdot n^c\right] = \prod_{i=1}^{n}\Pr\left[\sum_{j=1}^{n} a_{ij} > n^c\right] = \prod_{i=1}^{n}\left(1-\Pr\left[\sum_{j=1}^{n} a_{ij} \leqslant n^c\right]\right)\\
&\geq \left(1-\frac{\mathrm{e}^2}{\sqrt{2\pi n}}\cdot\left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n-2}\right)^n \geq 1-n\cdot\frac{\mathrm{e}^2}{\sqrt{2\pi n}}\cdot\left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n-2} \qquad (2)\\
&= 1-\frac{\mathrm{e}^2}{\sqrt{2\pi}}\cdot\sqrt{n}\cdot\left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n-2}
\end{aligned}$$

where inequality (2) is due to Bernoulli's inequality and the condition that $\frac{\mathrm{e}^2}{\sqrt{2\pi n}}\cdot\left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n-2} < 1$. Hence,

$$\Pr[SW_{RP}(\mathrm{A}) \leq n^c] \leq \frac{\mathrm{e}^2}{\sqrt{2\pi}}\cdot\sqrt{n}\cdot\left(\frac{2\mathrm{e}}{n^{1-c}}\right)^{n-2}. \qquad \blacktriangleleft$$

Secondly, we lower bound the probability of the social welfare of random priority when it is lager than $\frac{n}{2}$.

▶ **Lemma 6.** *Given any n, we have* $\Pr[SW_{RP}(\mathrm{A}) \geq \frac{n}{2}] \geq \frac{1}{2}$.

Proof. Firstly, as we show in Lemma 4, the utility of agent i in any instance A is $u_i(\mathrm{A}) = \sum_{j=1}^{n} a_{ij}x_{ij} \geq \frac{1}{n}\cdot\sum_{j=1}^{n} a_{ij}$, and the social welfate $\mathrm{SW}_{\mathrm{RP(A)}} = \sum_i u_i(\mathrm{A}) \geq \frac{1}{n}\sum_{i,j} a_{ij}$. Therefore, event $\left\{\sum_{i,j} a_{ij} \geq \frac{n^2}{2}\right\}$ implies $\left\{\mathrm{SW}_{\mathrm{RP(A)}} \geq \frac{n}{2}\right\}$. So, we have $\Pr\left[\sum_{i,j} a_{ij} \geq \frac{n^2}{2}\right] \leq \Pr\left[\mathrm{SW}_{\mathrm{RP}}(\mathrm{A}) \geq \frac{n}{2}\right]$.

Secondly, let E be the $n \times n$ all-ones matrix. Denote matrix $\mathrm{A}' = (a'_{ij})_{n\times n} = \mathrm{E} - \mathrm{A}$. Obviously $\sum_{i,j} a_{ij} + \sum_{i,j} a'_{ij} = n^2$. Since all a_{ij}'s follow the uniform distribution,

$$\Pr\left[\sum_{i,j} a_{ij} \geq \frac{n^2}{2}\right] = \Pr\left[\sum_{i,j} a'_{ij} \geq \frac{n^2}{2}\right] = \Pr\left[n^2 - \sum_{i,j} a_{ij} \geq \frac{n^2}{2}\right] = \Pr\left[\sum_{i,j} a_{ij} \leq \frac{n^2}{2}\right].$$

So $\Pr\left[\sum_{i,j} a_{ij} \geq \frac{n^2}{2}\right] = \frac{1}{2}$. Hence, $\Pr\left[\mathrm{SW}_{\mathrm{RP}}(\mathrm{A}) \geq \frac{n}{2}\right] \geq \Pr\left[\sum_{i,j} a_{ij} \geq \frac{n^2}{2}\right] = \frac{1}{2}$. ◀

Next we will use Lemma 5 and Lemma 6 to prove our main result in this section. Essentially, Lemma 5 and Lemma 6 bound the probability of the social welfare of random priority on valuation profile A. By carefully choosing parameter c, we can divide the valuation space into three sets: $\{SW_{RP(\mathrm{A})} \leq n^c\}$, $\{n^c < SW_{RP(\mathrm{A})} < \frac{n}{2}\}$ and $\{SW_{RP(\mathrm{A})} \geq \frac{n}{2}\}$. We bound the probabilities of instance A falling into each set and the ratios of optimal social welfare against the social welfare of random priority mechanism. Note that in any cases, the worst-case ratio is upper bounded by $O(\sqrt{n})$. By adding them up together, we obtain our upper bound of $1 + e$.

▶ **Theorem 7.** *The average case approximation ratio is upper bounded by* $1 + e$. *That is,*

$$r_{average} = \mathop{\mathbb{E}}_{a_{ij}\sim \mathrm{U}}\left[\frac{SW_{OPT(\mathrm{A})}}{SW_{RP(\mathrm{A})}}\right] \leq 1 + e.$$

Proof. Let $1 - c = \log_n(2e) + \frac{2}{n-2}$. It is easy to verify that c satisfies the condition of Lemma 5 and $n^c < \frac{n}{2}$. So the above three sets are collectively exhaustive and mutually exclusive, and we have

$$\begin{aligned}
&\mathop{\mathbb{E}}_{a_{ij}\sim \mathrm{U}}\left[\frac{SW_{OPT(\mathrm{A})}}{SW_{RP(\mathrm{A})}}\right] = \Pr\left[SW_{RP(\mathrm{A})} \geq \frac{n}{2}\right] \cdot \mathop{\mathbb{E}}_{SW_{RP(\mathrm{A})} \geq \frac{n}{2}}\left[\frac{SW_{OPT(\mathrm{A})}}{SW_{RP(\mathrm{A})}}\right] + \\
&\Pr\left[n^c < SW_{RP(\mathrm{A})} < \frac{n}{2}\right] \cdot \mathop{\mathbb{E}}_{n^c < SW_{RP(\mathrm{A})} < \frac{n}{2}}\left[\frac{SW_{OPT(\mathrm{A})}}{SW_{RP(\mathrm{A})}}\right] + \\
&\Pr\left[SW_{RP(\mathrm{A})} \leq n^c\right] \cdot \mathop{\mathbb{E}}_{SW_{RP(\mathrm{A})} \leq n^c}\left[\frac{SW_{OPT(\mathrm{A})}}{SW_{RP(\mathrm{A})}}\right] \\
&\leq \Pr\left[SW_{RP(\mathrm{A})} \geq \frac{n}{2}\right] \cdot \frac{n}{n/2} + (1 - \Pr\left[SW_{RP(\mathrm{A})} \geq \frac{n}{2}\right]) \cdot \frac{n}{n^c} + \Pr\left[SW_{RP(\mathrm{A})} \leq n^c\right] \cdot O(\sqrt{n}) \\
&< n^{1-c} + (2 - n^{1-c}) \cdot \Pr\left[SW_{RP(\mathrm{A})} \geq \frac{n}{2}\right] + \frac{e^2}{\sqrt{2\pi}} \cdot \sqrt{n} \cdot \left(\frac{2e}{n^{1-c}}\right)^{n-2} \cdot O(\sqrt{n})
\end{aligned}$$

By our choice of c, we have $n^{1-c} = 2e \cdot n^{\frac{2}{n-2}}$. So,

$$\begin{aligned}
r_{average} &< 2e \cdot n^{\frac{2}{n-2}} - (2e \cdot n^{\frac{2}{n-2}} - 2) \cdot \Pr\left[SW_{RP(\mathrm{A})} \geq \frac{n}{2}\right] + \frac{e^2}{\sqrt{2\pi}} \cdot \sqrt{n} \cdot \left(\frac{2e}{2e \cdot n^{\frac{2}{n-2}}}\right)^{n-2} \cdot O(\sqrt{n}) \\
&\leq 2e \cdot n^{\frac{2}{n-2}} - (2e \cdot n^{\frac{2}{n-2}} - 2) \cdot \frac{1}{2} + \frac{e^2}{\sqrt{2\pi}} \cdot \frac{1}{n^2} \cdot O(n) \\
&= e \cdot n^{\frac{2}{n-2}} + 1 + \frac{e^2}{\sqrt{2\pi}} \cdot \frac{1}{n^2} \cdot O(n) < 1 + e.
\end{aligned}$$

◀

5 Conclusion and Discussion

In this paper, we extend the worst-case approximation ratio analysis in mechanism design to the smoothed approximation ratio and average-case approximation ratio analysis. For social welfare maximization in one-sided matching problem, we show a clear separation of

the approximation ratio bounds from the smoothed analysis and average-case analysis to the worst-case analysis. Our results partially explain why random priority has been successfully used in practice, although in the worst case the optimal social welfare is $\Theta(\sqrt{n})$ times of what random priority achieves.

There are quite a few emerging open questions in the smoothed and average-case approximation ratio analysis of mechanisms. Firstly, it would be good to improve our upper bounds and to characterize matching lower bound. Secondly, our average-case analysis is based on a uniform distribution; it is open to consider other distributions that resembles real-life applications of the one-sided matching mechanisms. Thirdly, beside unit-range representation, another interesting valuation normalization is unit-sum; it is open to study the smoothed and average-case analysis in that setting. Last but most importantly, our analysis pave the way of characterizing the smoothed and average-case approximation ratio in other mechanism design problems, such as the scheduling problem.

References

1. Atila Abdulkadiroğlu and Tayfun Sönmez. Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica*, pages 689–701, 1998.
2. Atila Abdulkadiroğlu and Tayfun Sönmez. Matching Markets: Theory and Practice. *Advances in Economics and Econometrics (Tenth World Congress)*, pages 3–47, 2013.
3. Saeed Alaei, Jason D. Hartline, Rad Niazadeh, Emmanouil Pountourakis, and Yang Yuan. Optimal auctions vs. anonymous pricing. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS*, pages 1446–1463. IEEE Computer Society, 2015.
4. Aaron Archer, Christos H. Papadimitriou, Kunal Talwar, and Éva Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agents. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 205–214, 2003.
5. David Arthur and Sergei Vassilvitskii. Worst-case and smoothed analysis of the ICP algorithm, with an application to the k-means method. *SIAM J. Comput.*, 39(2):766–782, 2009.
6. I. Ashlagi, F. Fischer, I. Kash, and Ariel D. Procaccia. Mix and match. In *Proceedings of the 11th ACM conference on Electronic commerce (ACM-EC)*, pages 305–314, 2010.
7. Cyril Banderier, René Beier, and Kurt Mehlhorn. Smoothed analysis of three combinatorial problems. In *28th International Symposium, Mathematical Foundations of Computer Science, MFCS*, volume 2747 of *Lecture Notes in Computer Science*, pages 198–207. Springer, 2003.
8. Salvador Barbera. Strategy-proof Social Choice. In K. J. Arrow, A. K. Sen, and K. Suzumura, editors, *Handbook of Social Choice and Welfare*, volume 2, chapter 25. North-Holland: Amsterdam, 2010.
9. Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis of the multi-level feedback algorithm. In *Algorithms for Optimization with Incomplete Information, pages 16–21*, volume 05031 of *Dagstuhl Seminar Proceedings*, 2005.
10. Markus Bläser, Bodo Manthey, and B. V. Raghavendra Rao. Smoothed analysis of partitioning algorithms for euclidean functionals. *Algorithmica*, 66(2):397–418, 2013.
11. Avrim Blum and John Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 905–914, 2002.
12. Anna Bogomolnaia and Hervé Moulin. A New Solution to the Random Assignment Problem. *Journal of Economic Theory*, 100:295–328, 2001.

13 Karl-Heinz Borgwardt. The average number of pivot steps required by the simplex-method is polynomial. *Zeitschr. für OR*, 26(1):157–177, 1982.

14 Endre Boros, Khaled M. Elbassioni, Mahmoud Fouz, Vladimir Gurvich, Kazuhisa Makino, and Bodo Manthey. Stochastic mean payoff games: Smoothed analysis and approximation schemes. In *Automata, Languages and Programming - 38th International Colloquium, ICALP Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 147–158. Springer, 2011.

15 Shuchi Chawla and Balasubramanian Sivan. Bayesian algorithmic mechanism design. *SIGecom Exchanges*, 13(1):5–49, 2014.

16 Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *J. ACM*, 56(3), 2009. doi:10.1145/1516512.1516516.

17 George Christodoulou, Elias Koutsoupias, and Angelina Vidali. A lower bound for scheduling mechanisms. *Algorithmica*, 55(4):729–740, 2009.

18 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

19 Shaddin Dughmi and Arpita Ghosh. Truthful assignment without money. In *ACM Conference on Electronic Commerce*, pages 325–334, 2010.

20 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. *Algorithmica*, 68(1):190–264, 2014.

21 Aris Filos-Ratsikas, Søren Kristoffer Stiil Frederiksen, and Jie Zhang. Social welfare in one-sided matchings: Random priority and beyond. In *7th International Symposium on Algorithmic Game Theory (SAGT)*, pages 1–12, 2014.

22 Aris Filos-Ratsikas, Minming Li, Jie Zhang, and Qiang Zhang. Facility location with double-peaked preferences. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 893–899. AAAI Press, 2015.

23 Mahmoud Fouz, Manfred Kufleitner, Bodo Manthey, and Nima Zeini Jahromi. On smoothed analysis of quicksort and hoare's find. *Algorithmica*, 62(3-4):879–905, 2012.

24 M. Guo and V. Conitzer. Strategy-proof allocation of multiple items between two agents without payments or priors. In *Ninth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS)*, volume 10, pages 881–888, 2010.

25 Jason D. Hartline and Brendan Lucier. Bayesian algorithmic mechanism design. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 301–310, 2010.

26 Jason D. Hartline and Tim Roughgarden. Simple versus optimal mechanisms. In *Proceedings 10th ACM Conference on Electronic Commerce, ACM-EC*, pages 225–234. ACM, 2009.

27 Aanund Hylland and Richard Zeckhauser. The Efficient Allocation of Individuals to Positions. *The Journal of Political Economy*, 87(2):293–314, 1979.

28 Elias Koutsoupias. Scheduling without payments. *Theory Comput. Syst.*, 54(3):375–387, 2014.

29 Elias Koutsoupias and Angelina Vidali. A lower bound of $1+\phi$ for truthful scheduling mechanisms. In *the 32nd International Symposium, Mathematical Foundations of Computer Science, MFCS*, volume 4708 of *Lecture Notes in Computer Science*, pages 454–464. Springer, 2007.

30 Daniel J. Lehmann, Liadan O'Callaghan, and Yoav Shoham. Truth revelation in approximately efficient combinatorial auctions. *J. ACM*, 49(5):577–602, 2002.

31 Bodo Manthey and Rüdiger Reischuk. Smoothed analysis of binary search trees. *Theor. Comput. Sci.*, 378(3):292–315, 2007.

32 Bodo Manthey and Heiko Röglin. Smoothed analysis: Analysis of algorithms beyond worst case. *it - Information Technology*, 53(6):280–286, 2011.

33 Timo Mennle and Sven Seuken. An axiomatic approach to characterizing and relaxing strategyproofness of one-sided matching mechanisms. In *Proceedings of the 15th ACM Conference on Economics and Computation*, pages 37–38, 2014.

34 Ahuva Mu'alem and Noam Nisan. Truthful approximation mechanisms for restricted combinatorial auctions. *Games and Economic Behavior*, 64(2):612–631, 2008.

35 Noam Nisan and Amir Ronen. Algorithmic mechanism design. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, STOC*, pages 129–140, 1999.

36 Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1-2):166–196, 2001.

37 Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors. *Algorithmic Game Thoery*. Cambridge University Press, 2007.

38 Jay Sethuraman Parag A. Pathak. Lotteries in student assignment: An equivalence result. *Theoretical Economics*, 6:1–17, 2011.

39 Ariel D Procaccia and Moshe Tennenholtz. Approximate mechanism design without money. In *Proceedings of the 10th ACM Conference on Electronic Commerce*, pages 177–186. ACM, 2009.

40 Arvind Sankar, Daniel A. Spielman, and Shang-Hua Teng. Smoothed analysis of the condition numbers and growth factors of matrices. *SIAM J. Matrix Analysis Applications*, 28(2):446–476, 2006.

41 Guido Schäfer and Naveen Sivadasan. Topology matters: Smoothed competitiveness of metrical task systems. *Theor. Comput. Sci.*, 341(1-3):216–246, 2005.

42 Tayfun Sönmez and Utku Ünver. Matching, allocation and exchange of discrete resources. *Handbook of Social Economics*, 1A:781–852, 2011.

43 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, STOC*, pages 296–305. ACM, 2001.

44 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of termination of linear programming algorithms. *Math. Program.*, 97(1-2):375–404, 2003.

45 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis: an attempt to explain the behavior of algorithms in practice. *Commun. ACM*, 52(10):76–84, 2009.

46 Lars-Gunnar Svensson. Strategy-proof allocation of indivisble goods. *Social Choice and Welfare*, 16(4):557–567, 1999.

47 Lars-Gunnar Svensson. Strategy-proof allocation of indivisible goods. *Social Choice and Welfare*, 16(4):557–567, 1999.

48 Wojciech Szpankowsk. *Average case analysis of algorithms*. Chapman Hall CRC, 2010.

49 John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton university press, 1953.

50 John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior (60th Anniversary Commemorative Edition)*. Princeton university press, 2007.

51 Lin Zhou. On a Conjecture by Gale about One-Sided Matching Problems. *Journal of Economic Theory*, 52:123–135, 1990.

Time Complexity of Constraint Satisfaction via Universal Algebra*

Peter Jonsson[1], Victor Lagerkvist[2], and Biman Roy[3]

1 **Department of Computer and Information Science, Linköping University, Linköping, Sweden**
`peter.jonsson@liu.se`
2 **Institut für Algebra, TU Dresden, Dresden, Germany**
`victor.lagerqvist@tu-dresden.de`
3 **Department of Computer and Information Science, Linköping University, Linköping, Sweden**
`biman.roy@liu.se`

Abstract

The *exponential-time hypothesis* (ETH) states that 3-SAT is not solvable in subexponential time, i.e. not solvable in $O(c^n)$ time for arbitrary $c > 1$, where n denotes the number of variables. Problems like k-SAT can be viewed as special cases of the *constraint satisfaction problem* (CSP), which is the problem of determining whether a set of constraints is satisfiable. In this paper we study the worst-case time complexity of NP-complete CSPs. Our main interest is in the CSP problem parameterized by a constraint language Γ (CSP(Γ)), and how the choice of Γ affects the time complexity. It is believed that CSP(Γ) is either tractable or NP-complete, and the *algebraic CSP dichotomy conjecture* gives a sharp delineation of these two classes based on algebraic properties of constraint languages. Under this conjecture and the ETH, we first rule out the existence of subexponential algorithms for finite-domain NP-complete CSP(Γ) problems. This result also extends to certain infinite-domain CSPs and structurally restricted CSP(Γ) problems. We then begin a study of the complexity of NP-complete CSPs where one is allowed to arbitrarily restrict the values of individual variables, which is a very well-studied subclass of CSPs. For such CSPs with finite domain D, we identify a relation S_D such that (1) CSP($\{S_D\}$) is NP-complete and (2) if CSP(Γ) over D is NP-complete and solvable in $O(c^n)$ time, then CSP($\{S_D\}$) is solvable in $O(c^n)$ time, too. Hence, the time complexity of CSP($\{S_D\}$) is a lower bound for all CSPs of this particular kind. We also prove that the complexity of CSP($\{S_D\}$) is decreasing when $|D|$ increases, unless the ETH is false. This implies, for instance, that for every $c > 1$ there exists a finite-domain Γ such that CSP(Γ) is NP-complete and solvable in $O(c^n)$ time.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, G.2.0 Discrete Mathematics General.

Keywords and phrases Clone Theory, Universal Algebra, Constraint Satisfaction Problems

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.17

1 Introduction

The *constraint satisfaction problem* over a constraint language Γ (CSP(Γ)) is the computational decision problem of verifying whether a set of constraints over Γ is satisfiable or not.

* The second author has received funding from the DFG-funded project "Homogene Strukturen, Bedingungserfüllungsprobleme, und topologische Klone" (Project number 622397). The third author is partially supported by the *National Graduate School in Computer Science* (CUGS), Sweden.

This problem is widely studied from both a theoretical and a practical standpoint. From a practical point of view this problem can be used to model many natural problems occurring in real-world applications. From a more theoretical point of view the CSP problem is (among several other things) of great interest due to its connections with *universal algebra*. It is widely believed that finite-domain CSP problems admit a dichotomy between tractable and NP-complete problems, and the so-called *algebraic approach* has been used to conjecture an exact borderline between tractable and NP-complete problems [15]. This conjectured borderline is sometimes called the *algebraic CSP dichotomy conjecture*. The gist of the algebraic approach is to associate an algebra, a set of functions satisfying a certain closure property, to each constraint language. This associated algebra is usually referred to as the *polymorphisms* of a constraint language, and is known to determine the complexity of a CSP problem up to polynomial-time many-one reductions [26]. However, the mere fact that two CSPs are polynomial-time interreducible does not offer much insight into their relative worst-case time complexity. For example, on the one hand, it has been conjectured that the Boolean satisfiability problem with unrestricted clause length, SAT, is not solvable strictly faster than $O(2^n)$, where n denotes the number of variables [23]. On the other hand, k-SAT is known to be solvable strictly faster than $O(2^n)$ for every $k \geq 1$ [22], and even more efficient algorithms are known for severely restricted satisfiability problems such as 1-in-3-SAT [36]. This discrepancy in complexity stems from the fact that a polynomial time reduction can change the structure of an instance and e.g. introduce a large number of fresh variables. Hence, it is worthwhile to study the complexity of NP-complete CSPs using more fine-grained notions of reductions. To make this a bit more precise, given a constraint language Γ we let

$$\mathsf{T}(\Gamma) = \inf\{c \mid \mathrm{CSP}(\Gamma) \text{ is solvable in time } 2^{cn}\}$$

where n denotes the number of variables. If $\mathsf{T}(\Gamma) = 0$ then $\mathrm{CSP}(\Gamma)$ is said to be solvable in *subexponential time*, and the conjecture that 3-SAT is not solvable in subexponential time is known as the *exponential-time hypothesis* (ETH) [23]. It is worth remarking that no concrete values of $\mathsf{T}(\Gamma)$ are known when $\mathrm{CSP}(\Gamma)$ is NP-complete. Despite this, studying properties of the function T can still be of great interest since such properties can be used to compare and relate the worst-case running times of NP-complete CSP problems. Moreover, for Boolean constraint languages, several properties of the function T are known. For example, it is known that there exists a finite Boolean constraint language Γ such that $\mathrm{CSP}(\Gamma)$ is NP-complete and $\mathsf{T}(\Gamma) = 0$ if and only if $\mathsf{T}(\Gamma) = 0$ for *every* Boolean constraint language Γ [27]. Hence, even though the status of the ETH is unclear at the moment, finding a subexponential time algorithm for one NP-complete Boolean CSP problem is tantamount to being able to solve every Boolean CSP problem in subexponential time. It is also known that there exists a Boolean relation R such that $\mathrm{CSP}(\{R\})$ is NP-complete but $\mathsf{T}(\{R\}) \leq \mathsf{T}(\Gamma)$ for *every* Boolean constraint language Γ such that $\mathrm{CSP}(\Gamma)$ is NP-complete. In Jonsson et al. [27] this problem is referred to as the *easiest NP-complete Boolean CSP problem*. The existence of this relation e.g. rules out the possibility that for each Boolean constraint language Γ there exists Δ such that $\mathsf{T}(\Delta) < \mathsf{T}(\Gamma)$ — a scenario which otherwise would have been compatible with the ETH. These results were obtained by considering more refined algebras than polymorphisms, so-called *partial polymorphisms*. We will describe this algebraic approach in greater detail later on, but the most important property is that the partial polymorphisms of finite constraint languages give rise to a partial order $\sqsubseteq$ with the property that if $\Gamma \sqsubseteq \Delta$, then $\mathsf{T}(\Gamma) \leq \mathsf{T}(\Delta)$. We remark that partial polymorphisms are not only useful when studying CSPs with this very fine-grained notion of complexity, but have also been used to study the classical complexity of many different computational problems where polymorphisms are not applicable [3, 4, 11, 14, 21].

Hence, even though no concrete values are known for $\mathsf{T}(\Gamma)$ when CSP(Γ) is NP-complete, quite a lot is known concerning the relationship between $\mathsf{T}(\Gamma)$ and $\mathsf{T}(\Delta)$ for Boolean Γ and Δ. In this paper we study similar properties of the function T for constraint languages defined over arbitrary finite domains. After having introduced the necessary definitions in Section 2, in Section 3 we consider the existence of subexponential time algorithms for NP-complete CSP problems, in light of the ETH and the algebraic CSP dichotomy conjecture. For this question we obtain a complete understanding and prove that, assuming the algebraic CSP dichotomy conjecture, the ETH is false if and only if (1) there exists a finite constraint language Γ over a finite domain such that CSP(Γ) is NP-complete and $\mathsf{T}(\Gamma) = 0$, if and only if (2) $\mathsf{T}(\Gamma) = 0$ for every finite constraint language Γ defined over a finite domain. In other words, finding a subexponential time algorithm for a single NP-complete, finite-domain CSP problem is tantamount to being able to solve all CSP problems in subexponential time. We also study structurally restricted CSPs where the maximum number of constraints a variable may appear in is bounded by a constant B (CSP(Γ)-B). For problems of this form our results are not as sharp, but we prove that, again assuming the algebraic CSP dichotomy conjecture, that if CSP(Γ) is NP-complete and Γ satisfies an additional algebraic condition, then there exists a constant B such that CSP(Γ)-B is not solvable in subexponential time (unless the ETH is false). We also remark that our proof extends to certain constraint languages defined over infinite domain, and give several examples of infinite-domain NP-complete CSP problems that are not solvable in subexponential time, unless the ETH is false. These results may be interesting to compare to those of de Haan et al. [17], who study subexponential algorithms for structurally restricted CSPs. One crucial difference to our results is that de Haan et al. do not consider constraint language restrictions. For example, it is proven that CSP(Δ)-2, where Δ is the set of all finitary relations of finite cardinality, is not solvable in subexponential time unless the ETH is false. However, a result of this form tells us very little about the complexity of CSP(Γ)-2 for specific constraint languages, since it does not imply that CSP(Γ)-2 is not solvable in subexponential time for every NP-complete CSP(Γ)-2.

We have thus established that $\mathsf{T}(\Gamma) > 0$ for every NP-complete, finite-domain CSP(Γ), assuming the ETH and the algebraic CSP dichotomy conjecture. This immediately raises the question of which further insights can be gained concerning the behaviour of the function T. For example, for a fixed finite domain, is it possible to construct an infinite chain of NP-complete CSPs with strictly decreasing complexity such that T tends to 0? We study such questions in Section 4 for CSPs where one in an instance is allowed to restrict the values of individual variables arbitrarily. This restricted CSP problem is particularly well-studied, and it is used as *the* definition of CSPs in many cases: see, for instance, the textbook by Russell and Norvig [33, Section 3.7] and the handbook by Rossi et al. [32, Section 2]. This may be viewed as restricting oneself to constraint languages that contain all unary relations. A closely related restriction (that is typically used when studying CSPs from the algebraic viewpoint) is that every unary relation is primitively positively definable in Γ (see Section 2). Such constraint languages are known as *conservative*. These two restrictions are computationally equivalent up to polynomial-time many-one reductions but it is not known whether they are equivalent under reductions that preserve time complexity. Thus, we need to separate them, so we say that a constraint language that contains all unary relations is *ultraconservative*. We note that the algebraic CSP dichotomy conjecture has been verified to hold for the conservative CSPs [12] so it holds for ultraconservative CSPs, too. We show that for every finite domain D there exists a relation S_D such that CSP($\{S_D\}$) is NP-complete and $\mathsf{T}(\{S_D\}) = \mathsf{T}(\{S_D\} \cup 2^D) \leq \mathsf{T}(\Gamma)$ for every ultraconservative and NP-complete CSP(Γ) over D. This relation will be formally defined in Section 4.1, but is worth pointing out that S_D

contains only three tuples and that CSP($\{S_D\}$) can be viewed as a higher-domain variant of the monotone 1-in-3-SAT problem. We refer to CSP($\{S_D\} \cup 2^D$) as the *easiest NP-complete ultraconservative CSP problem over* D[1]. Note that the properties of the relation S_D rule out the possibility of an infinite sequence of ultraconservative languages $\Gamma_1, \Gamma_2, \ldots$ such that each CSP(Γ_i) is NP-complete and $\mathsf{T}(\Gamma_i)$ tends to 0, but also have stronger implications, since the value $\mathsf{T}(\{S_D\})$ is a conditional lower bound for the complexity of *all* NP-complete, ultraconservative CSPs over D.

To prove these results we have to overcome several major obstacles. Similar to Jonsson et al. [27]) we use partial polymorphisms instead of total polymorphisms in order to achieve more fine-grained notions of reductions. However, the proof strategy used in Jonsson et al. [27] does not work for arbitrary finite domains since it requires a comprehensive understanding of the polymorphisms of constraint languages resulting in NP-complete CSPs, which is only known for the Boolean domain [29]. Our first observation to tackle this difficulty is that the reformulation of conservative CSP dichotomy theorem making use of *primitive positive interpretations* (pp-interpretations) is useful in our context. At the moment, we may think of a pp-interpretation as a tool which allows us to compare the expressitivity of constraint languages defined over different domains, modulo logical formulas consisting of existential quantification, conjunction, and equality constraints. It is well-known that pp-interpretations can be used to obtain polynomial-time reductions between CSPs, and that a conservative CSP(Γ) problem is NP-complete if and only if Γ pp-interprets 3-SAT [1, 12]. However, as already pointed out, such reductions are not useful when studying CSPs with respect to the function T, and it is a priori not evident how the assumption that Γ can pp-interpret 3-SAT can be used to show that $\mathsf{T}(\{S_D\}) \leq \mathsf{T}(\Gamma)$. Using properties of conservative constraint languages and quantifier-elimination techniques we in Section 4.1 first show that this assumption can be used to prove there exists a relation R over D of cardinality 3 such that (1) CSP($\{R\}$) is NP-complete and (2) $\mathsf{T}(\{R\}) \leq \mathsf{T}(\Gamma)$. However, this is not enough in order to isolate a unique easiest problem, since there for every finite domain exists a large number of such relations. In Section 4.2, using a combination of partial clone theory and size-preserving reductions, we show that $\mathsf{T}(\{S_D\}) \leq \mathsf{T}(\{R\})$ for every such relation R of cardinality 3. We then analyse the time complexity of the problem CSP($\{S_D\}$) and prove that $\mathsf{T}(\{S_D\})$ tends to 0 for increasing values of $|D|$. This also shows, despite the fact that no finite-domain NP-complete CSP(Γ) is solvable in subexponential time (if the algebraic CSP dichotomy conjecture and the ETH are true), that one for every $c > 0$ can find Γ over a finite domain such that CSP(Γ) is NP-complete and solvable in $O(2^{cn})$ time. When all of these results are adjoined, they demonstrate that the function T can indeed be analysed without an extensive knowledge of the polymorphisms related to a constraint language.

2 Preliminaries

2.1 Constraint Languages and the Constraint Satisfaction Problem

A k-ary *relation* R over a set D is a subset of D^k, and we write $\mathrm{ar}(R) = k$ to denote its arity. A finite set of relations Γ over a set D is called a *constraint language*. Given two tuples s and t we let $s^\frown t$ denote the concatenation of s and t, i.e., if $s = (s_1, \ldots, s_{k_1})$ and $t = (t_1, \ldots, t_{k_2})$ then $s^\frown t = (s_1, \ldots, s_{k_1}, t_1, \ldots, t_{k_2})$. If t is an n-ary tuple we let $t[i]$ denote its ith element and $\mathrm{Proj}_{i_1,\ldots,i_{n'}}(t) = (t[i_1], \ldots, t[i_{n'}])$, $n' \leq n$, denote the *projection*

[1] Note that 2^D is the set of all unary relations over D.

of t on the coordinates $i_1, \ldots, i_{n'} \in \{1, \ldots, n\}$. Similarly, if R is an n-ary relation we let $\mathrm{Proj}_{i_1,\ldots,i_{n'}}(R) = \{\mathrm{Proj}_{i_1,\ldots,i_{n'}}(t) \mid t \in R\}$. We write Eq_D for the equality relation $\{(x, x) \mid x \in D\}$. If there is no risk for confusion we omit the subscript and simply write Eq. For each $d \in D$ we write R^d for the unary, constant relation $\{(d)\}$. We will occasionally represent relations by first-order formulas, and if $\varphi(x_1, \ldots, x_k)$ is a first-order formula with free variables $x_1, \ldots, x_k$ then we write $R(x_1, \ldots, x_k) \equiv \varphi(x_1, \ldots, x_k)$ to define the relation $R = \{(f(x_1), \ldots, f(x_k)) \mid f \text{ is a model of } \varphi(x_1, \ldots, x_k)\}$. As a graphical representation, we will sometimes view a k-ary relation $R = \{t_1, \ldots, t_m\}$ as an $m \times k$ matrix where the columns of the matrix enumerate the arguments of the relation (in some fixed ordering). For example, $\left(\begin{smallmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{smallmatrix}\right)$ represents the relation $\{(0, 0, 1, 1), (0, 1, 0, 1)\}$.

The *constraint satisfaction problem* over a constraint language Γ over D ($\mathrm{CSP}(\Gamma)$) is the computational decision problem defined as follows.

INSTANCE: A set V of variables and a set C of constraint applications $R(x_1, \ldots, x_k)$ where $R \in \Gamma$, $\mathrm{ar}(R) = k$, and $x_1, \ldots, x_k \in V$.
QUESTION: Does there exist $f : V \to D$ such that $(f(x_1), \ldots, f(x_k)) \in R$ for each $R(x_1, \ldots, x_k)$ in C?

If $\Gamma = \{R\}$ is singleton then we write $\mathrm{CSP}(R)$ instead of $\mathrm{CSP}(\{R\})$, and if Γ is Boolean we typically write $\mathrm{SAT}(\Gamma)$ instead of $\mathrm{CSP}(\Gamma)$. We let $\mathbb{B} = \{0, 1\}$. For example, let $R_{1/3}^{\neq\neq\neq 01} = \{(0, 0, 1, 1, 1, 0, 0, 1), (0, 1, 0, 1, 0, 1, 0, 1), (1, 0, 0, 0, 1, 1, 0, 1)\}$. The SAT problem over $R_{1/3}^{\neq\neq\neq 01}$ can be seen as a variant of 1-in-3-SAT where each variable in each constraint has a complementary variable. We will return to this SAT problem several times in the sequel. For each $k \geq 3$ let Γ_{SAT}^k be the constraint language which for every $t \in \mathbb{B}^k$ contains the relation $\mathbb{B}^k \setminus \{t\}$. Hence, $\mathrm{SAT}(\Gamma_{\mathrm{SAT}}^k)$ can be viewed as an alternative formulation of k-SAT.

2.2 Primitive Positive Definitions and Interpretations

Let Γ be a constraint language. A k-ary relation R is said to have a *primitive positive definition* (pp-definition) over Γ if $R(x_1, \ldots, x_k) \equiv \exists y_1, \ldots, y_{k'} . R_1(\mathbf{x_1}) \wedge \ldots \wedge R_m(\mathbf{x_m})$, where each $R_i \in \Gamma \cup \{\mathrm{Eq}\}$ and each $\mathbf{x_i}$ is an $\mathrm{ar}(R_i)$-ary tuple of variables over $x_1, \ldots, x_k$, $y_1, \ldots, y_{k'}$. In addition, if the primitive positive formula does not contain any existentially quantified variables, we say that it is a *quantifier-free primitive positive formula* (qfpp), and if it does not contain any equality constraints we say that it is a *equality-free primitive positive formula* (efpp). For example, the reader can verify that the textbook reduction from k-SAT to $(k-1)$-SAT, where a clause of length k is replaced by clauses of length $k-1$ making use of one fresh variable, can be formulated as a pp-definition but not as a qfpp-definition. We write $\langle\Gamma\rangle$ (respectively $\langle\Gamma\rangle_{\exists}$) to denote the smallest set of relations containing Γ and which is closed under pp-definitions (respectively qfpp-definitions). If $\Gamma = \{R\}$ is singleton then we instead write $\langle R\rangle$ and $\langle R\rangle_{\exists}$. Note that $\langle\Gamma\rangle$ is closed under projections, in the sense that if $R \in \langle\Gamma\rangle$ then $\mathrm{Proj}_{i_1,\ldots,i_n}(R) \in \langle\Gamma\rangle$ for all $i_1, \ldots, i_n \in \{1, \ldots, \mathrm{ar}(R)\}$, but that this does not necessarily hold for $\langle\Gamma\rangle_{\exists}$. Jeavons [25] proved the following important result.

▶ **Theorem 1.** *If Γ is a constraint language and Δ is a finite subset of $\langle\Gamma\rangle$, then CSP(Δ) is polynomial-time reducible to CSP(Γ).*

Theorem 1 naturally holds also for relations defined by qfpp- or efpp-formulas. However, there are additional advantages of these more restricted ways of defining relations and we will return to them later on. We are now ready to define the concept of primitive positive interpretations.

▶ **Definition 2.** Let D and E be two domains and let Γ and Δ be two constraint languages over D and E, respectively. A *primitive positive interpretation* (pp-interpretation) of Δ over Γ consists of a d-ary relation $F \subseteq D^d$ and a surjective function $f : F \to E$ such that $F, f^{-1}(\mathrm{Eq}_E) \in \langle\Gamma\rangle$ and $f^{-1}(R) \in \langle\Gamma\rangle$ for every $R \in \Delta$, where $f^{-1}(R)$, $\mathrm{ar}(R) = k$, denotes the $(k \cdot d)$-ary relation

$$\{(x_{1,1}, \ldots, x_{1,d}, \ldots, x_{k,1}, \ldots, x_{k,d}) \in D^{k \cdot d} \mid (f(x_{1,1}, \ldots, x_{1,d}), \ldots, f(x_{k,1}, \ldots, x_{k,d})) \in R\}.$$

The main purpose of pp-interpretations is to relate constraint languages which might be incomparable with respect to pp-definitions. For an example, let us consider the relation $R_{\neq} = \{(x, y) \in \{0, 1, 2\}^2 \mid x \neq y\}$, and observe that $\mathrm{CSP}(\{R_{\neq}\})$ corresponds to the 3-coloring problem. We invite the reader to verify that the standard reduction from 3-coloring to 3-SAT can be phrased as a pp-interpretation of $R_{\neq}$ over Γ^3_{SAT}, but that this reduction cannot be expressed via pp-definitions due to the different domains. Hence, pp-interpretations are generalizations of pp-definitions, and can be used to obtain polynomial-time reductions between CSPs.

▶ **Theorem 3** (cf. Theorem 5.5.6 in Bodirsky [5]). *If Γ, Δ are constraint languages and there is a pp-interpretation of Δ over Γ, then CSP(Δ) is polynomial-time reducible to CSP(Γ).*

2.3 Polymorphisms and Partial Polymorphisms

Let f be a k-ary function over a finite domain D. We say that f is a *polymorphism* of an n-ary relation R over D if $f(t_1, \ldots, t_k) \in R$ for each k-ary sequence of tuples $t_1, \ldots, t_k \in R$. Here, and in the sequel, we use $f(t_1, \ldots, t_k)$ to denote the componentwise application of the function f to the tuples $t_1, \ldots, t_k$, i.e., $f(t_1, \ldots, t_k)$ is a shorthand for the n-ary tuple $(f(t_1[1], \ldots, t_k[1]), \ldots, f(t_1[n], \ldots, t_k[n]))$. Similarly, if f is a partial function over D, we say that f is a *partial polymorphism* of an n-ary relation R over D if $f(t_1, \ldots, t_k) \in R$ for every sequence $t_1, \ldots, t_k$ such that $f(t_1, \ldots, t_k)$ is defined for each componentwise application. If f is a polymorphism or a partial polymorphism of a relation R then we occasionally also say that R is *invariant* under f. We let $\mathrm{Pol}(R)$ and $\mathrm{pPol}(R)$ denote the set of all polymorphisms, respectively partial polymorphisms, of the relation R. Similarly, for a constraint language Γ, we write $\mathrm{Pol}(\Gamma)$ for the set $\bigcap_{R \in \Gamma} \mathrm{Pol}(R)$, and $\mathrm{pPol}(\Gamma)$ for the set $\bigcap_{R \in \Gamma} \mathrm{pPol}(R)$. We write $\mathrm{Inv}(F)$ to denote the set of all relations invariant under the set of total or partial functions F. It is known that $\mathrm{Inv}(\mathrm{Pol}(\Gamma)) = \langle\Gamma\rangle$ and that $\mathrm{Inv}(\mathrm{pPol}(\Gamma)) = \langle\Gamma\rangle_{\not\exists}$, giving rise to the following *Galois connections*.

▶ **Theorem 4** ([9, 10, 19, 31]). *Let Γ and Γ' be two constraint languages. Then $\Gamma \subseteq \langle\Gamma'\rangle$ if and only if $\mathrm{Pol}(\Gamma') \subseteq \mathrm{Pol}(\Gamma)$ and $\Gamma \subseteq \langle\Gamma'\rangle_{\not\exists}$ if and only if $\mathrm{pPol}(\Gamma') \subseteq \mathrm{pPol}(\Gamma)$.*

2.4 Time Complexity and Size-Preserving Reductions

Given a constraint language Γ we let $\mathsf{T}(\Gamma) = \inf\{c \mid \mathrm{CSP}(\Gamma) \text{ is solvable in time } 2^{cn}\}$ where n denotes the number of variables in a given instance. If $\mathsf{T}(\Gamma) = 0$ then $\mathrm{CSP}(\Gamma)$ is said to be solvable in *subexponential time*. The conjecture that $\mathrm{SAT}(\Gamma^3_{\mathrm{SAT}}) > 0$ is known as the *exponential-time hypothesis* (ETH) [24]. We now introduce a type of reduction useful for studying the complexity of CSPs with respect to the function T.

▶ **Definition 5.** Let Γ and Δ be two constraint languages. The function f from the instances of $\mathrm{CSP}(\Gamma)$ to the instances of $\mathrm{CSP}(\Delta)$ is a *many-one linear variable reduction* (LV-reduction) with parameter $d \geq 0$ if (1) f is a polynomial-time many-one reduction from $\mathrm{CSP}(\Gamma)$ to

$\mathrm{CSP}(\Delta)$ and (2) $|V'| = d \cdot |V| + O(1)$ where V, V' are the set of variables in I and $f(I)$, respectively.

The term CV-reduction, short for *constant variable reduction*, is used to denote LV-reductions with parameter 1, and we write $\mathrm{CSP}(\Gamma) \leq^{\mathrm{CV}} \mathrm{CSP}(\Delta)$ when $\mathrm{CSP}(\Gamma)$ has a CV-reduction to $\mathrm{CSP}(\Delta)$. It follows that if $\mathrm{CSP}(\Gamma) \leq^{\mathrm{CV}} \mathrm{CSP}(\Delta)$ then $\mathsf{T}(\Gamma) \leq \mathsf{T}(\Delta)$, and if $\mathrm{CSP}(\Gamma)$ LV-reduces to $\mathrm{CSP}(\Delta)$ then $\mathsf{T}(\Gamma) = 0$ if $\mathsf{T}(\Delta) = 0$. We have the following theorem from Jonsson et al. [27], relating the partial polymorphisms of constraint languages with the existence of CV-reductions.

▶ **Theorem 6** ([27]). *Let D be a finite domain and let Γ and Δ be two constraint languages over D. If $\mathrm{pPol}(\Delta) \subseteq \mathrm{pPol}(\Gamma)$ then $\mathrm{CSP}(\Gamma) \leq^{\mathrm{CV}} \mathrm{CSP}(\Delta)$.*

We remark that the original proof only concerned Boolean constraint languages but that the same proof also works for arbitrary finite domains. Using Theorem 6 and algebraic techniques from Schnoor and Schnoor [35], Jonsson et al. [27] proved that $\mathsf{T}(\{R_{1/3}^{\neq\neq 01}\}) \leq \mathsf{T}(\Gamma)$ for any finite Γ such that $\mathrm{SAT}(\Gamma)$ is NP-complete. This problem was referred to as the *easiest NP-complete* SAT *problem*. We will not go into the details but remark that the proof idea does not work for arbitrary finite domains since it requires a characterisation of every $\mathrm{Pol}(\Gamma)$ such that $\mathrm{CSP}(\Gamma)$ is NP-complete. Such a list is known for the Boolean domain due to Post [29] and Schaefer [34], but not for larger domains.

2.5 Complexity of CSP

Let Γ be a constraint language over a finite domain D. We say that Γ is *idempotent* if $R^d \in \langle\Gamma\rangle$ for every $d \in D$, *conservative* if $2^D \subseteq \langle\Gamma\rangle$, and *ultraconservative* if $2^D \subseteq \Gamma$. A unary function $f \in \mathrm{Pol}(\Gamma)$ is said to be an *endomorphism*, and if f in addition is bijective it is said to be an *automorphism*. A constraint language Γ is a *core* if every endomorphism is an automorphism. The following theorem is well-known, see e.g. Barto [1], but is usually expressed in term of polynomial-time many-one reductions instead of CV-reductions.

▶ **Theorem 7.** *Let Γ be a core constraint language over the domain $\{d_0, \ldots, d_{k-1}\}$. Then $\mathrm{CSP}(\Gamma \cup \{R^{d_0}, \ldots, R^{d_{k-1}}\}) \leq^{\mathrm{CV}} \mathrm{CSP}(\Gamma)$.*

If Γ is a constraint language over $D = \{d_0, \ldots, d_{k-1}\}$, then $\Gamma \cup \{R^{d_0}, \ldots, R^{d_{k-1}}\}$ is both idempotent and a core since its only endomorphism is the identity function on D. The *CSP dichotomy conjecture* states that for any Γ over a finite domain, $\mathrm{CSP}(\Gamma)$ is either tractable or NP-complete [18]. This conjecture was later refined by Bulatov et al. [15] to also induce a sharp characterization of the tractable and intractable cases, expressed in terms of algebraic properties of the constraint language, and is usually called the *algebraic CSP dichotomy conjecture*. We will use the following variant of the conjecture which is expressed in terms of pp-interpretations.

▶ **Conjecture 8.** *[1, 15] Let Γ be an idempotent constraint language over a finite domain. Then $\mathrm{CSP}(\Gamma)$ is NP-complete if Γ pp-interprets Γ_{SAT}^3 and tractable otherwise.*

It is worth remarking that if Γ pp-interprets Γ_{SAT}^3 then Γ can pp-interpret every finite-domain relation [5, Theorem 5.5.17].

3 Subexponential Time Complexity

For Boolean constraint languages it has been proven that $\mathrm{SAT}(\Gamma_{\mathrm{SAT}}^3)$ is solvable in subexponential time if and only if there exists a finite Boolean constraint language Γ such that

SAT(Γ) is NP-complete and solvable in subexponential time [27]. We will strengthen this result to arbitrary domains and prove that CSP(Γ) is never solvable in subexponential time if Γ can pp-interpret Γ^3_{SAT}, unless the ETH is false. The result can also be extended to certain structurally restricted CSPs. The *degree* of a variable $x \in V$ of an instance (V, C) of CSP(Γ) is the number of constraints in C containing x. We let CSP(Γ)-B, $B \geq 1$, denote the restricted CSP(Γ) problem where each variable occurring in an instance has degree at most B. We then obtain the following theorem, whose proof can be found in the extended preprint [28].

▶ **Theorem 9.** *Assume that the ETH is true and let* Γ *be a finite constraint language over a domain* D *such that* Γ *pp-interprets* Γ^3_{SAT}*. Then* CSP(Γ) *is not solvable in subexponential time, and if* Γ *efpp-defines* Eq_D *then there exists a constant* B*, depending only on* Γ*, such that* CSP(Γ)-B *is not solvable in subexponential time.*

We have now obtained a complete understanding of subexponential solvability of finite-domain CSPs modulo the ETH.

▶ **Corollary 10.** *Assume that the algebraic CSP dichotomy conjecture is true. Then the following statements are equivalent: (1) the ETH is false, (2),* CSP(Γ) *is solvable in subexponential time for every finite* Γ *over a finite domain, and (3) there exists a finite constraint language* Γ *over a finite domain* D *such that* CSP(Γ) *is NP-complete and subexponential.*

Proof. The implication from (1) to (2) follows from Impagliazzo et al. [24, Theorem 3]. The implication from (2) to (3) is trivial. For the implication from (3) to (1), we first note that $\mathrm{CSP}(\Gamma^c) \leq^{\mathrm{CV}} \mathrm{CSP}(\Gamma)$, where Γ^c is the core of Γ [1, Theorem 3.5]. If Γ^c is expanded with all constants, then Theorem 7 shows that the complexity does not change, and, last, this language can pp-interpret Γ^3_{SAT}, due to the assumption that the algebraic CSP dichotomy conjecture is true, which via Theorem 9 implies that 3-SAT is solvable in subexponential time, and thus that the ETH is false. ◀

For CSP(Γ)-B our results are not as precise since we need the additional assumption that the equality relation is efpp-definable. This is not surprising since the most powerful dichotomy results for CSPs are usually concerned with either constraint language restrictions [12, 15], structural restrictions [17, 20], but rarely both simultaneously. However, in the Boolean domain there are plenty of examples which illustrates how the equality relation may be efpp-defined [16, 27], suggesting that similar techniques may also exist for larger domains.

Theorem 9 also applies to many interesting classes of infinite-domain CSPs. For example, if we consider Γ such that each $R \in \Gamma$ has a first-order definition over the structure $(\mathbb{Q}; <)$, it is known that CSP(Γ) is NP-complete if and only if Γ can pp-interpret Γ^3_{SAT} [5, 7]. Hence, Theorem 9 is applicable, implying that if CSP(Γ) is not solvable in subexponential time if it is NP-complete, unless the ETH fails. More examples of infinite-domain CSPs where Theorem 9 is applicable includes graph satisfiability problems [8] and phylogeny constraints [6]. Note that all of these results hold independently of whether the algebraic CSP dichotomy is true or not. We also remark that the intractable cases of the CSP dichotomy conjecture for certain infinite-domain CSPs are all based on pp-interpretability of Γ^3_{SAT} [2]. If this conjecture is correct, Theorem 9 and the ETH implies that none of these problems are solvable in subexponential time.

4 The Easiest NP-Complete Ultraconservative CSP Problem

The results from Section 3, assuming the algebraic CSP dichotomy conjecture and the ETH, implies that $\mathsf{T}(\Gamma) > 0$ for any finite-domain and NP-complete CSP(Γ). However, it is safe to

say that very little is known about the behaviour of the function T in more general terms. For example, is there for an arbitrary NP-complete CSP(Γ) possible to find an NP-complete CSP(Δ) such that $\mathsf{T}(\Delta) < \mathsf{T}(\Gamma)$? Such a scenario would be compatible with the consequences of Theorem 9. We will show that this is unlikely, and prove that there for every finite domain D exists a relation S_D such that CSP(S_D) is NP-complete but $\mathsf{T}(\{S_D\}) \leq \mathsf{T}(\Gamma)$ for any ultraconservative Γ over D such that CSP(Γ) is NP-complete. To prove this we have divided this section into two parts. In Section 4.1 we show that if Γ is ultraconservative and CSP(Γ) is NP-complete, then there exists a relation $R \in \langle\Gamma\rangle_{\exists'}$ which shares certain properties with the relation $R_{1/3}^{\neq\neq\neq 01}$. In Section 4.2 we use properties of these relations in order to prove that there for every finite domain D is possible to find a relation S_D such that CSP(S_D) is CV-reducible to any other NP-complete and ultraconservative CSP(Γ) problem.

4.1 $S_{\mathbb{B}}$-Extensions

The columns of the matrix representation of the relation $R_{1/3}^{\neq\neq\neq 01}$ from Jonsson et al. [27] (resulting in the easiest NP-complete SAT problem) enumerates all Boolean ternary tuples. We generalize this relation to arbitrary finite domains as follows.

▶ **Definition 11.** For each finite D let $S_D = \{t_1, t_2, t_3\}$ denote the $|D|^3$-ary relation such that there for every $(d_1, d_2, d_3) \in D^3$ exists $1 \leq i \leq |D|^3$ such that $(t_1[i], t_2[i], t_3[i]) = (d_1, d_2, d_3)$.

Hence, similar to $R_{1/3}^{\neq\neq\neq 01}$, the columns of the matrix representation of S_D enumerates all ternary tuples over D. For each D the relation S_D is unique up to permutation of arguments, and although we will usually not be concerned with the exact ordering, we sometimes assume that $S_{\mathbb{B}} = R_{1/3}^{\neq\neq\neq 01}$ and that $\mathrm{Proj}_{1,\ldots,8}(S_D) = S_{\mathbb{B}}$. The notation S_D is a mnemonic for *saturated*, and the reason behind this will become evident in Section 4.2.1. For example, for $\{0, 1, 2\}$ we obtain a relation $\{t_1, t_2, t_3\}$ with 27 distinct arguments such that $(t_1[i], t_2[i], t_3[i]) \in \{0, 1, 2\}^3$ for each $1 \leq i \leq 27$. Jonsson et al. [27] proved that $S_{\mathbb{B}} \in \langle\Gamma\rangle_{\exists'}$ for every Boolean and idempotent constraint language Γ such that SAT(Γ) is NP-complete. This is not true for arbitrary finite domains, and in order to prove an analogous result we will need the following definition.

▶ **Definition 12.** Let R be an n-ary relation of cardinality 3 over a domain D, $|D| \geq 2$. Let $a, b \in D$ be two distinct values. If there exists $i_1, \ldots, i_8 \in \{1, \ldots, n\}$ such that

$$\mathrm{Proj}_{i_1,\ldots,i_8}(R) = \{(a,a,b,b,b,a,a,b), (a,b,a,b,a,b,a,b), (b,a,a,a,b,b,a,b)\},$$

then we say that R is an *$S_{\mathbb{B}}$-extension*.

For example, S_D is an $S_{\mathbb{B}}$-extension for every domain D. Note that CSP(R) is always NP-complete when R is an $S_{\mathbb{B}}$-extension. We will now prove that if CSP(Γ) is NP-complete and Γ is ultraconservative, then Γ can pp-define an $S_{\mathbb{B}}$-extension.

▶ **Lemma 13.** *Let Γ be an ultraconservative constraint language over a finite domain D such that* CSP(Γ) *is NP-complete. Then there exists a relation $R \in \langle\Gamma\rangle$ which is an $S_{\mathbb{B}}$-extension.*

Proof. Since CSP(Γ) is NP-complete and Γ is ultraconservative, Γ can pp-interpret every Boolean relation. Therefore let $f : F \to \mathbb{B}$, $F \subseteq D^d$ denote the parameters in the pp-interpretation of $S_{\mathbb{B}}$, and note that $f^{-1}(S_{\mathbb{B}}) \in \langle\Gamma\rangle$, but that $f^{-1}(S_{\mathbb{B}})$ is not necessarily an $S_{\mathbb{B}}$-extension since it could be the case that $|f^{-1}(S_{\mathbb{B}})| > 3$. Pick two tuples s and t in F such that $f(s) = 0$ and $f(t) = 1$. Such tuples must exist since f is surjective. Now consider the relation $F_1(x_1, \ldots, x_d) \equiv F(x_1, \ldots, x_d) \wedge \{(s[1]), (t[1])\}(x_1) \wedge \ldots \wedge \{(s[d], t[d])\}(x_d)$.

This relation is pp-definable over Γ since Γ is ultraconservative and since $F \in \langle\Gamma\rangle$. By construction, it is clear that $s, t \in F_1$. Assume furthermore than $|F_1| > 2$, i.e., that there exists $u \in F_1 \setminus \{s, t\}$. Assume without loss of generality that $f(u) = 0$, and observe that there for each $i \in \{1, \dots, d\}$ holds that $u[i] \in \{s[i], t[i]\}$. We claim that there exists some $i \in \{1, \dots, d\}$ such that $u[i] = t[i] \neq s[i]$. To see this, observe that there must exist i such that $u[i] \neq s[i]$, since otherwise $u = s$, and it then follows that $u[i] = t[i]$. Construct the relation $F_2(x_1, \dots, x_d) \equiv F_1(x_1, \dots, x_d) \wedge \{(u[1]), (t[1])\}(x_1) \wedge \dots \wedge \{(u[d]), (t[d])\}(x_d)$, and note that $F_2 \subset F_1$ since $s \notin F_2$. By repeating this procedure we will obtain a relation $F' \subseteq F$ such that $F' = \{s_0, s_1\}$ and such that $f(s_0) = 0$, $f(s_1) = 1$. Using F' we can then pp-define

$$R(x_{1,1}, \dots, x_{1,d}, \dots, x_{8,1}, \dots, x_{8,d}) \equiv f^{-1}(S_{\mathbb{B}})(x_{1,1}, \dots, x_{1,d}, \dots, x_{8,1}, \dots, x_{8,d}) \wedge F'(x_{1,1}, \dots, x_{1,d}) \wedge \dots \wedge F'(x_{8,1}, \dots, x_{8,d}).$$

Clearly, if $(a_{1,1}, \dots, a_{1,d}, \dots, a_{8,1}, \dots, a_{8,d}) \in R$, then $(a_{i,1}, \dots, a_{i,d}) \in \{s_0, s_1\}$ for each $1 \leq i \leq 8$, and $(f(a_{1,1}, \dots, a_{1,d}), \dots, f(a_{8,1}, \dots, a_{8,d})) \in S_{\mathbb{B}}$ if and only if $(a_{1,1}, \dots, a_{1,d}, \dots, a_{8,1}, \dots, a_{8,d}) \in f^{-1}(S_{\mathbb{B}})$. Since $R \subseteq f^{-1}(S_{\mathbb{B}})$, this implies that $(f(a_{1,1}, \dots, a_{1,d}), \dots, f(a_{8,1}, \dots, a_{8,d})) \in S_{\mathbb{B}}$ if and only if $(a_{1,1}, \dots, a_{1,d}, \dots, a_{8,1}, \dots, a_{8,d}) \in R$ and each $(a_{i,1}, \dots, a_{i,d}) \in \{s_0, s_1\}$. In other words each element $f(a_{i,1}, \dots, a_{i,d})$ in a tuple of $S_{\mathbb{B}}$ uniquely correponds to d arguments $a_{i,1}, \dots, a_{i,d}$ in the corresponding tuple of R, since $(a_{i,1}, \dots, a_{i,d}) = s_0$ if $f(a_{i,1}, \dots, a_{i,d}) = 0$, and $(a_{i,1}, \dots, a_{i,d}) = s_1$ if $f(a_{i,1}, \dots, a_{i,d}) = 1$. It follows that

$$R = \{s_0^\frown s_0^\frown s_1^\frown s_1^\frown s_1^\frown s_0^\frown s_0^\frown s_1, s_0^\frown s_1^\frown s_0^\frown s_1^\frown s_0^\frown s_1^\frown s_0^\frown s_1, s_1^\frown s_0^\frown s_0^\frown s_0^\frown s_1^\frown s_1^\frown s_0^\frown s_1\},$$

and therefore also that R is an $S_{\mathbb{B}}$-extension. ◀

Observe that the existence of an $S_{\mathbb{B}}$-extension $R \in \langle\Gamma\rangle$ does not imply that $\mathrm{CSP}(R) \leq^{\mathrm{CV}} \mathrm{CSP}(\Gamma)$. To accomplish this, we need to show that Γ can also qfpp-define an $S_{\mathbb{B}}$-extension. The proof is available in the extended preprint [28].

▶ **Lemma 14.** *Let Γ be an ultraconservative constraint language over a finite domain D such that $\mathrm{CSP}(\Gamma)$ is NP-complete. Then there exists a relation in $\langle\Gamma\rangle_{\exists}$ which is an $S_{\mathbb{B}}$-extension.*

4.2 Properties of and Reductions between $S_{\mathbb{B}}$-Extensions

By Lemma 14, we can completely concentrate on $S_{\mathbb{B}}$-extensions. We will prove that $\mathsf{T}(\{S_D\}) \leq \mathsf{T}(\Gamma)$ for every ultraconservative Γ over D such that $\mathrm{CSP}(\Gamma)$ is NP-complete. To prove this, we begin in Section 4.2.1 by investigating properties of $S_{\mathbb{B}}$-extensions, which we use to simplify the total number of distinct cases we need to consider. With the help of these results we in Section 4.2.2 develop techniques in order to show that $\mathrm{CSP}(S_D) \leq^{\mathrm{CV}} \mathrm{CSP}(R)$ for every $S_{\mathbb{B}}$-extension over D.

4.2.1 Saturated $S_{\mathbb{B}}$-Extensions

In this section we simplify the number of cases we need to consider in Section 4.2.2. First note that if $R = \{t_1, t_2, t_3\}$ over D is a relation with $\mathrm{ar}(R) > |D|^3$ then there exists i and j such that $(t_1[i], t_2[i], t_3[i]) = (t_1[j], t_2[j], t_3[j])$. We say that the jth argument is *redundant*, and it is possible to get rid of this by identifying the ith and jth argument with the qfpp-definition

$$R'(x_1, \dots, x_i, \dots, x_{j-1}, x_{j+1}, \dots, x_n) \equiv R(x_1, \dots, x_i, \dots, x_{j-1}, x_i, x_{j+1}, \dots, x_n).$$

This procedure can be repeated until no redundant arguments exist, and we will therefore always implicitly assume that $\mathrm{ar}(R) \leq |D|^3$ and that R has no redundant arguments. If R is an n-ary $S_{\mathbb{B}}$-extension then the argument $i \in \{1, \ldots, n\}$ is said to be *1-choice*, or *constant*, if $|\mathrm{Proj}_i(R)| = 1$, *2-choice* if $|\mathrm{Proj}_i(R)| = 2$, and *3-choice* if $|\mathrm{Proj}_i(R)| = 3$.

▶ **Definition 15.** An n-ary $S_{\mathbb{B}}$-extension $R = \{t_1, t_2, t_3\}$ is said to be *saturated* if there for each $1 \leq i \leq n$ and every function $\tau : \{1, 2, 3\} \to \{1, 2, 3\}$, exists $1 \leq j \leq n$ such that $(t_{\tau(1)}[i], t_{\tau(2)}[i], t_{\tau(3)}[i]) = (t_1[j], t_2[j], t_3[j])$.

▶ **Example 16.** The relation S_D is saturated for every D, but if we consider the relations R and R' defined by the matrices $\left(\begin{smallmatrix} 0&0&1&1&1&0&0&0&1&2 \\ 0&1&0&1&0&1&0&0&1&2 \\ 1&0&0&0&1&1&2&0&1&2 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 0&0&1&1&1&0&0&0&1&2 \\ 0&1&0&1&0&1&1&0&1&2 \\ 1&0&0&0&1&1&2&0&1&2 \end{smallmatrix}\right)$ then neither relation is saturated. First, R is not saturated since its matrix representation, for example, does not contain the column $(0, 2, 0)$. Second, R' is not saturated due to the 3-choice argument in position 7.

We will now see that we without loss of generality may assume that an $S_{\mathbb{B}}$-extension is saturated (see the extended preprint for proof [28]).

▶ **Lemma 17.** *Let R be an $S_{\mathbb{B}}$-extension. Then there exists a saturated $S_{\mathbb{B}}$-extension $R' \in \langle R \rangle_{\not\exists}$.*

▶ **Example 18.** If R is the relation from Example 16 then the saturated relation R' in $\langle R \rangle_{\not\exists}$ from Lemma 17 is given by $R' = \left(\begin{smallmatrix} 0&0&1&1&1&0&0&0&2&2&2&0&0&1&2 \\ 0&1&0&1&0&1&0&2&0&2&0&2&0&1&2 \\ 1&0&0&0&1&1&2&0&0&0&2&2&0&1&2 \end{smallmatrix}\right)$.

4.2.2 Reductions Between $S_{\mathbb{B}}$-Extensions

The main result of this section (Theorem 23 and Theorem 24) show that $\mathsf{T}(\{S_D\}) = \mathsf{T}(\{S_D\} \cup 2^D) \leq \mathsf{T}(\Gamma)$ whenever Γ is an ultraconservative constraint language over D such that $\mathrm{CSP}(\Gamma)$ is NP-complete. The result is proven by a series of CV-reductions that we present in Lemmas 19–22. Due to space constraints, we only present the proof of Lemma 20 which illustrates several useful techniques, and the remaining proofs can be found in the extended preprint [28]. Before we begin, we note that if R is an $S_{\mathbb{B}}$-extension over D then $\{R\}$ is not necessarily a core. For a simple counterexample, $\{S_{\mathbb{B}}\}$ is not a core over $\{0, 1, 2\}$ since the endomorphism $e(0) = 0$, $e(1) = 1$, $e(2) = 0$, is not an automorphism. However, if R is an $S_{\mathbb{B}}$-extension and $E = \{d_1, \ldots, d_m\}$ the set $\bigcup_{1 \leq i \leq \mathrm{ar}(R)} \mathrm{Proj}_i(R)$, every endomorphism $e : E \to E$ of R must be an automorphism. Hence, Theorem 7 is applicable, and we conclude that $\mathrm{CSP}(\{R, R^{d_1}, \ldots, R^{d_m}\}) \leq^{\mathrm{CV}} \mathrm{CSP}(R)$. When working with reductions between $S_{\mathbb{B}}$-extensions we may therefore freely make use of constant relations. Given an instance (V, C) of $\mathrm{CSP}(R)$, where R is an $S_{\mathbb{B}}$-extension, we say that a variable $x \in V$ occurring in a k-choice position in a constraint in C, $1 \leq k \leq 3$, is a *k-choice variable*.

▶ **Lemma 19.** *Let R be a saturated $S_{\mathbb{B}}$-extension. Then there exists a CV-reduction f from $\mathrm{CSP}(R)$ to $\mathrm{CSP}(R)$ such that for every instance I of $\mathrm{CSP}(R)$, each variable in $f(I)$ occurs as a 3-choice variable in at most one constraint.*

▶ **Lemma 20.** *Let R be a saturated $S_{\mathbb{B}}$-extension and let R' be R with one or more 3-choice arguments removed, such that R' is still saturated. Then $\mathrm{CSP}(R) \leq^{\mathrm{CV}} \mathrm{CSP}(R')$.*

Proof. Let $R = \{t_1, t_2, t_3\}$, $n = \mathrm{ar}(R)$, $n' = \mathrm{ar}(R')$, and assume that $\mathrm{Proj}_{1,\ldots,n'}(R) = R'$. Let $I = (V, C)$ be an instance of $\mathrm{CSP}(R)$. First apply Lemma 19 in order to obtain an instance $I_1 = (V_1, C_1)$ of $\mathrm{CSP}(R)$ such that each 3-choice variable only occurs in a 3-choice position in a single constraint. Assume there exists $x \in V_1$ and two distinct constraints $c, c' \in C_1$

such that x occurs in positions $i \in \{n'+1,\dots,n\}$ in c and in a 1- or 2-choice position $j \in \{1,\dots,n\}$ in c'. Let $S = \mathrm{Proj}_i(R) \cap \mathrm{Proj}_j(R)$, and note that $|S| \leq 2$. Assume first that $|S| = 2$, let $S = \{d_1, d_2\}$, and assume without loss of generality that $t_1[i] = t_1[j] = d_1$, $t_2[i] = t_2[j] = d_2$, and that $t_3[i] \neq t_3[j]$ (the other cases can be treated similarly). Since R is saturated there exists a 2-choice argument $i' \in \{1,\dots,n\}$ such that $t_1[i'] = t_1[i] = t_1[j]$, $t_2[i'] = t_2[i] = t_2[j]$, and such that $t_3[i'] \neq t_3[i]$. Let y be the variable occurring in the i'th position of c. Create a fresh variable $\hat{x}$, replace x in position i with $\hat{x}$, and for each constraint where x occurs as a 1- or 2-choice variable, replace x with y. Repeat this procedure until every 3-choice variable occurring in position $n'+1,\dots,n$ only occurs in a single constraint, and let $I_2 = (V_2, C_2)$ be the resulting instance. Assume there exists $x \in V_2$ and a constraint $c \in C_2$ such that x occurs as a 3-choice variable in position $i \in \{n'+1,\dots,n\}$ and also in a distinct position $j \in \{1,\dots,n\}$ in c. Let $L = \{t_r \mid 1 \leq r \leq 3, t_r[i] = t_r[j]\}$. Since R does not have any redundant arguments it must be the case that $|L| < 3$. If $|L| = 0$ then the instance is unsatisfiable, in which case we output an arbitrary unsatisfiable instance, and if $|L| = 1$ it is easy to see that any variable occurring in c can be assigned a fixed value, and the constraint may be removed. Therefore, assume that $|L| = 2$, and e.g. that $L = \{t_1, t_2\}$. Since R is saturated there exists a 2-choice argument $j' \in \{1,\dots,n\}$ such that $t_1[j'] = t_2[j'] \neq t_3[j']$. Let y be the variable occurring in position j' in c and add the constraint $R^{t_1[j']}(y)$. Repeat this for every variable occurring in position $n'+1,\dots,n$ in a constraint in C_2, and then replace each constraint $R(x_1,\dots,x'_n,\dots,x_n)$ by $R'(x_1,\dots,x_n)$. Note that any variable $\hat{x}$ introduced in the previous step of this reduction will be removed. Hence, the reduction is a CV-reduction. ◀

▶ **Lemma 21.** *Let R be an $S_{\mathbb{B}}$-extension and let R' be an $S_{\mathbb{B}}$-extension obtained by adding additional 2-choice arguments to R. Then* $\mathrm{CSP}(R') \leq^{\mathrm{CV}} \mathrm{CSP}(R)$.

▶ **Lemma 22.** *Let R be a saturated $S_{\mathbb{B}}$-extension over D with 3-choice arguments. Then* $\mathrm{CSP}(S_D) \leq^{\mathrm{CV}} \mathrm{CSP}(R)$.

We have thus proved the main result of this section.

▶ **Theorem 23.** *Let D be a finite domain and let Γ be a finite, ultraconservative constraint language over D. If* $\mathrm{CSP}(\Gamma)$ *is NP-complete then* $\mathsf{T}(\{S_D\}) \leq \mathsf{T}(\Gamma)$.

Proof. We first observe that if R is an $S_{\mathbb{B}}$-extension over a finite domain D, then $\mathrm{CSP}(S_D) \leq^{\mathrm{CV}} \mathrm{CSP}(R)$. By Lemma 17 we may assume that R is saturated. If R does not contain any 3-choice arguments we use Lemma 20 together with Lemma 21 and obtain a CV-reduction from $\mathrm{CSP}(S_D)$ to $\mathrm{CSP}(R)$. Hence, assume that R contains one or more 3-choice arguments. In this case we use Lemma 22 and obtain a CV-reduction from $\mathrm{CSP}(S_D)$ to $\mathrm{CSP}(R)$. By Lemma 14 there exists an $S_{\mathbb{B}}$-extension $R \in \langle\Gamma\rangle_{\not\exists}$, implying that $\mathrm{CSP}(R) \leq^{\mathrm{CV}} \mathrm{CSP}(\Gamma)$ via Theorem 6, and we know that $\mathrm{CSP}(S_D) \leq^{\mathrm{CV}} \mathrm{CSP}(R)$. We conclude that $\mathsf{T}(\{S_D\}) \leq \mathsf{T}(\{R\}) \leq \mathsf{T}(\Gamma)$. ◀

Clearly, $\{S_D\}$ is not an ultraconservative constraint language but the complexity of $\mathrm{CSP}(S_D)$ does not change when we expand the language by adding all unary relations over D (the proof can be found in the extended preprint [28]).

▶ **Theorem 24.** *Let D be a finite domain. Then* $\mathsf{T}(\{S_D\}) = \mathsf{T}(\{S_D\} \cup 2^D)$.

Thus, no NP-complete CSP over an ultraconservative constraint language over D is solvable strictly faster than $\mathrm{CSP}(S_D)$, and, in particular, $\mathsf{T}(\{S_{D'}\}) \leq \mathsf{T}(\{S_D\})$ whenever $D' \supseteq D$. This raises the question of whether $\mathsf{T}(S_D) = \mathsf{T}(S_{D'})$ for all $D, D' \supseteq \{0,1\}$, or if

it is possible to find D and D' such that $\mathsf{T}(\{S_{D'}\}) < \mathsf{T}(\{S_D\})$. As the following theorem shows, this is indeed the case, unless $\mathsf{T}(\{S_D\}) = 0$ for every finite D and the ETH fails.

▶ **Theorem 25.** $\inf\{\mathsf{T}(\{S_D\}) \mid D \textit{ finite and } |D| \geq 2\} = 0$.

Proof. Let $D_k = \{0, \ldots, k-1\}$, $k \geq 5$. We will analyse a simple algorithm for $\mathrm{CSP}(S_{D_k})$. Let $I = (V, C)$ be an arbitrary instance of $\mathrm{CSP}(S_{D_k})$. Extend the instance with variables $Z = \{z_0, \ldots, z_{k-1}\}$ and the constraints $R^i(z_i)$, $0 \leq i \leq k-1$. Arbitrarily choose a constraint $c = S_{D_k}(x_1, \ldots, x_{k^3})$ and let $X = \{x_1, \ldots, x_{k^3}\}$. It is straightforward to verify that if a variable x appears in k^2+1 or more positions, then c cannot be satisfied. Thus, $|X| \geq k$. If $X \cap Z = \emptyset$, then we branch on the three tuples in S_{D_k} and in each branch at least k variables in $V \setminus Z$ will be given fixed values. If a variable, say x_i, is given the fixed value d, then we identify x_i with z_d. Thus, at least k variables in $V \setminus Z$ are removed. Assume to the contrary that $X \cap Z \neq \emptyset$. If a variable $z \in Z$ occurs in a 3-choice position, then the variables in $X \setminus Z$ can be assigned fixed values and no branching is needed. If no variable $z \in Z$ occurs in a 3-choice position, then there are $k(k-1)(k-2)$ 3-choice positions in S_{D_k} and they are all covered by variables in $V \setminus Z$. Thus, we perform three branches based on the tuples in S_{D_k}. Recall that a variable can occur in at most k^2 positions in the constraint c since c is otherwise not satisfiable. This implies that at least $\lfloor \frac{k(k-1)(k-2)}{k^2} \rfloor \geq 1$ variables in $V \setminus Z$ are given fixed values (and are removed from $V \setminus Z$) in each branch. When there are no S_{D_k} constraints left, we check whether the remaining set of unary constraints are satisfiable or not. It is straightforward to perform this test in polynomial time. A recursive equation that gives an upper bound on the time complexity of this algorithm is thus $T(1) = poly(||I||), T(n) = 3T(n - \lfloor \frac{k(k-1)(k-2)}{k^2} \rfloor) + poly(||I||))$ (where n denotes the number of variables and $||I||$ the number of bits required to represent I) so $T(n) \in O(3^{n \cdot \frac{k^2}{k(k-1)(k-2)}} \cdot poly(||I||))$. The function $\frac{k^2}{k(k-1)(k-2)}$ obviously tends to 0 with increasing k so the infimum of the set $\{\mathsf{T}(\{S_D\}) \mid D \text{ is finite and } |D| \geq 2\}$ is equal to 0. ◀

5 Concluding Remarks and Future Research

In this paper we have studied the time complexity of NP-complete CSPs. Assuming the algebraic CSP dichotomy conjecture, we have ruled out subexponential time algorithms for NP-complete, finite-domain CSPs, unless the ETH is false. This proof also extends to degree-bounded CSPs and many classes of CSPs over infinite domains. We then proceeded to study the time complexity of CSPs over ultraconservative constraint languages, and proved that no such NP-complete CSP is solvable strictly faster than $\mathsf{T}(\{S_D\})$. These results raise several directions for future research.

First, Theorem 9 shows that the algebraic approach is viable for analysing the existence of subexponential algorithms for certain structurally restricted $\mathrm{CSP}(\Gamma)$ problems. An interesting continuation would be to determine which of the structurally (but not constraint language) restricted CSPs investigated by de Haan et al. [17] could be used to prove similar results. For example, is it the case that $\mathrm{CSP}(\Gamma)$ is not solvable in subexponential time whenever $\mathrm{CSP}(\Gamma)$ is NP-complete and the primal treewidth of an instance is bounded by $\Omega(n)$, unless the ETH fails?

Second, Several independent solutions to the algebraic CSP dichotomy conjecture have recently been announced [13, 30, 37]. If any of these proposed proofs is correct, it is tempting to extend Theorem 23 to constraint languages that are not necessarily ultraconservative or conservative. As a starting point, one could try to strengthen the results in Section 4.1, in order to prove that $\langle\Gamma\rangle_{\not\equiv}$ contains an $S_{\mathbb{B}}$-extension whenever $\mathrm{CSP}(\Gamma)$ is NP-complete and Γ is conservative (but not ultraconservative).

Acknowledgements. We thank Hannes Uppman for several helpful discussions on the topic of this paper.

References

1 L. Barto. Constraint satisfaction problem and universal algebra. *ACM SIGLOG News*, 1(2):14–24, October 2014.

2 L. Barto and M. 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 2016)*, pages 615–622, New York, NY, USA, 2016. ACM.

3 M. Behrisch, M. Hermann, S. Mengel, and G. Salzer. Give me another one! In *Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC-2015)*, pages 664–676, 2015.

4 M. Behrisch, M. Hermann, S. Mengel, and G. Salzer. As close as it gets. In *Proceedings of the 10th International Workshop on Algorithms and Computation (WALCOM-2016)*, pages 222–235, 2016.

5 M. Bodirsky. Complexity classification in infinite-domain constraint satisfaction. Mémoire d'habilitation à diriger des recherches, Université Diderot – Paris 7. Available at arXiv:1201.0856, 2012.

6 M. Bodirsky, P. Jonsson, and T. V. Pham. The complexity of phylogeny constraint satisfaction. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 20:1–20:13, 2016.

7 M. Bodirsky and J. Kára. The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, 57(2):9:1–9:41, 2010.

8 M. Bodirsky and M. Pinsker. Schaefer's theorem for graphs. *J. ACM*, 62(3):19:1–19:52, June 2015.

9 V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. I. *Cybernetics*, 5:243–252, 1969.

10 V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. II. *Cybernetics*, 5:531–539, 1969.

11 E. Böhler, E. Hemaspaandra, S. Reith, and H. Vollmer. Equivalence and isomorphism for boolean constraint satisfaction. In *In Proceedings of the 16th International Workshop on Computer Science Logic (CSL-2002)*, pages 412–426, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

12 A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Transactions on Computational Logic*, 12(4):24:1–24:66, July 2011.

13 A. Bulatov. A dichotomy theorem for nonuniform csps. *CoRR*, abs/1703.03021, 2017. URL: `http://arxiv.org/abs/1703.03021`.

14 A. Bulatov and A. Hedayaty. Counting problems and clones of functions. *Multiple-Valued Logic and Soft Computing*, 18(2):117–138, 2012.

15 A. Bulatov, P. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, March 2005. `doi:10.1137/S0097539700376676`.

16 N. Creignou, U. Egly, and J. Schmidt. Complexity classifications for logic-based argumentation. *ACM Transactions on Computational Logic (TOCL)*, 15(3):19:1–19:20, 2014.

17 R. de Haan, I. A. Kanj, and S. Szeider. On the subexponential-time complexity of CSP. *Journal of Artificial Intelligence Research (JAIR)*, 52:203–234, 2015. `doi:10.1613/jair.4540`.

18 T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.

19 D. Geiger. Closed systems of functions and predicates. *Pacific Journal of Mathematics*, 27(1):95–100, 1968.

20 M. Grohe. The structure of tractable constraint satisfaction problems. In *Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS 2006)*, pages 58–72, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

21 L. Ham. Gap theorems for robust satisfiability: Boolean CSPs and beyond. *To appear in Theoretical Computer Science*, 2017. doi:10.1016/j.tcs.2017.03.006.

22 T. Hertli. 3-SAT faster and simpler - unique-SAT bounds for PPSZ hold in general. *SIAM Journal on Computing*, 43(2):718–729, 2014. doi:10.1137/120868177.

23 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

24 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

25 P. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.

26 P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–548, July 1997. doi:10.1145/263867.263489.

27 P. Jonsson, V. Lagerkvist, G. Nordh, and B. Zanuttini. Strong partial clones and the time complexity of SAT problems. *Journal of Computer and System Sciences*, 84:52–78, 2017.

28 P. Jonsson, V. Lagerkvist, and B. Roy. Time Complexity of Constraint Satisfaction via Universal Algebra. *ArXiv e-prints*, June 2017. arXiv:1706.05902.

29 E. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, 5:1–122, 1941.

30 A. Rafiey, J. Kinne, and T. Feder. Dichotomy for digraph homomorphism problems. *CoRR*, abs/1701.02409, 2017. URL: http://arxiv.org/abs/1701.02409.

31 B.A. Romov. The algebras of partial functions and their invariants. *Cybernetics*, 17(2):157–167, 1981.

32 F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

33 S. J. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

34 T. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory Of Computing (STOC-78)*, pages 216–226. ACM Press, 1978.

35 H. Schnoor and I. Schnoor. Partial polymorphisms and constraint satisfaction problems. In N. Creignou, P. G. Kolaitis, and H. Vollmer, editors, *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 229–254. Springer Berlin Heidelberg, 2008.

36 M. Wahlström. *Algorithms, measures and upper bounds for satisfiability and related problems*. PhD thesis, Linköping University, TCSLAB - Theoretical Computer Science Laboratory, The Institute of Technology, 2007.

37 D. Zhuk. The proof of csp dichotomy conjecture. *CoRR*, abs/1704.01914, 2017. URL: https://arxiv.org/abs/1704.01914.

The Hardness of Solving Simple Word Equations

Joel D. Day[1], Florin Manea[2], and Dirk Nowotka[3]

1 **Kiel University, Department of Computer Science, Kiel, Germany**
jda@informatik.uni-kiel.de
2 **Kiel University, Department of Computer Science, Kiel, Germany**
flm@informatik.uni-kiel.de
3 **Kiel University, Department of Computer Science, Kiel, Germany**
dn@informatik.uni-kiel.de

Abstract

We investigate the class of regular-ordered word equations. In such equations, each variable occurs at most once in each side and the order of the variables occurring in both left and right hand sides is preserved (the variables can be, however, separated by potentially distinct constant factors). Surprisingly, we obtain that solving such simple equations, even when the sides contain exactly the same variables, is NP-hard. By considerations regarding the combinatorial structure of the minimal solutions of the more general quadratic equations we obtain that the satisfiability problem for regular-ordered equations is in NP. The complexity of solving such word equations under regular constraints is also settled. Finally, we show that a related class of simple word equations, that generalises one-variable equations, is in P.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.4.3 Formal Languages

Keywords and phrases Word Equations, Regular Patterns, Regular Constraints

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.18

1 Introduction

A *word equation* is an equality $\alpha = \beta$, where α and β are words over an alphabet $\Sigma \cup X$ (called the left, respectively, right side of the equation); $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots\}$ is the alphabet of *constants* and $X = \{x_1, x_2, x_3, \ldots\}$ is the alphabet set of *variables*. A *solution* to the equation $\alpha = \beta$ is a morphism $h : (\Sigma \cup X)^* \to \Sigma^*$ that acts as the identity on Σ and satisfies $h(\alpha) = h(\beta)$. For instance, $\alpha = x_1\mathtt{ab}x_2$ and $\beta = \mathtt{a}x_1x_2\mathtt{b}$ define the equation $x_1\mathtt{ab}x_2 = \mathtt{a}x_1x_2\mathtt{b}$, whose solutions are the morphisms h with $h(x_1) = \mathtt{a}^k$, for $k \geq 0$, and $h(x_2) = \mathtt{b}^\ell$, for $\ell \geq 0$.

The study of word equations (or the existential theory of equations over free monoids) is an important topic found at the intersection of algebra and computer science, with significant connections to, e.g., combinatorial group or monoid theory [19, 18, 2], unification [25, 11, 12], and, more recently, data base theory [9, 8]. The problem of deciding whether a given word equation $\alpha = \beta$ has a solution or not, known as the satisfiability problem, was shown to be decidable by Makanin [20] (see Chapter 12 of [17] for a survey). Later it was shown that the satisfiability problem is in PSPACE by Plandowski [22]; a new proof of this result was obtained in [14], based on a new simple technique called recompression. However, it is conjectured that the satisfiability problem is in NP; this would match the known lower bounds: the satisfiability of word equations is NP-hard, as it follows immediately from, e.g., [4]. This hardness result holds in fact for much simpler classes of word equations, like quadratic equations (where the number of occurrences of each variable in $\alpha\beta$ is at most two), as shown in [3]. There are also cases when the satisfiability problem is tractable. For

instance, word equations with only one variable can be solved in linear time in the size of the equation, see [13]; equations with two variables can be solved in time $\mathcal{O}(|\alpha\beta|^5)$, see [1].

In most cases, the NP-hardness of the satisfiability problem for classes of word equations was shown as following from the NP-completeness of the *matching problem* for corresponding classes of patterns with variables. In the matching problem we essentially have to decide whether an equation $\alpha = \beta$, with $\alpha \in (\Sigma \cup X)^*$ and $\beta \in \Sigma^*$, has a solution; that is, only one side of the equation, called pattern, contains variables. The aforementioned results [4, 3] show, in fact, that the matching problem is NP-complete for general α, respectively when α is quadratic. Many more tractability and intractability results concerning the matching problem are known (see [24, 6, 7]). In [5], efficient algorithms were defined for, among others, patterns which are *regular* (each variable has at most one occurrence), *non-cross* (between any two occurrences of a variable, no other distinct variable occurs), or patterns with only a constant number of variables occurring more than once.

Naturally, for a class of patterns that can be matched efficiently, the hardness of the satisfiability problem for word equations with sides in the respective class is no longer immediate. A study of such word equations was initiated in [21], where the following results were obtained. Firstly, the satisfiability problem for word equations with non-cross sides (for short non-cross equations) remains NP-hard. In particular, solving non-cross equations $\alpha = \beta$ where each variable occurs at most three times, at most twice in α and exactly once in β, is NP-hard. Secondly, the satisfiability of one-repeated variable equations (where at most one variable occurs more than once in $\alpha\beta$, but arbitrarily many other variables occur only once) having at least one non-repeated variable on each side, was shown to be trivially in P.

In this paper we mainly address the class of regular-ordered equations, whose sides are regular patterns and, moreover, the order of the variables occurring in both sides is the same. This seems to be one of the structurally simplest classes of equations whose number of variables is not bounded by a constant. One central motivation for studying these equations with a simple structure is that understanding their complexity and combinatorial properties may help us to identify a boundary between classes of word equations whose satisfiability is tractable and intractable. Moreover, we wish to gain a better understanding of the core reasons why solving word equations is hard. In the following, we overview our results, methods, and their connection to existing works from the literature.

Lower bounds. Our first result closes the main problem left open in [21]. Namely, we show that it is (still) NP-hard to solve regular (ordered) word equations. Note that in these word equations each variable occurs at most twice: at most once in every side. They are particular cases of both quadratic equations and non-cross equations, so the reductions showing the hardness of solving these more general equations do not carry over. To begin with, matching quadratic patterns is NP-hard, while matching regular patterns can be done in linear time. Showing the hardness of the matching problem for quadratic patterns in [3] relied on a simple reduction from 3-SAT, where the two occurrences of each variable were used to simulate an assignment of a corresponding variable in the SAT formula, respectively to ensure that this assignment satisfies the formula. To facilitate this final part, the second occurrences of the variables were grouped together, so the equation constructed in this reduction was not non-cross. Indeed, matching non-cross patterns can be done in polynomial time. So showing that solving non-cross equations is hard, in [21], required slightly different techniques. This time, the reduction was from an assignment problem in graphs. The (single) occurrences of the variables in one side of the equation were used to simulate an assignment in the graph, while the (two) occurrences of the variables from the other side were used for two reasons:

to ensure that the previously mentioned assignment is correctly constructed and to ensure that it also satisfies the requirements of the problem. For the second part it was also useful to allow the variables to occur in one side in a different order as in the other side.

As stated in [21], showing that the satisfiability problem for regular equations seems to require a totally different approach. Our hardness reduction relies on some novel ideas, and, unlike the aforementioned proofs, has a deep word-combinatorics core. As a first step, we define a reachability problem for a certain type of (regulated) string rewriting systems, and show it is NP-complete (in Lemma 7). This is achieved via a reduction from the strongly NP-complete problem 3-Partition [10]. Then we show that this reachability problem can be reduced to the satisfiability of regular-ordered word equations; in this reduction (described in the successive Lemmas 9, 10, and 11), we essentially try to encode the applications of the rewriting rules of the system into the periods of the words assigned to the variables in a solution to the equation. In doing this, we are able to only use one occurrence of each variable per side, and moreover to even have the variables in the same order in both sides.

Our reduction suggests the ability of this simple class of equations to model other natural problems in rewriting, combinatorics on words, and beyond. In this respect, our construction seems interesting with respect to the expressibility of word equations, as studied in [15].

Upper bounds. A consequence of the results in [23] is that the satisfiability problem for a certain class of word equations is in NP if the lengths of the minimal solutions of such equations (where the length of the solution defined by a morphism h is the image of the equation's sides under h) are at most exponential. With this in mind, we show Lemma 14, which gives us an insight in the combinatorial structure of the minimal solutions of quadratic equations: if we follow around the minimal solutions the positions that are fixed inside the images of the variables by each terminal of the original equation (in order, starting with that terminal), we obtain sequences that should not contain repetitions. Consequently, in Proposition 17, we give a simple and concise proof of the fact that the image of any variable in a minimal solution to a regular-ordered equation is at most linear in the size of the equation. It immediately follows that the satisfiability problem for regular-ordered equations is in NP. While this result was expected, the approach we use to obtain it seems rather interesting to us, and also a promising approach to showing that other, more complicated, classes of restricted word equations can be solved in NP-time. For instance, it is an open problem to show this for arbitrary regular or quadratic equations. It is worth noting that our polynomial upper bound on length of minimal solutions of regular-ordered equations does not hold even for slightly relaxed versions of such equations. More precisely, non-cross equations $\alpha = \beta$ where the order of the variables is the same in both sides and each variable occurs exactly three times in $\alpha\beta$, but never only on one side, may already have exponentially long minimal solutions (see Proposition 2). To this end, it seems even more surprising that it is NP-hard to solve equations with such a simple structure (regular-ordered), which, moreover, have quadratically short solutions. As such, regular-ordered equations seem to be among the structurally simplest word-equations, whose satisfiability problem is intractable.

Extending our ideas, we settle the complexity of solving regular-ordered equations with regular constraints (as defined in [3], where each variable is associated with an NFA), which is in NP for regular-ordered equations whose sides contain exactly the same variables, or when the languages defining the scope of the variables are all accepted by NFAs with at most c states, where c is a constant. For regular-ordered equations with regular constraints without these restrictions, the problem remains PSPACE-complete.

Finally, we use again a reasoning on the structure of the minimal solutions of equations, similar to the above, to show that if we preserve the non-cross structure of the sides of the

considered word equations, but allow only one variable to occur an arbitrary number of times (all the others occur exactly once in total, and hence in at most one side), we get a class of equations whose satisfiability problem is in P. This problem is related to the one-repeated variable equations considered in [21]; in this case, we restrict the equations to a non-cross structure of the sides, but drop the condition that at least one non-repeated variable should occur on each side. Moreover, this problem generalises the one-variable equations [13], while preserving the tractability of their satisfiability problem. Last, but not least, this result shows that the pattern searching problem, in which, given a pattern $\alpha \in (\Sigma \cup \{x_1\})^*$ containing constants and exactly one variable x_1 (occurring several times) and a text $\beta \in (\Sigma \cup \{x_1\})^*$ containing constants and the same single (repeated) variable, we check whether there exists an assignment of x_1 that makes α a factor of β, is tractable; indeed, this problem is the same as checking whether the word equation $x_2 \alpha x_3 = \beta$, with $\alpha, \beta \in (\Sigma \cup \{x_1\})^*$, is satisfiable.

2 Preliminaries

Let Σ be an *alphabet*. We denote by Σ^* the set of all words over Σ; by ε we denote the empty word. Let $|w|$ denote the *length* of a word w. For $1 \leq i \leq j \leq |w|$ we denote by $w[i]$ the letter on the i^{th} position of w and $w[i..j] = w[i]w[i+1]\cdots w[j]$. A word w is *p-periodic* for $p \in \mathbb{N}$ (and p is called a period of w) if $w[i] = w[i+p]$ for all $1 \leq i \leq |w| - p$; the smallest period of a word is called its *period*. By extension, for a word w of period p, we sometimes call $w[1..p]$ the period of w. Let $w = xyz$ for some words $x, y, z \in \Sigma^*$, then x is called *prefix* of w, y is a *factor* of w, and z is a *suffix* of w. Two words w and u are called *conjugate* if there exist non-empty words x, y such that $w = xy$ and $u = yx$. The powers of w are defined by $w^0 = \varepsilon$, $w^n = ww^{n-1}$ for $n \geq 1$, and $w^\omega = ww\cdots$, an infinite concatenation of the word w.

Let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots\}$ be an alphabet of *constants* and let $X = \{x_1, x_2, x_3, \ldots\}$ be an alphabet of *variables*. We assume that X and Σ are disjoint. A word $\alpha \in (\Sigma \cup X)^*$ is usually called *pattern*. For a pattern α and a letter $z \in \Sigma \cup X$, let $|\alpha|_z$ denote the number of occurrences of z in α; $\mathsf{var}(\alpha)$ denotes the set of variables from X occurring in α. A morphism $h : (\Sigma \cup X)^* \to \Sigma^*$ with $h(a) = a$ for every $a \in \Sigma$ is called a *substitution*. We say that $\alpha \in (\Sigma \cup X)^*$ is *regular* if, for every $x \in \mathsf{var}(\alpha)$, we have $|\alpha|_x = 1$; e. g., $\mathtt{a}x_1\mathtt{a}x_2\mathtt{c}x_3x_4\mathtt{b}$ is regular. Note that $L(\alpha) = \{h(\alpha) \mid h \text{ is a substitution}\}$ (the pattern language of α) is regular when α is regular, hence the name of such patterns. The pattern α is *non-cross* if between any two occurrences of the same variable x no other variable different from x occurs, e. g., $\mathtt{a}x_1\mathtt{ba}x_1x_2\mathtt{a}x_2x_2\mathtt{b}$ is non-cross, but $x_1\mathtt{b}x_2x_2\mathtt{b}x_1$ is not.

A *word equation* is a tuple $(\alpha, \beta) \in (\Sigma \cup X)^+ \times (\Sigma \cup X)^+$; we usually denote such an equation by $\alpha = \beta$, where α is the left-hand side (LHS, for short) and β the right-hand side (RHS) of the equation. A *solution* to an equation $\alpha = \beta$ is a substitution h with $h(\alpha) = h(\beta)$, and $h(\alpha)$ is called the *solution word* (*defined by* h); the length of a solution h of the equation $\alpha = \beta$ is $|h(\alpha)|$. A solution of shortest length to an equation is also called *minimal*.

A word equation is *satisfiable* if it has a solution and the *satisfiability problem* is to decide for a given word equation whether or not it is satisfiable. The satisfiability problem for general word equations can be solved non-deterministically in time polynomial in $n \log N$, where n is the length of the equation and N the length of its minimal solution [23]. The next result follows.

▶ **Lemma 1.** *Let $\mathcal{E}$ be a class of word equations. Suppose there exists a polynomial P such that for any equation in $\mathcal{E}$ its minimal solution, if it exists, has length at most $2^{P(n)}$ where n is the length of the equation. Then the satisfiability problem for $\mathcal{E}$ is in* NP.

A word equation $\alpha = \beta$ is *regular* or *non-cross*, if both α and β are regular or both α and β are non-cross, respectively; $\alpha = \beta$ is *quadratic* if each variable occurs at most twice in $\alpha\beta$. We call a regular or non-cross equation *ordered* if the order in which the variables occur in both sides of the equation is the same. That is, if x and y are variables occurring both in α and β, then all occurrences of x occur before all occurrences of y in α if and only if all occurrences of x occur before all occurrences of y in β. Note, however, that variables may occur only in one side of a regular or non-cross ordered equation. For instance $x_1x_1\mathtt{a}x_2x_3\mathtt{b}x_4 = x_1\mathtt{a}x_1x_2\mathtt{b}x_3$ is ordered non-cross, while $x_1x_1\mathtt{a}x_3x_2\mathtt{b} = x_1\mathtt{a}x_1x_2\mathtt{b}x_3$ is still non-cross but not ordered. Next we give an example of very simple word equations whose minimal solution has exponential length, whose structure follows that in [16, Theorem 4.8].

▶ **Proposition 2.** *The minimal solution to the word equation* $x_n\mathtt{a}x_n\mathtt{b}x_{n-1}\mathtt{b}x_{n-2}\cdots\mathtt{b}x_1 = \mathtt{a}x_nx_{n-1}^2\mathtt{b}x_{n-2}^2\mathtt{b}...\mathtt{b}x_1^2\mathtt{ba}^2$ *has length* $\Theta(2^n)$.

For a word equation $\alpha = \beta$ and an $x \in \mathsf{var}(\alpha\beta)$, a *regular constraint* (*for* x) is a regular language L_x. A solution h for $\alpha = \beta$ *satisfies* the regular constraint L_x if $h(x) \in L_x$. The satisfiability problem for word equations with regular constraints is to decide on whether an equation $\alpha = \beta$ with regular constraints L_x, $x \in \mathsf{var}(\alpha\beta)$, given as an NFA, has a solution that satisfies all regular constraints.

Finally, we recall the 3-Partition problem (see [10]). This problem is NP-complete in the strong sense, i.e., it remains NP-hard even when the input numbers are given in unary.

▶ **Problem 3** (3-Partition – 3-PAR).
Instance: $3m$ nonnegative integers (given in unary) $A = (k_1, \ldots, k_{3m})$, whose sum is ms
Question: Is there a partition of A into m disjoint groups of three elements, such that each group sums exactly to s.

3 Lower bounds

In this section, we show that the highly restricted class of regular-ordered word equations is NP-hard, and, thus, that even when the order in which the variables occur in an equation is fixed, and each variable may only repeat once – and never on the same side of the equation – satisfiability remains intractable. As mentioned in the introduction, our result shows the intractability of the satisfiability problem for a class of equations considerably simpler than the simplest intractable classes of equations known so far. Our result seems also particularly interesting since we are able to provide a corresponding upper bound in the next section, and even show that the minimal solutions of regular-ordered equations are "optimally short".

▶ **Theorem 4.** *The satisfiability problem for regular-ordered word equations is* NP*-hard.*

In order to show NP-hardness, we shall provide a reduction from a reachability problem for a simple type of regulated string-rewriting system. Essentially, given two words – a starting point, and a target – and an ordered series of n rewriting rules (a rewriting program, in a sense), the problem asks whether this series of rules may be applied consecutively (in the predefined order) to the starting word such that the result matches the target. We stress that the order of the rules is predefined, but the place where a rule is to be applied within the sentential form is non-deterministically chosen.

▶ **Problem 5** (Rewriting with Programmed Rules – REP).
Instance: Words $u_{start}, u_{end} \in \Sigma^*$ and an ordered series of n substitution rules $w_i \to w_i'$, with $w_i, w_i' \in \Sigma^*$, for $1 \leq i \leq n$.

Question: Can u_{end} be obtained from u_{start} by applying each rule (i.e., replacing an occurrence of w_i with w'_i), in order, to u_{start}.

▶ **Example 6.** Let $u_{start} = b^5$ and $u_{end} = (a^{11}bc^2)^5$; for $1 \leq i \leq 10$, consider the rules $w_i \to w'_i$ with $w_i = b$ and $w'_i = a^i bc$. We can obtain u_{end} from u_{start} by first applying $w_1 \to w'_1$ to the first b, then $w_2 \to w'_2$ to the second b, and further, in order for $3 \leq i \leq 5$, by applying $w_i \to w'_i$ to the i^{th} b. Then, we apply w_6 to the fifth b (counting from left to right). Further we apply in order, for $7 \leq i \leq 10$, $w_i \to w'_i$ to the $(11-i)^{th}$ occurrence of b.

It is not so hard to see that REP is NP-complete (the size of the input is the sum of the lengths of u_{start}, u_{end}, w_i and w'_i). A reduction can be given from 3-PAR, in a manner similar to the construction in the example above. Importantly for our proof, 3-PAR is strongly NP-complete, so it is simpler to reduce it to a problem whose input consists of words.

▶ **Lemma 7.** REP *is* NP*-complete.*

Our reduction centres on the construction, for any instance μ of REP, of a regular-ordered word equation $\alpha_\mu = \beta_\mu$ which possesses a specific form of solution – which we shall call *overlapping* – if and only if the instance of REP has a solution. By restricting the form of solutions in this way, the exposition of the rest of the reduction is simplified considerably. The main idea is that the applications of the rewriting rules are encoded in the periods of each variable in the solution.

▶ **Definition 8.** Let $n \in \mathbb{N}$, μ be an instance of REP with u_{start}, u_{end} and rules $w_i \to w'_i$ for $1 \leq i \leq n$. Let # be a 'new' letter not occurring in any component of REP. We define the regular-ordered equation $\alpha_\mu = \beta_\mu$ such that:
$\alpha_\mu := x_1\ w_1\ x_2\ w_2\ \cdots\ x_n\ w_n\ x_{n+1}\ \#\ u_{end}$, $\beta_\mu := \#\ u_{start}\ x_1\ w'_1\ x_2\ w'_2\ \cdots\ x_n\ w'_n\ x_{n+1}$.
A solution $h : (X \cup \Sigma)^* \to \Sigma^*$ is called *overlapping* if, for every $1 \leq i \leq n$, there exists $z_i \in \Sigma^*$ such that $w_i z_i$ is a suffix of $h(x_i)$ and $h(\#u_{start}x_1 \cdots w'_{i-1}x_i) = h(x_1 w_1 \cdots x_i w_i)z_i$.

Of course, satisfiability of a class of word equations asks whether any solution exists, rather than just overlapping solutions. Hence, before we prove our claim that $\alpha_\mu = \beta_\mu$ has an overlapping solution if and only if μ satisfies REP, we present a construction of an equation $\alpha = \beta$ which has a solution if and only if $\alpha_\mu = \beta_\mu$ has an overlapping solution. Essentially, this shows that solving the satisfiability of regular-ordered equations is as hard as solving the satisfiability of word equations when we restrict our search to overlapping solutions.

▶ **Lemma 9.** *Let μ be an instance of* REP*. There exists a regular-ordered equation $\alpha = \beta$ of size $O(|\alpha_\mu \beta_\mu|)$ such that $\alpha = \beta$ is satisfiable if and only if $\alpha_\mu = \beta_\mu$ has an overlapping solution.*

The rest of the proof relies on the following technical characterisation of overlapping solutions to $\alpha_\mu = \beta_\mu$ in terms of periods v_i of the images $h(x_i)$. The y_i factors will correspond to the z_i factors discussed in the definition of overlapping solutions (see also Figure 1).

▶ **Lemma 10.** *Let μ be a an instance of* REP *with u_{start}, u_{end} and rules $w_i \to w'_i$ for $1 \leq i \leq n$. A substitution $h : (X \cup \Sigma)^* \to \Sigma^*$ is an overlapping solution to $\alpha_\mu = \beta_\mu$ if and only if there exist prefixes $v_1, v_2, \ldots, v_n$ of $h(x_1), h(x_2), \ldots, h(x_n)$ such that:*
1. *$h(x_i)\ w_i$ is a prefix of v_i^ω for $1 \leq i \leq n$, and*
2. *$v_1 = \#u_{start}$, and for $2 \leq i \leq n$, $v_i = y_{i-1}w'_{i-1}$, and*
3. *$y_n\ w'_n\ h(x_{n+1}) = h(x_{n+1})\ \#u_{end}$,*
where for $1 \leq i \leq n$, y_i is the suffix of $h(x_i)$ of length $|v_i| - |w_i|$.

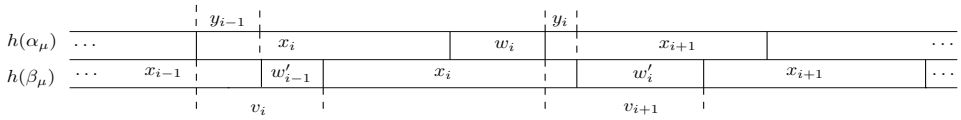


Figure 1 The period of $h(x_i)$ is v_i, and since w_iy_i is a suffix of $h(x_i)$ and $|w_iy_i| = |v_i|$, we have that w_iy_i is a cyclic shift of v_i (i.e., they are conjugate) – so $v_i = sw_it$ and $w_iy_i = w_its$ for some s, t. v_{i+1} is conjugate to sw'_it since $v_{i+1} = y_iw'_i = tsw'_i$. Thus v_{i+1} is obtained from v_i by "applying" the rule $w_i \to w'_i$ (and conjugating, but we can keep track of this due to the unique occurrence of #).

We shall now take advantage of Lemma 10 in order to demonstrate the correctness of our construction of $\alpha_\mu = \beta_\mu$ – i.e., that it has an overlapping solution if and only if μ satisfies REP. In particular, the periods v_i of the images $h(x_i)$ of the variables – which are obtained as the 'overlap' between the two occurrences of $h(x_i)$ – store the i^{th} stage of a rewriting $u_{start} \to \ldots \to u_{end}$. In fact, this is obtained as the conjugate of v_i starting with #. Thus the solution-word, when it exists, stores a sort-of rolling computation history.

▶ **Lemma 11.** *Let μ be a an instance of* REP *with u_{start}, u_{end} and rules $w_i \to w'_i$ for $1 \le i \le n$. There exists an overlapping solution $h : (X \cup \Sigma)^* \to \Sigma^*$ to the equation $\alpha_\mu = \beta_\mu$ if and only if μ satisfies* REP*.*

Proof. Suppose firstly that μ satisfies REP. Then there exist $s_1, s_2, \ldots, s_n, t_1, t_2, \ldots, t_n$ such that $u_{start} = s_1w_1t_1$, for $1 \le i \le n-1$, $s_iw'_it_i = s_{i+1}w_{i+1}t_{i+1}$ and $s_nw'_nt_n = u_{end}$. Let $h : (X \cup \Sigma)^* \to \Sigma^*$ be the substitution such that $h(x_1) = \#s_1w_1t_1\#s_1$, $h(x_{n+1}) = t_n\#s_nw'_nt_n\#s_nw'_nt_n$, and for $2 \le i \le n$, $h(x_i) = t_{i-1}\#s_{i-1}w'_{i-1}t_{i-1}\#s_i$. We shall now show that h satisfies Lemma 10, and hence that h is an overlapping solution to $\alpha_\mu = \beta_\mu$.

Let $v_1 = \#s_1w_1t_1$, let $y_1 := t_1\#s_1$, and for $2 \le i \le n$, let $v_i := t_{i-1}\#s_{i-1}w'_{i-1}$ and let $y_i := t_i\#s_i$. Note that for $1 \le i \le n$, v_i is a prefix of $h(x_i)$, and moreover, since $s_{i-1}w'_{i-1}t_{i-1} = s_iw_it_i$, y_i is the suffix of $h(x_i)$ of length $|v_i| - |w_i|$.

It is clear that h satisfies Condition (1) of Lemma 10 for $i = 1$. For $2 \le i \le n$, we have $h(x_i)w_it_i = t_{i-1}\#s_{i-1}w_{i-1}t_{i-1}\#s_iw_it_i = t_{i-1}\#s_{i-1}w_{i-1}t_{i-1}\#s_{i-1}w_{i-1}t_{i-1}$, which is a prefix of v_i^ω, and hence $h(x_i)w_i$ is also a prefix of v_i^ω. Thus h satisfies Condition (1) for all i. Moreover, $v_1 = \#u_{start}$, and for $2 \le i \le n$, $y_{i-1}w'_{i-1} = t_{i-1}\#s_{i-1}w'_{i-1} = v_i$, so h satisfies Condition (2). Finally, $y_nw'_nh(x_{n+1}) = t_n\#s_nw'_nt_n\#s_nw'_nt_n\#s_nw'_nt_n = h(x_{n+1})\#u_{end}$ so h also satisfies Condition (3).

Now suppose that h is an overlapping solution to $\alpha_\mu = \beta_\mu$. Then h satisfies Conditions (1), (2) and (3) of Lemma 10. Let v_i, y_i be defined according to the lemma for $1 \le i \le n$, and let $v_{n+1} = y_nw'_n$. We shall show that μ satisfies REP as follows. We begin with the following observations, whose proofs are omitted due to space constraints.

▶ **Claim 12.** *For $1 \le i \le n$, y_iw_i and v_i are conjugate. Hence, for $1 \le i \le n+1$, $|v_i|_\# = 1$.*

Let $\tilde{v}_i$ be the (unique) conjugate of v_i which has # as a prefix. Then we have the following:

▶ **Claim 13.** *For $1 \le i \le n$, there exist s_i, t_i such that $\tilde{v}_i = \#s_iw_it_i$ and $\tilde{v}_{i+1} = \#s_iw'_it_i$.*

Recall from Condition (3) of Lemma 10 that $v_{n+1}h(x_{n+1}) = y_nw'_nh(x_{n+1}) = h(x_{n+1})\#u_{end}$. Consequently, v_{n+1} and $\#u_{end}$ are conjugate, so $\tilde{v}_{n+1} = \#u_{end}$. Moreover, by Condition (2) of Lemma 10, $v_1 = \tilde{v}_1 = \#u_{start}$. Thus, it follows from Claim 13 that μ satisfies REP. ◀

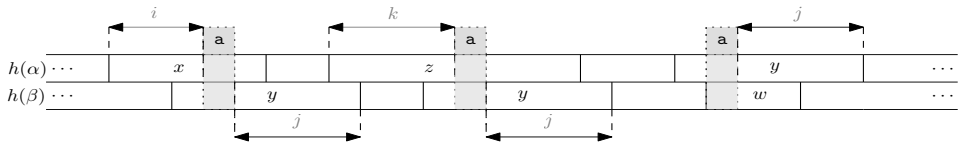


Figure 2 Fixing positions: since an occurrence of the i^{th} letter of $h(x)$ corresponds to an occurrence of the $(|h(y)| - j)^{th}$ letter of y, whose other occurrences correspond to the k^{th} letter of $h(z)$ and first letter of $h(w)$, all these positions are equivalent and contain the same letter, e.g., `a`.

It is clear that the equation $\alpha_\mu = \beta_\mu$ (and hence also the equation $\alpha = \beta$ given in Lemma 9) may be constructed in polynomial time, therefore our reduction from REP is complete. So, by Lemmas 7 and 11, we have shown Theorem 4.

4 NP-upper bound

In this section, we discuss a series of results related to the satisfiability of regular-ordered word equations. To this end, we extend the classical approach of filling the positions (see e.g., [15] and the references therein). This method essentially comprises of assuming that for a given equation $\alpha = \beta$, we have a solution h with specified lengths $|h(x)|$ for each variable x. The assumption that h satisfies the equation induces an equivalence relation on the positions of each $h(x)$: if a certain position in the solution-word is produced by an occurrence of the i^{th} letter of $h(x)$ on the RHS and an occurrence of the j^{th} letter of $h(y)$ on the LHS, then these two positions must obviously have the same value/letter and we shall say that these occurrences *correspond*. These individual equivalences can be combined to form equivalence classes, and if no contradictions occur (i.e., two different terminal symbols `a` and `b` do not belong to the same class), a valid solution can be derived.

Such an approach already allows for some straightforward observations regarding the (non-)minimality of a solution h. In particular, if an equivalence class of positions is not associated with any terminal symbol, then all positions in this class can be mapped to ε, resulting in a strictly shorter solution. On the other hand, even for our restricted setting, this observation is insufficient to provide a bound on the length of minimal solutions. In fact, in the construction of the equivalence classes we ignore, or at least hide, some of the structural information about the solution. In what follows, we shall see that by considering the exact 'order' in which positions are equated, we are able to give some more general conditions under which a solution is not minimal.

For our approach, rather than just constructing these equivalence classes, we shall construct sequences of equivalent positions, and then analyse "similar" sequences. For example, one occurrence of a position i in $h(x)$ might correspond to an occurrence of position j in $h(y)$, while another occurrence of position j in $h(y)$ might correspond to position k in $h(z)$, and so on, in which case we would consider the sequence: $\ldots \to (x, i) \to (y, j) \to (z, k) \to \ldots$.

The sequence terminates when either a variable which occurs only once or a terminal symbol is reached. For general equations, considering all such sequences leads naturally to a graph structure where the nodes are positions $(x, i) \in X \times \mathbb{N}$, and number of edges from each node is determined by the number of occurrences of the associated variable. Each connected component of such a graph corresponds to an equivalence class of positions as before. In the case of quadratic (and therefore also regular) equations, where each variable occurs at most twice, each 'node' (x, i) has at most two edges, and hence our graph is simply a set of disjoint chains, without any loops. As before, each chain (called in the following sequence) must be

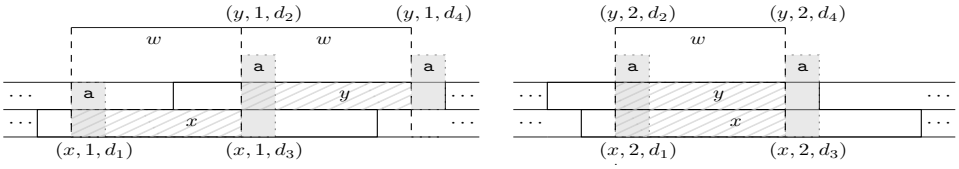


Figure 3 Illustration of Lemma 14 in the case of a short subsequence $\ldots,(x,1,d_1),(y,2,d_2),(x,1,d_3),(y,2,d_4),\ldots$: since the two sequences starting at $(x,1,d_1)$ and $(x,1,d_3)$ are similar, they define a common region w (shaded). Since they are consecutive, the first and last occurrences of w are adjacent, and on opposite sides of the equation. Thus, removing the region w from $h(x)$ and $h(y)$ does not alter the fact that h satisfies the equation.

associated with some occurrence of a terminal symbol, which must occur either at the start or the end. Hence we have $k < n$ sequences, where n is the length of the equation, such that every position (x,i), where x is a variable and $1 \leq i \leq |h(x)|$, occurs in exactly one sequence. It is also not hard to see that the total length of the sequences is upper bounded by $2|h(\alpha)|$.

In order to be fully precise, we will distinguish between different occurrences of a variable/terminal symbol by associating each with an index $z \in \mathbb{N}$ by enumerating occurrences from left to right in $\alpha\beta$. When considering quadratic equations, $z \in \{1,2\}$ for each variable x.

Formally, we define our sequences for a given solution h to a quadratic equation $\alpha = \beta$ as follows: a *position* is a tuple (x,z,d) such that x is a variable or terminal symbol occurring in $\alpha\beta$, $1 \leq z \leq |\alpha\beta|_x$, and $1 \leq d \leq |h(x)|$. Two positions (x,z,d) and (y,z',d') *correspond* if they generate the same position in the solution-word. The positions are *similar* if they belong to the same occurrence of the same variable (i.e., $x = y$ and $z = z'$). For each position p associated with either a terminal symbol or a variable occurring only once in $\alpha\beta$, we construct a sequence $S_p = p_1, p_2, \ldots$ such that

- $p_1 = p$ and p_2 is the (unique) position corresponding with p_1, and
- for $i \geq 2$, if $p_i = (x,z,d)$ such that x is a terminal symbol or occurs only once in $\alpha\beta$, then the sequence terminates, and
- for $i \geq 2$, if $p_i = (x,z,d)$, such that x is a variable occurring twice, then p_{i+1} is the position corresponding to the (unique) position (x,z',d) with $z' \neq z$ (i.e., the 'other' occurrence of the i^{th} letter in $h(x)$).

We extend the idea of similarity from positions to sequences of positions in the natural way: two sequences $p_1, p_2, \ldots, p_i$ and $q_1, q_2, \ldots, q_i$ are *similar* whenever p_j and q_j are similar for all $j \in \{1,2,\ldots,i\}$. Our main tool is the following lemma, which essentially shows that if a sequence contains two similar consecutive subsequences, then the solution defining that sequence is not minimal.

▶ **Lemma 14.** *Let h be a solution to a quadratic equation $\alpha = \beta$, and let p be a position associated with a single-occurring variable or terminal symbol. If the sequence S_p has a subsequence $p_1, p_2, \ldots, p_t, p_{t+1}, p_{t+2}, \ldots, p_{2t}$ such that $p_1, p_2, \ldots, p_t$ and $p_{t+1}, p_{t+2}, \ldots, p_{2t}$ are similar, then h is not minimal.*

Proof. Assume that S_p has such a subsequence and assume w.l.o.g. that it is length-minimal (so t is chosen to be as small as possible). For $1 \leq i \leq 2t$, let $p_i = (x_i, z_i, d_i)$ and note that by definition of similarity, for $1 \leq i \leq t$, $x_i = x_{i+t}$ and $z_i = z_{i+t}$. Assume that $d_1 < d_{t+1}$ (the case where $d_1 > d_{t+1}$ may be treated identically). We need the following two claims, whose proofs are omitted due to space constraints.

▶ **Claim 15.** *Suppose that* $(x, z, d), (x, z, d'), (x, z, d'')$ *are positions with* $d \leq d' < d''$ *such that* (x, z, d) *and* (x, z, d'') *correspond to* (y, z', e) *and* (y, z', e'') *respectively. Then* $e'' - e = d'' - d$*, and there exists* e' *with* $e' - e = d' - d$ *such that* (x, z, d') *and* (y, z', e') *correspond.*

A straightforward consequence of Claim 1 is that there exists a constant $C \in \mathbb{N}$ such that for all $i \in \{1, 2, \ldots, t\}$, $d_{i+t} - d_i = C$. Intuitively, each pair of similar positions $p_i = (x_i, z_i, d_i)$ and $p_{i+t} = (x_i, z_i, d_i + C)$ are the first and last positions of a factor $h(x_i)[d_i..d_i + C - 1]$, which as we shall see later on in the proof, can be removed to produce a shorter solution g.

It also follows from Claim 1, along with the fact that we chose t to be minimal, that there does not exist a position $p_i = (x_1, z_1, k)$ in the chain such that $d_1 < k < d_1 + C$. Symmetrically, there does not exist a position $p_i = (x_t, z_t, k)$ such that $d_t < k < d_t + C$:

▶ **Claim 16.** *Let* $j \in \{2, \ldots, t, t + 2, \ldots, 2t\}$ *such that* $x_j = x_1 (= x_{t+1})$ *and* $z_j = z_1 (= z_{t+1})$*. Then* $d_j \notin \{d_1, \ldots, d_{t+1} (= d_1 + C)\}$*. Likewise, if* $j \in \{1, \ldots, t - 1, t + 1, \ldots, 2t - 1\}$ *such that* $x_j = x_t$ *and* $z_j = z_t$*, then* $d_j \notin \{d_t, \ldots, d_{2t}\}$*.*

Using the observations above, we shall remove parts of the solution h to obtain a new, strictly shorter solution and thus show that h is not minimal as required.

To do this, we shall define a new equation $\alpha' = \beta'$ obtained by replacing the second occurrence of each variable x (when it exists) with a new variable x'. The reason is so we may delete factors independently from different occurrences of $h(x_i)$, and more easily keep track of the equivalence of the left and right hand side of the equation. Note that we can derive a solution h' to $\alpha' = \beta'$ from the solution h to our original equation by simply setting $h'(x) = h'(x') = h(x)$ for all $x \in \mathsf{var}(\alpha\beta)$. Likewise, any solution to $\alpha' = \beta'$ for which this condition holds (i.e., $h'(x) = h'(x')$ for all $x \in \mathsf{var}(\alpha\beta)$) induces a solution g to our original equation $\alpha = \beta$ given by $g(x) = h'(x) (= h'(x'))$. Finally, for each position (x, z, d) in the original solution h, there exists a unique "associated position" in h' given by $h(x)[d]$ if $z = 1$ and $h(x')[d]$ if $z = 2$. Furthermore, it follows from the definitions that for any pair of positions p, q which correspond (in terms of h), we can remove the associated positions from h' and the result will still be a valid solution to our modified equation $\alpha' = \beta'$ (although such a solution may no longer induce a valid solution to our original equation, since it is no longer necessarily the case where $h'(x) = h'(x')$ for all x).

We construct our shorter solution g to $\alpha = \beta$ as follows. Let h' be the solution to $\alpha' = \beta'$ derived directly from h. Recall from the definition of S_p that, for $1 \leq i < t$, the positions $(x_i, \overline{z}_i, d_i)$ and $(x_{i+1}, z_{i+1}, d_{i+1})$ correspond, where $\overline{z} = (z + 1) \mod 2$ (i.e., so that $\overline{z} \neq z$). Moreover, $(x_i, \overline{z}_i, d_i + C)$ and $(x_{i+1}, z_{i+1}, d_{i+1} + C)$ correspond, and thus by Claim 1, $(x_i, \overline{z}_i, d_i + k)$ and $(x_{i+1}, z_{i+1}, d_{i+1} + k)$ correspond for $0 \leq k \leq C - 1$. Since corresponding positions must have the same value/letter, it follows that there exists a factor $w \in \Sigma^+$ such that $w = h(x_i)[d_i..d_i + C - 1] (= h'(x)[d_i..d_i + C - 1] = h'(x')[d_i..d_i + C - 1])$ for $1 \leq i \leq t$.

We now produce a new solution, h'' to $\alpha' = \beta'$ as follows. For each i such that $1 \leq i \leq t-1$ and for each k such that $d_i \leq k \leq d_i + C - 1$, delete from h' the pair of positions associated with $(x_i, \overline{z}_i, d_i + k)$ and $(x_{i+1}, z_{i+1}, d_{i+1} + k)$. Note that since these positions correspond, in deleting them, we continue to have a valid solution to $\alpha' = \beta'$. Notice also that for every position associated with $(x_i, \overline{z}_i, d_i + k)$ such that $1 < i \leq t$, we also delete the position associated with $(x_i, z_i, d_i + k)$. Hence, for all $x \notin \{x_1, x_t\}$, $h''(x) = h''(x')$. The same does not necessarily hold for x_1 and x_t, however. In particular, we deleted positions associated with $(x_1, \overline{z_1}, d_1 + k)$ and $(x_t, z_t, d_t + k)$ for $0 \leq k \leq C - 1$, but not their counterparts $(x_1, z_1, d_1 + k)$ and $(x_t, \overline{z}_t, d_t + k)$. Hence, to derive a valid solution g, we must also delete these positions. To see that, in doing so, we still have a valid solution to $\alpha' = \beta'$, note firstly that, by Claim 2, we have not deleted any of these positions already. Moreover, it

follows from the sequence S_p that $(x_t, \overline{z}_t, d_t)$ corresponds to $(x_1, z_1, d_1 + C)$. Assume $z_1 = 1$ (the case $z_1 = 2$ is symmetric). It follows that $z_t = 1$ (since $\overline{z}_t \neq z_1$). Thus there exists an index m such that $h''(x_1)[d_1..d_1 + C - 1]$ generates the factor w starting at position m in $h''(\alpha')$ and $h''(x_t)[d_t..d_t + C - 1]$ generates the (same) factor w starting at position $m + |w|$ in $h''(\beta)$. It is straightforward to see that removing these factors (i.e., deleting the positions associated with $(x_1, z_1, d_1 + k)$ and $(x_t, \overline{z}_t, d_t + k)$ for $0 \leq k \leq C - 1$) does not affect the agreement of the two sides of the equation. Thus we obtain a shorter solution h'' to $\alpha' = \beta'$ such that $h(x) = h(x')$ for all variables x, hence a shorter solution g given by $g(x) = h''(x)$ to $\alpha = \beta$. ◀

An important fact related to the representation of the minimal solutions of quadratic equations as oriented sequences of positions, that start or end with a terminal symbol, is that the number of such sequences is linear in the size of the equation. Also, for variables that occur only once in a quadratic equation (so, only in one side), each of their positions is the start or end of a chain, so their length can be at most linear in the size of the equation. Thus, when interested in showing that a certain class of quadratic equation is in NP, we can assume that these equations do not contain variables occurring only once. Such variables could be non-deterministically replaced by words of at most linear size, and we would have to solve a new equation, whose size is at most quadratic in the size of the original one.

Further, using Lemma 14 and the observations above, we obtain immediately that minimal solutions to regular-ordered equations are at most linear in the length of the equation.

▶ **Proposition 17.** *Let E be a regular-ordered word equation with length n, and let h be a minimal solution to E. Then $|h(x)| < n$ for each variable x occurring in E.*

Proof. Every position of a minimal solution h to E occurs somewhere in one of the associated sequences S_p, and there are no more than n such sequences. So, it is sufficient to show that each one contains at most one position (x, z, d) for each variable x. This follows easily from the fact that E is regular and ordered and h is minimal, so it fulfils the restrictions of Lemma 14. ◀

We can see that, in terms of restricting the lengths of individual variables, the result in Proposition 17 is optimal. For instance, in a minimal solution h to the equation $w\mathsf{c}x_1 = x_1\mathsf{c}w$, with $w \in \{a, b\}^*$, the variable x_1 is mapped to w, so $|h(x)| = |E| - 2 \in O(|E|)$. Furthermore, Theorem 18 follows now as a direct consequence of Proposition 17 and Lemma 1, as the length of a minimal solution to a regular-ordered equation $\alpha = \beta$ is $O(|\alpha\beta|^2)$.

▶ **Theorem 18.** *The satisfiability problem for regular-ordered equations is in* NP.

It is a simple consequence of Proposition 17 that the satisfiability of a regular-ordered equation E with a constant number k of variables can be checked in P-time.

We now consider the satisfiability problem for regular equations with regular constraints, where we can make further use of the ideas presented so far in this section. Firstly, we observe that the problem is, in general, PSPACE-complete, even for regular-ordered equations (the proof given in [21] that the problem is PSPACE-complete for regular equations is also sufficient for the regular-ordered case).

▶ **Proposition 19.** *The satisfiability of regular-ordered equations with regular constraints is* PSPACE*-complete*

However, when considering regular equations whose sides contain exactly the same variables, and, moreover, in the same order, we get the following result, which shows an

important difference between the unrestricted regular-ordered equations, where the variables occurring only on one side do not affect the complexity of the satisfiability problem, and the same type of equations subject to regular constraints.

▶ **Theorem 20.** *The satisfiability of regular-ordered equations whose sides contain exactly the same variables, with regular constraints, is in* NP.

The same upper bound holds also for regular-ordered equations with regular constraints, when the regular constraints are regular languages accepted by NFAs with at most c states, where c is a constant (called constant regular constraints in the following).

▶ **Theorem 21.** *The satisfiability problem for regular-ordered equations with constant regular constraints is in* NP.

The proofs of the last two results rely also on the approach outlined in Lemma 14, but they are more involved. We first define the sequences of positions fixed by the terminals in a minimal solution of an equation. While this time the sequences may contain repetitions, we show that they cannot contain repetitions of non-constant exponent (where the constant, however, depends exponentially on c). This provides, again, a polynomial upper bound on the length of minimal solutions of such equations.

5 Tractable equations

Finally, we discuss a class of equations for which satisfiability is in P. Tractability was obtained so far from two sources: bound the number of variables by a constant (e.g., one or two-variable equations [13, 1]), or heavily restrict their structure (e.g., regular equations whose sides do not have common variable, or equations that only have one repeated variable, but at least one non-repeated variable on each side [21]). The class we consider slightly relaxes the previous restrictions. As the satisfiability of quadratic or even regular-ordered equations is NP-hard it seems reasonable to consider here patterns where the number of repeated variables is bounded by a constant (but may have an arbitrary number of non-repeated variables). More precisely, we consider here non-cross equations with only one repeated variable. This class generalises naturally the class of one-repeated variables.

▶ **Theorem 22.** *Let $x \in X$ be a variable and $\mathcal{D}$ be the class of word equations $\alpha = \beta$ such that $\alpha, \beta \in (\Sigma \cup X)^*$ are non-cross and each variable of X other than x occurs at most once in $\alpha\beta$. Then the satisfiability problem for $\mathcal{D}$ is in* P.

In the light of the results from [21], it follows that the interesting case of the above theorem is when the equation $\alpha = \beta$ is such that $\alpha = xu_1xu_2 \cdots u_kx$ and $\beta = \beta' v_0 x v_1 x v_2 \cdots x v_k \beta''$ where $v_0, v_1, \ldots, v_k, u_1, u_2, \ldots, u_k \in \Sigma^*$ and β', β'' are regular patterns that do not contain x and are variable disjoint. Essentially, this is a matching problem in which we try to align two non-cross patterns, one that only contains a repeated variable and constants, while the other contains the repeated variable, constants, and some wild-cards that can match any factor. As the proofs of [21] used essentially the non-repeated variables occurring in each of the sides, a novel and much more involved approach was needed here. The idea of our proof is to first show that such equations have minimal solutions of polynomial length. Further, we note that if we know the length of β' (w.r.t. the length of α) then we can determine the position where the factor $v_0xv_1xv_2 \cdots xv_k$ occurs in α, so the problem boils down to seeing how the positions of x are fixed by the constant factors v_i. Once this is done, we check if there exists an assignment of the variables of β' and β'' such that the constant factors of these patterns fit correctly to the corresponding prefix, respectively, suffix of α.

6 Conclusions and Prospects

The main result of this paper is the NP-completeness of the satisfiability problem for regular-ordered equations. While the lower bound seems remarkable to us because it shows that solving very simple equations, which also always have short solutions, is NP-hard, the upper bound seems more interesting from the point of view of the tools we developed to show it. We expect the combinatorial analysis of sequences of equivalent positions in a minimal solution to an equation (which culminated here in Lemma 14) can be applied to obtain upper bounds on the length of the minimal solutions to more general equations than just the regular-ordered ones. It would be interesting to see whether this type of reasoning leads to polynomial upper bounds on the length of minimal solutions to regular (not ordered) or quadratic equations, or to exponential upper bounds on the length of minimal solutions of non-cross or cubic equations. In the latter cases, a more general approach should be used, as the equivalent positions can no longer be represented as linear sequences, but rather as directed graphs.

Regarding the final section our paper, it seems interesting to us to see whether deciding the satisfiability of word equations with one repeated variable (so without the non-cross sides restriction) is still tractable. Also, it seems interesting to analyse the complexity of word equations where the number of repeated variables is bounded by a constant.

References

1. R. Dąbrowski and W. Plandowski. Solving two-variable word equations. In *Proc. 31th International Colloquium on Automata, Languages and Programming, ICALP 2004*, volume 3142 of *Lecture Notes in Computer Science*, pages 408–419, 2004.
2. V. Diekert, A. Jez, and M. Kufleitner. Solutions of word equations over partially commutative structures. In *Proc. 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 127:1–127:14, 2016.
3. V. Diekert and J. M. Robson. On quadratic word equations. In *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1999*, volume 1563 of *Lecture Notes in Computer Science*, pages 217–226, 1999.
4. A. Ehrenfeucht and G. Rozenberg. Finding a homomorphism between two words is NP-complete. *Information Processing Letters*, 9:86–88, 1979.
5. H. Fernau, F. Manea, R. Mercaş, and M.L. Schmid. Pattern matching with variables: Fast algorithms and new hardness results. In *Proc. 32nd Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 302–315, 2015.
6. H. Fernau and M. L. Schmid. Pattern matching with variables: A multivariate complexity analysis. *Information and Computation*, 242:287–305, 2015.
7. H. Fernau, M. L. Schmid, and Y. Villanger. On the parameterised complexity of string morphism problems. *Theory of Computing Systems*, 2015. http://dx.doi.org/10.1007/s00224-015-9635-3.
8. D. D. Freydenberger. A logic for document spanners. In *Proc. 20th International Conference on Database Theory, ICDT 2017*, Leibniz International Proceedings in Informatics (LIPIcs), 2017. To appear.
9. D. D. Freydenberger and M. Holldack. Document spanners: From expressive power to decision problems. In *Proc. 19th International Conference on Database Theory, ICDT 2016*, volume 48 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:17, 2016.

10 M. R. Garey and D. S. Johnson. *Computers And Intractability*. W. H. Freeman and Company, 1979.

11 J. Jaffar. Minimal and complete word unification. *Journal of the ACM*, 37(1):47–85, 1990.

12 A. Jez. Context unification is in PSPACE. In *Proc. 41st International Colloquium on Automata, Languages, and Programming, ICALP 2014*, volume 8573 of *Lecture Notes in Computer Science*, pages 244–255. Springer, 2014.

13 A. Jeż. One-variable word equations in linear time. *Algorithmica*, 74:1–48, 2016.

14 A. Jeż. Recompression: A simple and powerful technique for word equations. *Journal of the ACM*, 63, 2016.

15 J. Karhumäki, F. Mignosi, and W. Plandowski. The expressibility of languages and relations by word equations. *Journal of the ACM*, 47:483–505, 2000.

16 A. Koscielski and L. Pacholski. Complexity of Makanin's algorithm. *Journal of the ACM*, 43(4):670–684, 1996.

17 M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, Cambridge, New York, 2002.

18 R. C. Lyndon. Equations in free groups. *Transactions of the American Mathematical Society*, 96:445–457, 1960.

19 R. C. Lyndon and P. E. Schupp. *Combinatorial Group Theory*. Springer, 1977.

20 G. S. Makanin. The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik*, 103:147–236, 1977.

21 F. Manea, D. Nowotka, and M. L. Schmid. On the solvability problem for restricted classes of word equations. In *Proc. 20th International Conference on Developments in Language Theory, DLT 2016*, volume 9840 of *Lecture Notes in Computer Science*, pages 306–318. Springer, 2016.

22 W. Plandowski. An efficient algorithm for solving word equations. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC 2006*, pages 467–476, 2006.

23 W. Plandowski and W. Rytter. Application of Lempel-Ziv encodings to the solution of words equations. In *Proc. 25th International Colloquium on Automata, Languages and Programming, ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 731–742. Springer, 1998.

24 D. Reidenbach and M. L. Schmid. Patterns with bounded treewidth. *Information and Computation*, 239:87–99, 2014.

25 K. U. Schulz. Word unification and transformation of generalized equations. *Journal of Automated Reasoning*, 11:149–184, 1995.

Comparison of Max-Plus Automata and Joint Spectral Radius of Tropical Matrices

Laure Daviaud[*1], **Pierre Guillon**[2], **and Glenn Merlet**[3]

1 **MIMUW, University of Warsaw, Warsaw, Poland**
ldaviaud@mimuw.edu.pl
2 **Université d'Aix-Marseille CNRS, Centrale Marseille, I2M, UMR 7373, Marseille, France**
pierre.guillon@math.cnrs.fr
3 **Université d'Aix-Marseille CNRS, Centrale Marseille, I2M, UMR 7373, Marseille, France**
glenn.merlet@univ-amu.fr

Abstract

Weighted automata over the tropical semiring $\mathbb{Z}_{\max} = (\mathbb{Z} \cup \{-\infty\}, \max, +)$ are closely related to finitely generated semigroups of matrices over $\mathbb{Z}_{\max}$. In this paper, we use results in automata theory to study two quantities associated with sets of matrices: the joint spectral radius and the ultimate rank. We prove that these two quantities are not computable over the tropical semiring, *i.e.* there is no algorithm that takes as input a finite set of matrices Γ and provides as output the joint spectral radius (resp. the ultimate rank) of Γ. On the other hand, we prove that the joint spectral radius is nevertheless approximable and we exhibit restricted cases in which the joint spectral radius and the ultimate rank are computable. To reach this aim, we study the problem of comparing functions computed by weighted automata over the tropical semiring. This problem is known to be undecidable, and we prove that it remains undecidable in some specific subclasses of automata.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases max-plus automata, max-plus matrices, weighted automata, tropical semiring, joint spectral radius, ultimate rank

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.19

1 Introduction

Weighted automata were introduced by Schützenberger in [25] as a quantitative extension of nondeterministic finite automata. They compute functions from the set of words over a finite alphabet to the set of values of a semiring, allowing one to model quantities such as costs, gains or probabilities. In this paper, we particularly focus on max-plus automata: automata weighted within the tropical semiring $\mathbb{Z}_{\max} = (\mathbb{Z} \cup \{-\infty\}, \max, +)$. A max-plus automaton is thus a nondeterministic finite automaton whose transitions are weighted by integers. The value associated to a word w depends on the runs labelled by w: the weight of a given run is the sum of the weights of the transitions in the run, and the weight of w is the maximum of the weights of the accepting runs labelled by w. This kind of automata is

* The first author was partly supported by ANR Project ELICA ANR-14-CE25-0005, by ANR Project RECRE ANR-11-BS02-0010 and by project LIPA that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 683080).

particularly suitable to model gain maximisation, to study worst-case complexity [8] and to describe discrete event systems [12, 14]. The so-called linear presentation gives a matrix representation of such an automaton. More precisely, there is a canonical way to associate a max-plus automaton with a finitely generated semigroup of matrices over $\mathbb{Z}_{\max}$. Usually, the matrix representation is used to provide algebraic proofs of automata results. In this paper, we use results in automata theory to study two quantities related to sets of matrices: the joint spectral radius and the ultimate rank. The joint spectral radius generalises the notion of spectral radius for sets of matrices. The ultimate rank unifies some usual other notions of ranks. We link some comparison problems on max-plus automata with the computation of these two quantities. This leads to (1) proving results about comparison problems in some restricted classes of max-plus automata that we believe to be interesting for themselves and (2) applying these results to the study of the computability of the joint spectral radius and the ultimate rank.

Decidability questions about the description of functions computed by max-plus automata have been intensively studied. In his celebrated paper [20], Krob proves the undecidability of the equivalence problem for max-plus automata: there is no algorithm to decide if two max-plus automata compute the same function. In fact, his proof gives a stronger result: it is undecidable to determine whether a max-plus automaton computes a positive function. More recent approaches are based on a reduction from the halting problem of two-counter machines [1, 6]. By various reductions, this leads to the undecidability of several properties of automata with weights in different versions of the tropical semiring: $(\mathbb{N} \cup \{-\infty\}, \max, +)$, $(\mathbb{N} \cup \{+\infty\}, \min, +)$... The reader is referred to [21] for a survey on these questions.

By encoding the alphabet, and splitting each transition into several transitions with weight 1 and -1, it can be derived that the undecidability remains even if the automata are restricted to have weights within $\{-1, 1\}$ (Theorem 2). In [10], Gaubert and Katz notice that the undecidability of the comparison also remains true even if the number of states of the automata is bounded by a certain integer d. This extension is based on Krob's original proof and on the use of a universal diophantine equation. However, they ask for a more direct proof that would allow one to control the bound d. As an attempt to answer this question, we extend the proofs through two-counter machines. This allows a much sharper bound on the number of states for which comparison is undecidable (Theorem 2) than the one that would have followed from a universal diophantine equation.

The class of functions computed by max-plus automata that have all their states both initial and final is strictly included in the class of functions computed by max-plus automata. It is closely related to the study of finitely generated semigroup of tropical matrices. In this paper, we prove that comparison remains undecidable in this restricted class (Theorem 4). (In [1], it is proved that the undecidability remains for min-plus automata with all states final; we can deduce the same result for max-plus automata.)

The tropical (sub-)joint spectral radius is a natural counterpart of the usual joint spectral radius over the semiring $(\mathbb{R}, +, \times)$. Although the latter is a well-studied notion when considering the semiring $(\mathbb{R}, +, \times)$ (see [18] and the references therein) only few results are known when considering the tropical semiring. As far as we know, the best known result concerning its computability is given in [4], where it is shown that the joint spectral radius is NP-hard to compute and to approximate for tropical matrices. The tropical spectral radius as we defined it latter in this paper can be used to approximate the usual one in the spirit of [2] or [9], and from an applied point of view, for a max-plus linear system, it corresponds to the cycle time for an optimal scheduling of tasks, already studied in [12, 11, 14].

We drastically improve the NP-hardness result, by proving that the joint spectral radius is not computable in the tropical semiring, *i.e.* there is no algorithm that takes as input a

finite set Γ of matrices and provides as output the joint spectral radius of Γ (Theorem 8). As a corollary of this result, we also get the uncomputability of the ultimate rank, a notion introduced – and a question raised – in [15] (Theorem 9).

On the other hand, we also give positive results. By making a link with a result in [7] about approximate comparison of max-plus automata, we prove that the joint spectral radius is approximable in EXPSPACE (Theorem 10). We also show that, when restricted to matrices with only finite rational entries, computing the joint spectral radius or the ultimate rank is PSPACE-complete (Theorem 12).

2 Definitions and first properties

We introduce definitions and notation of tropical matrices and max-plus automata.

2.1 Tropical matrices

A *semigroup* $(S, \cdot)$ is a set S equipped with an associative binary operation '$\cdot$'. If, furthermore, the product has a neutral element 1, $(S, \cdot, 1)$ is called a *monoid*. The monoid is said to be *commutative* if $\cdot$ is commutative. A *semiring* $(S, \oplus, \otimes, 0_S, 1_S)$ is a set S equipped with two binary operations $\oplus$ and $\otimes$ such that $(S, \oplus, 0_S)$ is a commutative monoid, $(S - \{0_S\}, \otimes, 1_S)$ is a monoid, 0_S is absorbing for $\otimes$, and $\otimes$ distributes over $\oplus$. We shall use the *tropical semiring*:

$$\mathbb{Z}_{\max} = (\mathbb{Z} \cup \{-\infty\}, \max, +, -\infty, 0)$$

Note that $0_{\mathbb{Z}_{\max}} = -\infty$ and $1_{\mathbb{Z}_{\max}} = 0$. We may also use the restriction of $\mathbb{Z}_{\max}$ to the nonnegative integers, $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ denoted by $\mathbb{N}_{\max}$.

Semigroups of matrices. Let S be a semiring. The set of matrices with d rows and d' columns over S is denoted $\mathcal{M}^{d\times d'}(S)$, or simply $\mathcal{M}^d(S)$ if $d = d'$. The set of all matrices over S is $\mathcal{M}(S)$. As usual, the product AB for two matrices A, B (provided that the width of A and the height of B, denoted d, coincide) is defined as:

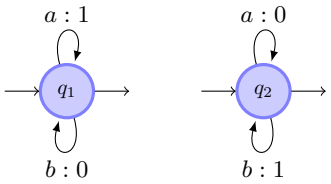
$$(AB)_{i,j} = \bigoplus_{1\leqslant k\leqslant d} (A_{i,k} \otimes B_{k,j}) \quad \left(= \max_{1\leqslant k\leqslant d} (A_{i,k} + B_{k,j}) \quad \text{for } S = \mathbb{Z}_{\max}\right)$$

The diagonal matrix with 1_S (*i.e.* 0 for $\mathbb{Z}_{\max}$) on the diagonal, and 0_S (*i.e.* $-\infty$ for $\mathbb{Z}_{\max}$) elsewhere is denoted I_d. It is a standard result that $(\mathcal{M}^d(S), \cdot, I_d)$ is a monoid.

For a positive integer k, we denote by M^k the product of M by itself k times. Moreover, $\|M\|_\infty$ denotes the maximal entry of a matrix M (it is not a norm). For $k \in \mathbb{Z}_{\max}$ and $A \in \mathcal{M}(\mathbb{Z}_{\max})$, $k \odot A$ is defined by $(k \odot A)_{ij} = k + A_{ij}$. For a set of matrices Γ, this notation is extended by $k \odot \Gamma = \{k \odot A | A \in \Gamma\}$. Finally, if $\Gamma \subset \mathcal{M}^d(S)$, we denote by $\langle\Gamma\rangle$ the submonoid generated by Γ.

Graph of a matrix. Any square matrix $A \in \mathcal{M}^d(\mathbb{Z}_{\max})$, for a positive integer d, can be represented by a graph $\mathcal{G}(A)$: the vertices are the indices $1, \ldots, d$, and there is an edge from i to j, labelled $A_{i,j}$, if and only if the latter is finite. The *spectral radius* $\rho(A)$ of a square matrix $A \in \mathcal{M}^d(\mathbb{Z}_{\max})$, for some positive integer d, known to be the limit $\lim_{n\to+\infty} \frac{1}{n} \|A^n\|_\infty$, can be seen as the maximal average weight of the cycles in $\mathcal{G}(A)$:

$$\rho(A) = \sup_{\substack{\ell\in\mathbb{N}\setminus\{0\} \\ 1\leqslant i_1,\ldots,i_\ell\leqslant d}} \left(\frac{1}{\ell}\left(A_{i_1,i_2} + A_{i_2,i_3} + \ldots + A_{i_{\ell-1},i_\ell} + A_{i_\ell,i_1}\right)\right)$$



$$\mu(a) = \begin{pmatrix} 1 & -\infty \\ -\infty & 0 \end{pmatrix} \qquad \mu(b) = \begin{pmatrix} 0 & -\infty \\ -\infty & 1 \end{pmatrix}$$

$$I = \begin{pmatrix} 0 & 0 \end{pmatrix} \qquad F = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Figure 1 Graph and matrix representations of a max-plus automaton.

The *critical graph* $\mathcal{G}_c(A)$ is the union of cycles $(i_1, \dots, i_\ell)$ that achieve this maximum. Its *strongly connected components* are the maximal subgraphs $C \subseteq \mathcal{G}_c(A)$ such that for any vertices i, j of C there is a path from i to j in $\mathcal{G}_c(A)$. The *cyclicity of a strongly connected component* is the greatest common divisor of the length of its cycles. The *cyclicity of* $\mathcal{G}_c(A)$ is the least common multiple of the cyclicities of its strongly connected components. The reader is referred to [3, 16, 5] for more detailed explanations.

2.2 Max-plus automata

We give the definition of max-plus automata that can be viewed as graphs or as sets of matrices. A *max-plus automaton* $\mathcal{A}$ over the alphabet Σ with d states is a map μ from Σ to $\mathcal{M}^d(\mathbb{Z}_{\max})$ together with an initial vector $I \in \mathcal{M}^{1\times d}(\{0, -\infty\})$ and a final vector $F \in \mathcal{M}^{d\times 1}(\{0, -\infty\})$[1]. The map μ is uniquely extended into a morphism, also denoted μ, from the semigroup Σ^+ of nonempty finite words over alphabet Σ into $\mathcal{M}^d(\mathbb{Z}_{\max})$. The *function computed by the automaton*, $[\![\mathcal{A}]\!]$, maps each word $w \in \Sigma^+$ to $I\mu(w)F \in \mathbb{Z}_{\max}$. Sometimes, 0 will denote the function constantly equal to 0, and $\geqslant$ the induced partial order over functions $\Sigma^+ \to \mathbb{Z}_{\max}$ (so that we can write things like $[\![\mathcal{A}]\!] \geqslant 0$).

Another way to represent a max-plus automaton is in terms of graphs. Given a map μ from Σ^+ to $\mathcal{M}^d(\mathbb{Z}_{\max})$, the corresponding automaton has d states $q_1, \dots q_d$, that correspond to the lines, or to the columns of the matrices. There is a transition from q_i to q_j labelled by a letter $a \in \Sigma$, with weight $\mu(a)_{i,j}$, if and only if the latter is finite. The initial (resp. final) states are the states q_i such that $I_i = 0$ (resp. $F_i = 0$). A run over the word w is a path (a sequence of compatible transitions) in the graph, labelled by w. Its weight is the sum of the weights of the transitions. The weight of a given word w is the maximum of the weights of the accepting runs (runs going from an initial state to a final state) labelled by w. The weight of w, given by the graph representation, is exactly the value $I\mu(w)F$, given by the matrix presentation.

Given a positive integer d and a max-plus automaton $\mathcal{A}$ defined by some map $\mu : \Sigma \to \mathcal{M}^d(\mathbb{Z}_{\max})$, we denote $\Gamma_{\mathcal{A}} = \{\, \mu(a) |\, a \in \Sigma\}$. Then the set of weights on the transitions of $\mathcal{A}$ corresponds to the finite entries appearing in matrices of $\Gamma_{\mathcal{A}}$.

▶ **Example 1.** Figure 1 gives the matrix and graph presentations of a max-plus automaton with 2 states, both initial (ingoing arrow) and final (outgoing arrow), over the alphabet $\{a, b\}$. The function that it computes associates a word w to the value $\max(|w|_a, |w|_b)$ where $|w|_x$ denotes the number of occurrences of the letter $x \in \{a, b\}$ in w.

This work aims to link results in automata theory with the study of semigroups of matrices. Concepts defined over semigroups of matrices correspond to concepts over the

[1] Note that, unlike variants in the literature, our weighted automata have no input or output weight (that is, I and F have entries in $\{0, -\infty\}$), but this does not restrict the set of computed functions.

subclass of automata in which all states are both initial and final, because if $M \in \mathcal{M}^d(\mathbb{Z}_{\max})$, and I and F have only 0 entries, then $IMF = \|M\|_\infty$, so that for this class of automata, $[\![\mathcal{A}]\!](w) = \|\mu(w)\|_\infty$ for every word w.

3 Undecidability of the comparison of max-plus automata

We are interested in the comparison problem, *i.e.* deciding, given two max-plus automata $\mathcal{A}$ and $\mathcal{B}$, whether $[\![\mathcal{A}]\!] \leqslant [\![\mathcal{B}]\!]$. There exist (at least) two different approaches to prove the undecidability of this problem. The original one by Krob [20] is a reduction from the tenth problem of Hilbert about diophantine equations. The proof is nicely written in [21], where it encodes a homogeneous polynomial P of degree 4 on n variables with integer coefficients into a max-plus automaton $\mathcal{A}$ computing a function with values in $\mathbb{N}$, such that $P-1$ has a root in $\mathbb{N}^n$ if and only if there is a word w such that $[\![\mathcal{A}]\!](w) = 0$, *i.e.* if $[\![\mathcal{A}]\!] \geq 1$. A more recent approach [1, 6], consists of a reduction from the halting problem of a two-counter machine. This computational model was introduced by Minsky [23, 24], and is as powerful as a Turing machine. It can be viewed as a finite-state machine with two counters. The equality of the counters with 0 can be tested and the counters can be incremented and decremented if not 0. The idea of the proof is to embed it into a max-plus automaton.

3.1 Restriction on the parameters

Different parameters can be taken into account when dealing with the size of a max-plus automaton: we will focus on the number of states, the maximal and minimal weights appearing on the transitions and the size of the alphabet. When considering the matrix representation of an automaton $\mathcal{A}$, these parameters correspond respectively to the dimension, the maximal and minimal finite entries and the number of matrices in $\Gamma_{\mathcal{A}}$.

Regarding the size of the alphabet, by a classical encoding from an arbitrary alphabet to a two-letter alphabet, one can see that the comparison problem remains undecidable when restricting to the class of automata on the binary alphabet.

Regarding the two other parameters, if they are both bounded, the problem becomes decidable since we are now only considering a finite number of max-plus automata. What is more interesting to study is when one of the parameter is bounded and not the other. We will see that this problem remains undecidable in these cases, and our purpose is to give bounds on these parameters that allow to keep the undecidability.

On the one hand, Gaubert and Katz notice in [10] that the original proof of Krob, applied to some specific diophantine equations gives that the problem remains undecidable when bounding the number of states. They also raised the question of finding an alternative proof that could allow to control this number of states. We roughly counted how many states we would obtain by using a so-called universal diophantine equation given in [17], of degree 4 with 58 unknowns. At the very least, we would be able to bound the number of states by 8700 on a 6-letter alphabet. On the other hand, the proof via two-counter machines allows to drastically improve this number, as we are going to see.

Define $\mathtt{Pos}^k(S)$ (resp. $\mathtt{Pos}^k_d(S)$) as the following problem: *Given a max-plus automaton $\mathcal{A}$ on a k-letter alphabet, with weights in $S \subseteq \mathbb{Z}_{\max}$ (resp. and d states), determine whether* $[\![\mathcal{A}]\!] \geqslant 0$.

▶ **Theorem 2.** *Problems $\mathtt{Pos}^2(\{-1,1\})$ and $\mathtt{Pos}^6_{553}(\mathbb{Z}_{\max})$ are undecidable.*

The first statement is easily derived from the general undecidability: given a max-plus automaton $\mathcal{A}$, one can first decompose all the transitions from $\mathcal{A}$ into a sequence of transitions

with weights 1 or -1 and 0 thanks to a suitable encoding of the alphabet, such that $\mathcal{A}$ computes a non negative function if and only if the newly created automaton computes a non negative function. Using then a usual encoding of any alphabet in a two-letter alphabet, we can obtain an automaton over two letters and with weights still in $\{-1, 0, 1\}$. Then, we transform again the automaton by multiplying all the weights by 2, so that a transition weighted by 1 (resp. -1) is now weighted by 2 (resp. -2). Finally, we decompose a transition labelled by a letter a with weight 2 (resp. -2) into two transitions each labelled by a with weight 1 (resp. -1), and a transition labelled by a with weight 0 into two transitions each labelled by a, one with weight 1 and the other with weight -1. For example, the word ab^2a is now encoded by $a^2b^4a^2$. All the words that are not composed with square of the letters (which form a rational language) are given weight $\geqslant 0$ (by using only weights in $\{1, -1\}$). It is easy to see that the obtained automaton computes a nonnegative function if and only if $[\![\mathcal{A}]\!] \geqslant 0$. Moreover the obtained automaton is on a two letter alphabet with weights within $\{-1, 1\}$.

The undecidability of $\mathtt{Pos}^6_{553}(\mathbb{Z}_{\max})$ is a contribution of the present paper. The halting problem is undecidable for a universal two-counter machine on any input (the initial value of the first counter). We construct a max-plus automaton simulating the runs of a two-counter machine where the input $n \in \mathbb{N}$ is now encoded by an additional widget involving two edges with weights n and $-n$. Thus we construct a max-plus automaton with a fixed number of states (depending only on the number of states of the universal two-counter machine) but arbitrary weights (induced by the value of the input of the universal two-counter machine).

3.2 Restriction on initial and final states

It is easy to see that max-plus automata having all their states initial and final compute only subadditive functions, that is to say functions f such that for any two words u and v, $f(uv) \leqslant f(u) + f(v)$. In particular, the support of such a function is closed by taking factors (if uvw is in the support then v is also in the support). Thus, as for unweighted automata [19], this subclass of automata defines a strict subclass of functions of the functions computed by max-plus automata. However, the following lemma shows that the comparison problem remains undecidable within this subclass.

▶ **Lemma 3.** *Let Σ be a finite alphabet and $\star \notin \Sigma$ be a special symbol. Given a max-plus automaton $\mathcal{A}$ on Σ, with d states and weights within a set S, one can build a max-plus automaton $\mathcal{A}'$ on $\Sigma' = \Sigma \cup \{\star\}$, with $d+1$ states, all of which are initial and final, and weights within $S \cup \{0\}$, such that:*

$$\min(\inf_{u\in\Sigma^+} \frac{[\![\mathcal{A}]\!](u)}{|u|}, 0) \leq \inf_{w\in\Sigma'^+} \frac{[\![\mathcal{A}']\!](w)}{|w|} \quad \text{and} \quad \inf_{u\in\Sigma'^+} \frac{[\![\mathcal{A}']\!](u)}{|u|} \leq \inf_{w\in\Sigma^+} \frac{[\![\mathcal{A}]\!](w)}{|w|+1}$$

In particular, $[\![\mathcal{A}]\!] \geqslant 0$ if and only if $[\![\mathcal{A}']\!] \geqslant 0$.

Proof. Consider a max-plus automaton $\mathcal{A}$ defined by a map $\mu : \Sigma \to \mathcal{M}^d(\mathbb{Z}_{\max})$, an initial vector I and a final vector F, and a new symbol $\star$. Let $\Sigma' = \Sigma \cup \{\star\}$. The idea is to construct a new automaton $\mathcal{A}'$ by adding a new state q and transitions from every final state of $\mathcal{A}$ to every initial state of $\mathcal{A}$ as well as transitions from every final state of $\mathcal{A}$ to q, loops around q and transitions from q to every initial state of $\mathcal{A}$, all labelled by $\star$ with weight 0. All the states of the new automaton $\mathcal{A}'$ are initial and final. Let us note μ', I' and F' defining this new automaton.

Any word $w \in \Sigma'^{+}\backslash\{\star\}^*$ can be written: $w = \star^{n_0} w_1 \star^{n_1} w_2 \star^{n_2} \dots w_k \star^{n_k}$ where for all $1 \leqslant i \leqslant k$, $w_i \in \Sigma^+$ and for all $0 < i < k$, $n_i > 0$, $n_0 \geqslant 0$ and $n_k \geqslant 0$. We get:

$$[\![\mathcal{A}']\!](w) = \|\mu'(w)\|_\infty \geq \sum_{i=1}^{k} I\mu(w_i)F$$

since the weight of $\star$ is 0. This is at least $\sum_{i=1}^{k} |w_i| \inf_u \frac{[\![\mathcal{A}]\!](u)}{|u|}$. If $[\![\mathcal{A}]\!] \geq 0$, then we get $[\![\mathcal{A}']\!](w) \geq 0$. Otherwise, $\inf_u \frac{[\![\mathcal{A}]\!](u)}{|u|} < 0$, and since $\sum_i |w_i| \leq |w|$, we get $\frac{[\![\mathcal{A}']\!](w)}{|w|} \geq \inf_u \frac{[\![\mathcal{A}]\!](u)}{|u|}$. Moreover, since the weights of the words in $\{\star\}^*$ is 0 in $\mathcal{A}'$, then the first inequality holds.

The other inequality is obtained by observing how arcs labelled $\star$ are positioned in $\mathcal{A}'$. Indeed, if a transition labelled by $\star$ is taken, then it has to start from a final state or q, and has to end in an initial state or q. Moreover, no other letter labels a transition starting or ending in q. So, when reading a word $w \in \Sigma^+$ between two $\star$, this word is read on a run that was already an existing accepting run in $\mathcal{A}$.

Thus, we see that for all words $w \in \Sigma^+$ and all $k \in \mathbb{N}$: $[\![\mathcal{A}']\!]((\star w)^k \star) = k\,[\![\mathcal{A}]\!](w)$, so that:

$$\inf_{u \in \Sigma'^+} \frac{[\![\mathcal{A}']\!](u)}{|u|} \leq \inf_{\substack{w \in \Sigma^+ \\ k \in \mathbb{N}}} \frac{[\![\mathcal{A}']\!]((\star w)^k \star)}{k(|w|+1)+1} \leq \inf_{\substack{w \in \Sigma^+ \\ k \in \mathbb{N}}} \frac{[\![\mathcal{A}]\!](w)}{|w|+1+1/k}\,. \qquad \blacktriangleleft$$

As a corollary of this lemma and of the previous results on the undecidability of comparison, we get the following theorem.

▶ **Theorem 4.** *The restrictions of Problems* $\texttt{Pos}^3(\{-1,0,1\})$ *and* $\texttt{Pos}^7_{554}(\mathbb{Z}_{\max})$ *to automata whose states are all initial and final are still undecidable.*

Simultaneously to the latter construction, one can also encode the alphabet into the binary alphabet in a smart way, yielding to the following result:

▶ **Theorem 5.** *The restrictions of Problems* $\texttt{Pos}^2(\{-1,0,1\})$ *and* $\texttt{Pos}^2_{3319}(\mathbb{Z}_{\max})$ *to automata whose states are all initial and final are still undecidable.*

4 Joint spectral radius and ultimate rank of tropical matrices

4.1 Joint spectral radius

The definition of spectral radius extends to the *joint spectral radius* of a set $\Gamma \subseteq \mathcal{M}^d(\mathbb{Z}_{\max})$ of matrices, as follows:

$$\rho(\Gamma) = \inf_{\ell > 0} \left\{ \frac{1}{\ell} \|M_1 \cdots M_\ell\|_\infty \,\middle|\, M_1, \dots, M_\ell \in \Gamma \right\}$$

The following lemma, which gives other equivalent definitions[2], is a known application of Fekete's subadditive lemma (see for example [11, Theorem 3.4]).

[2] Note that here we use the inf definition for the joint spectral radius instead of the sup definition used in the literature (see [18] and references therein). The latter is easy to compute in $\mathbb{Z}_{\max}$ since it is the spectral radius of the generators' tropical sum, unlike the notion considered here (sometimes called lower spectral radius or joint spectral subradius).

▶ **Lemma 6.** *For any set Γ of matrices in $\mathcal{M}^d(\mathbb{Z}_{\max})$, we have:*

$$\rho(\Gamma) = \lim_{\ell\to\infty} \min\left\{ \frac{1}{\ell} \left\|M_1 \cdots M_\ell\right\|_\infty \middle| M_1, \ldots, M_\ell \in \Gamma \right\} \tag{1}$$

$$= \inf_{\ell>0} \left\{ \frac{1}{\ell}\rho(M_1 \cdots M_\ell) \middle| M_1, \ldots, M_\ell \in \Gamma \right\} \tag{2}$$

Proof. Let $u_\ell = \inf\{ \left\|M_1 \cdots M_\ell\right\|_\infty \mid M_1, \ldots, M_\ell \in \Gamma\}$. The sequence $(u_\ell)_\ell$ is subadditive *i.e.* for all ℓ, ℓ', $u_{\ell+\ell'} \leqslant u_\ell + u_{\ell'}$. Indeed for all $M_1, \ldots, M_{\ell+\ell'} \in \Gamma$, $\left\|M_1 \cdots M_{\ell+\ell'}\right\|_\infty \leqslant \left\|M_1 \cdots M_\ell\right\|_\infty + \left\|M_{\ell+1} \cdots M_{\ell+\ell'}\right\|_\infty$. Thus by Fekete's lemma, $\lim_{\ell\to\infty} \frac{u_\ell}{\ell}$ is well defined and $\inf_{\ell>0} \frac{u_\ell}{\ell} = \lim_{\ell\to\infty} \frac{u_\ell}{\ell}$, which implies (1).

For the second equality, let us set: $\rho'(\Gamma) = \inf_{\ell>0} \left\{ \frac{1}{\ell}\rho(M_1 \cdots M_\ell) \middle| M_1, \ldots, M_\ell \in \Gamma \right\}$. Since for all matrices M, $\rho(M) \leqslant \left\|M\right\|_\infty$, we have $\rho'(\Gamma) \leqslant \rho(\Gamma)$. Let us show the reverse inequality. For all $\varepsilon > 0$, there is $\ell > 0$ and $M_1, \ldots, M_\ell \in \Gamma$ such that $\frac{1}{\ell}\rho(M_1 \cdots M_\ell) \leqslant \rho'(\Gamma) + \varepsilon$. By definition, it means that $\frac{1}{\ell} \lim_n \frac{1}{n} \left\|(M_1 \cdots M_\ell)^n\right\|_\infty \leqslant \rho'(\Gamma) + \varepsilon$, or equivalently, $\lim_n \frac{1}{n\ell} \left\|(M_1 \cdots M_\ell)^n\right\|_\infty \leqslant \rho'(\Gamma) + \varepsilon$. By definition, $\rho(\Gamma) \leqslant \lim_n \frac{1}{n\ell} \left\|(M_1 \cdots M_\ell)^n\right\|_\infty$, thus, for all $\varepsilon > 0$, $\rho(\Gamma) \leqslant \rho'(\Gamma) + \varepsilon$, that concludes the proof. ◀

It can be easily seen that $\rho(k \odot \Gamma) = \rho(\Gamma) + k$.

4.2 Ultimate rank

In the classical setting of a field, the notion of rank enjoys many equivalent definitions. These notions do not coincide in the case of $\mathbb{Z}_{\max}$. However, it was noticed in [15] that they coincide on the limit points of the powers of the matrix, when properly normalized (or considered projectively). This is formalized in [15, Theorem 5.2], and equivalent to the following definition: the *ultimate rank* $\operatorname{urk}(M)$ of a matrix $M \in \mathcal{M}^d(\mathbb{Z}_{\max})$ is the sum of the cyclicities of the strongly connected components of its critical graph. Clearly, $\operatorname{urk}(M) = 0$ (M has empty critical graph) if and only if $\rho(M) = -\infty$, and this corresponds to the nilpotency of M. As for the joint spectral radius, this notion can be generalized to sets of matrices. The *ultimate rank* of a set $\Gamma \subseteq \mathcal{M}^d(\mathbb{Z}_{\max})$ of matrices is: $\operatorname{urk}(\Gamma) = \min\{ \operatorname{urk}(M) \mid M \in \langle\Gamma\rangle\}$.

Clearly, $\operatorname{urk}(\Gamma) = 0$ if and only if $\rho(\Gamma) = -\infty$, and this corresponds to the mortality of the semigroup generated by Γ. It can be seen (or read in [15, Theorem 5.2]) that the ultimate rank is a projective notion: $\operatorname{urk}(k \odot \Gamma) = \operatorname{urk}(\Gamma)$ for any $k \in \mathbb{Z}$.

In some interesting cases, $\operatorname{urk}(\Gamma)$ is indeed the reached minimum of the ranks in the semigroup, so that it is the dimension of the limit set of the action of Γ on $\mathbb{R}^d$. Those cases include sets with irreducible fixed structure (all matrices have the same infinite entries), and sets of matrices with no line of $-\infty$ that contain one matrix with only finite entries. This is implicitely used in [22] and allows to extend some nice properties of products of random matrices from matrices with the so-called memory-loss property (case $\operatorname{urk}(\Gamma) = 1$) to more general ones (see [22, Corollary 1.2]).

4.3 Uncomputability and link with automata

Finitely generated semigroups of matrices exactly correspond to max-plus automata that have all their states initial and final. In particular, the following lemma links the computation of the joint spectral radius of the former to the comparison of the latter.

▶ **Lemma 7.** *Let $\mathcal{A}$ be a max-plus automaton over an alphabet Σ whose all states are both initial and final. The following statements are equivalent.*

1. $[\![\mathcal{A}]\!] \geqslant 0$.
2. *For all matrices M in $\langle\Gamma_{\mathcal{A}}\rangle$, $\left\|M\right\|_\infty \geqslant 0$.*

3. *For all matrices M in $\langle \Gamma_{\mathcal{A}} \rangle$, $\rho(M) \geqslant 0$.*
4. $\rho(\Gamma_{\mathcal{A}}) \geqslant 0$.

According to the terminology in [1], this also corresponds to the case when $\mathcal{A}$ is called *universal with threshold* 0.

Proof. Items 1. and 2. are equivalent since all the states of $\mathcal{A}$ are both initial and final. Thus, for all words w, $[\![\mathcal{A}]\!](w) = \|\mu(w)\|_\infty$. Moreover, $\langle \Gamma_{\mathcal{A}} \rangle$ is exactly the set $\{\mu(w) | w \in \Sigma^+\}$. Items 2. and 3. are equivalent by definition of the joint spectral radius. Finally, Items 3. and 4. are equivalent by Lemma 6. ◀

The uncomputability of the joint spectral radius is deduced from the equivalence in Lemma 7 and from Theorem 4 and 5. More precisely, define $\mathtt{JSR}^k(S)$ (resp. $\mathtt{JSR}^k_d(S)$) as the following problem: *Given a finite set of k matrices with coefficients in $S \subseteq \mathbb{Z}_{\max}$ (resp. and dimension d), determine whether their joint spectral radius is greater than or equal to* 0.

▶ **Theorem 8.** *Problems $\mathtt{JSR}^2(\{-\infty, -1, 0, 1\})$ and $\mathtt{JSR}^7_{554}(\mathbb{Z}_{\max})$ are undecidable.*

Proof. The undecidability comes from a reduction from the problem stated in Theorem 4 and 5. Consider a max-plus automaton $\mathcal{A}$ whose states are all initial and final. By Lemma 7, $[\![\mathcal{A}]\!] \geqslant 0$ if and only if $\rho(\Gamma_{\mathcal{A}}) \geqslant 0$. Thus $\mathtt{JSR}^2(\{-\infty, -1, 0, 1\})$ and $\mathtt{JSR}^7_{554}(\mathbb{Z}_{\max})$ are undecidable. ◀

By reduction from Theorem 8, we prove that the ultimate rank is also uncomputable. Define $\mathtt{UR}^k(S)$ (resp. $\mathtt{UR}^k_d(S)$) as the following problem: *Given a finite set of k matrices with coefficients in S (resp. and dimension d), determine whether the ultimate rank of the semigroup that they generate is equal to* 1.

▶ **Theorem 9.** *Problems $\mathtt{UR}^2(\{-\infty, -1, 0, 1\})$ and $\mathtt{UR}^7_{1109}(\mathbb{Z}_{\max})$ are undecidable.*

Proof. From any matrix M, one can build: $\widehat{M} = \begin{bmatrix} M & -\infty & -\infty \\ -\infty & M & -\infty \\ -\infty & -\infty & 0 \end{bmatrix}$. It is then clear that, for any finite family of matrices Γ, the semigroup generated by $\widehat{\Gamma} = \left\{ \widehat{M} \middle| M \in \Gamma \right\}$ is $\left\langle \widehat{\Gamma} \right\rangle = \left\{ \widehat{M} \middle| M \in \langle \Gamma \rangle \right\}$.

If M has size d and entries in S, then $\widehat{M}$ has size $2d + 1$ and entries in $S \cup \{-\infty, 0\}$. Moreover, if $\rho(M) < 0$, then the critical graph of $\widehat{M}$ is simply the loop over the last vertex (last line of the matrix $\widehat{M}$), so that $\mathrm{urk}(\widehat{M}) = 1$. Otherwise, the critical graph of $\widehat{M}$ contains at least two copies of that of M (which is nonempty), so that $\mathrm{urk}(\widehat{M}) \geq 2$. Thus, $\rho(M) \geqslant 0$ if and only if $\mathrm{urk}(\widehat{M}) \geq 2$. By reduction from the undecidable problems of Theorem 8, we can deduce that $\mathtt{UR}^2(\{-\infty, -1, 0, 1\})$ and $\mathtt{UR}^7_{1109}(\mathbb{Z}_{\max})$ are undecidable. ◀

▶ Remark. As noted above, the joint spectral radius and ultimate rank are not altered through translation by a constant; thus uncomputability is preserved with other restrictions over the entries. Regarding the joint spectral radius, the comparison to 0 may no longer be undecidable, but the comparison to some other constants will be. For instance, if $\Gamma \subset \mathcal{M}(\mathbb{N}_{\max})$, positivity is always true, but the question whether $\rho(\Gamma) \geqslant 1$ is undecidable.

4.4 Approximation of the joint spectral radius

Still by using results in automata theory, we prove that even though the joint spectral radius is not computable in general, it is approximable and computable in restricted cases in the following sense.

▶ **Theorem 10.** *There is an algorithm that, given a finite set* Γ *of matrices and* $n \in \mathbb{N} \setminus \{0\}$*, computes a value* $\alpha \in \mathbb{Q} \cup \{-\infty\}$ *such that* $\alpha - \frac{1}{n} \leqslant \rho(\Gamma) \leqslant \alpha + \frac{1}{n}$*.*

Proof. The proof uses the main result of [7]. This result is originally stated for min-plus automata using only positive weights. These automata are defined over the *min-plus semiring* $(\mathbb{Z} \cup \{+\infty\}, \min, +, +\infty, 0)$. By using the morphism from the min-plus to the max-plus semiring that associates k to $-k$, we can state the result of [7] in the max-plus case: there is an algorithm $\mathfrak{A}$ that, given a max-plus automaton $\mathcal{A}$ over an alphabet Σ using only nonpositive weights and $n \in \mathbb{N} \setminus \{0\}$, computes a value $\alpha \in \mathbb{Q} \cup \{+\infty\}$ such that: $\alpha - \frac{1}{n} \leqslant \inf_{w \in \Sigma^+} \frac{[\![\mathcal{A}]\!](w)}{|w|} \leqslant \alpha + \frac{1}{n}$.

Now, let us exhibit an algorithm that gives an approximation of the joint spectral radius of any finite set of matrices with only nonpositive entries. Consider a finite set of matrices Γ with only nonpositive entries, and a max-plus automaton $\mathcal{A}$ such that $\Gamma = \Gamma_{\mathcal{A}}$. From [7], $\mathfrak{A}$ also gives an approximation of the joint spectral radius of Γ, since we have:

$$\rho(\Gamma) = \inf_{\ell > 0} \left\{ \frac{1}{\ell} \left\| M_1 \cdots M_\ell \right\|_\infty \middle| M_1, \ldots, M_\ell \in \Gamma \right\} = \inf_{\ell > 0} \left\{ \frac{1}{\ell} [\![\mathcal{A}]\!](w) \middle| w \in \Sigma^\ell \right\}$$
$$= \inf_{w \in \Sigma^+} \frac{[\![\mathcal{A}]\!](w)}{|w|} .$$

Consider now a finite set of matrices Γ with arbitrary entries. Let k denote the greatest entry that appears in at least one of the matrices of Γ. Construct the set $\Gamma' = -k \odot \Gamma$. The set Γ' is then a finite set of matrices with only nonpositive entries, on which we can apply $\mathfrak{A}$. We then get an approximation of the joint spectral radius of Γ by adding k to the value given by the algorithm. ◀

This implies, in particular, that the joint spectral radius of every finite set of matrices is a computable real number.

Remarks about the complexity. The algorithm of [7] is EXPSPACE in the size of the automaton and in n. Moreover the problem is PSPACE-hard by reduction from the universality problem of a nondeterministic automaton: *Given a nondeterministic finite automaton* $\mathcal{A}$ *over a* 2*-letter alphabet* Σ*, the problem to determine whether the language accepted by* $\mathcal{A}$ *is* Σ^+ *is PSPACE-complete.* A precise statement of the reduction is given in the following lemma.

▶ **Lemma 11.** *Given a nondeterministic finite automaton* $\mathcal{A}$ *over a* 2*-letter alphabet* Σ*, one can construct in polynomial time a set of* 3 *matrices* Γ *with entries in* $\{-\infty, 0\}$ *such that* $\mathcal{A}$ *accepts* Σ^+ *if and only if the joint spectral radius of* Γ *is equal to* 0*. Otherwise, the joint spectral radius of* Γ *is equal to* $-\infty$*.*

Proof. Consider a nondeterministic finite automaton $\mathcal{A}$ over a 2-letter alphabet Σ. We construct a max-plus automaton $\mathcal{A}'$ from $\mathcal{A}$ by weighting the transitions by 0. Then, $\mathcal{A}$ accepts Σ^+ if and only if $[\![\mathcal{A}']\!] = 0$ (otherwise there is a word w such that $[\![\mathcal{A}']\!](w) = -\infty$). By Lemma 3, one can construct a max-plus automaton $\mathcal{B}$ over a 3-letter alphabet such that every state of $\mathcal{B}$ is both initial and final, $\mathcal{B}$ has only weight 0 on its transitions, and $[\![\mathcal{A}']\!] \geqslant 0$ if and only if $[\![\mathcal{B}]\!] \geqslant 0$. Hence, $[\![\mathcal{A}']\!] = 0$ if and only if $[\![\mathcal{B}]\!] = 0$. By Lemma 7, $[\![\mathcal{B}]\!] \geqslant 0$ if and

only if the joint spectral radius of $\Gamma_{\mathcal{B}}$ is nonnegative. Since, $\Gamma_{\mathcal{B}}$ contains only matrices with entries in $\{0, -\infty\}$, it implies that $[\![\mathcal{B}]\!] = 0$ if and only if the joint spectral radius of $\Gamma_{\mathcal{B}}$ is equal to 0. All the constructions are polynomial. ◀

Notice that Lemma 11 also proves that $\mathtt{JSR}^3(\{0, -\infty\})$ is PSPACE-hard. A result in [1] implies that $\mathtt{JSR}^k(\mathbb{Z}^- \cup \{-\infty\})$ is also PSPACE, where $\mathbb{Z}^-$ denotes the set of nonpositive integers. Hence, Problem $\mathtt{JSR}^3(\{0, -\infty\})$ is also PSPACE-complete.

4.5 Restriction to finite entries

Let us consider the restriction to matrices with only finite entries. In terms of automata, it means that for all letters a, there is a transition labelled by a between any pair of states. In this case, the joint spectral radius and ultimate rank are computable.

▶ **Theorem 12.** *There are PSPACE algorithms to compute the joint spectral radius and the ultimate rank of any finite set of matrices with finite entries. In particular, in this case, the joint spectral radius is a rational number. Moreover,* $\mathtt{JSR}^3(\{0, -1\})$ *and* $\mathtt{UR}^3(\{0, -1\})$ *are PSPACE-complete.*

PSPACE algorithm. To prove that the problems are PSPACE, the key point is the following lemma:

▶ **Lemma 13** ([13])**.** *Let* $\Gamma \subset \mathcal{M}^d(\{-b, \dots, b\})$ *for some nonnegative integers* b *and* d*. Then for all matrices* $M \in \langle\Gamma\rangle$ *and all indices* i, j*, the quantity* $M_{i,j} - M_{1,1}$ *belongs to* $\{-2b, \dots, 2b\}$*.*

Let Γ be a finite set of matrices with entries in $\{-b, \dots, b\}$. By Lemma 13, the set $\Lambda = \{-M_{1,1} \odot M \mid M \in \langle\Gamma\rangle\}$ contains at most $(4b+1)^{d^2-1}$ matrices.

Moreover, since the operation of adding the same constant to all the entries of a matrix commutes with the product of matrices, Λ is the set of matrices $-M_{1,1} \odot M$ such that M is a product of at most $(4b+1)^{d^2-1}$ matrices of Γ. Finally, the ultimate rank of Γ is the minimum of the ultimate rank of the matrices in Λ, which can be computed by the following algorithm in NPSPACE. Start with a matrix $M = M_1 \in \Gamma$ and a counter ℓ with value 1. At each (nondeterministic) step, either compute $\mathrm{urk}(M)$ and stop, or increase ℓ by one and multiply M by some matrix $M_\ell \in \Gamma$. If $\ell = (4b+1)^{d^2-1}$, then compute $\mathrm{urk}(M)$ and stop.

Since the maximum value of ℓ is simply exponential in the size $|\Gamma| d^2 \log(b)$ of the input, both ℓ and the size of the entries of $M = M_1 \cdots M_\ell$ are simply exponential and thus can be stored in polynomial space. Since the product of matrices and ultimate rank of one matrix can be computed in P the algorithm is in NPSPACE=PSPACE.

For the joint spectral radius, let us prove that

$$\rho(\Gamma) = \min_{\substack{\ell \leqslant (4b+1)^d \\ M_1, \dots, M_\ell \in \Gamma}} \left\{ \frac{1}{\ell} \rho(M_1 \cdots M_\ell) \right\} \tag{3}$$

and conclude in the same way. Let us consider a product $M_1 \cdots M_\ell$ of matrices in Γ and the orbit of the vector with all entries equal to 0 under the action of $M_1, M_2, \dots, M_\ell, M_1, M_2, \dots$. By Lemma 13, this orbit projectively has size at most $(4b+1)^d$. Hence, it cycles after t steps for some $t \leqslant (4b+1)^d$ and has a period $p \leqslant (4b+1)^d$. Each time the orbit goes back to the same vector projectively, all coordinates have increased by some value, which is the spectral radius of $M_{(t+1) \bmod \ell} M_{(t+2) \bmod \ell} \cdots M_{(t+p) \bmod \ell}$. Indeed, for matrices with only finite entries, the spectral radius is the only eigenvalue. Finally, we get $\frac{1}{\ell}\rho(M_1 \cdots M_\ell) = \frac{1}{p}\rho(M_{(t+1) \bmod \ell} M_{(t+2) \bmod \ell} \cdots M_{(t+p) \bmod \ell})$, which proves (3).

PSPACE-hardness. Let Γ be a finite set of matrices with entries in $\{0, -\infty\}$. Let Γ' be the set Γ where every entry with value $-\infty$ has been replaced by -1. The joint spectral radius of Γ is equal to 0 if and only if the joint spectral radius of Γ' is equal to 0. Otherwise, $\rho(\Gamma) = -\infty$ and $\rho(\Gamma')$ is strictly negative. By Lemma 11, $\mathtt{JSR}^3(\{0, -\infty\})$ is PSPACE-hard, and thus $\mathtt{JSR}^3(\{0, -1\})$ is PSPACE-hard.

Now, let us reduce $\mathtt{JSR}^3([\{0, -1\})$ to $\mathtt{UR}^3(\{0, -1\})$. From any matrix $M \in \mathcal{M}^d(\{0, -1\})$, with $d \in \mathbb{N} \setminus \{0\}$, one can build the matrix: $\widetilde{M} = \begin{bmatrix} M & (-1) \\ (-1) & 0 \end{bmatrix} \in \mathcal{M}^{d+1}(\{0, -1\})$, where (-1) is the vector with appropriate size whose entries are all -1.

It is then clear that, for any finite family of matrices Γ, the semigroup generated by $\widetilde{\Gamma} = \left\{ \widetilde{M} \middle| M \in \Gamma \right\}$ is $\left\langle \widetilde{\Gamma} \right\rangle = \left\{ \widetilde{M} \middle| M \in \langle \Gamma \rangle \right\}$.

Moreover, note that if $\rho(M) < 0$, then the critical graph of $\widetilde{M}$ is the loop over the last vertex, so that $\mathrm{urk}(\widetilde{M}) = 1$. Otherwise, $\rho(M) = 0$, and the critical graph is the union of this loop and the critical graph of M, so that $\mathrm{urk}(\widetilde{M}) = 1 + \mathrm{urk}(M)$. We deduce that the ultimate rank of $\widetilde{\Gamma}$ is greater than or equal to 2 if and only if $\rho(\Gamma) \geqslant 0$.

▶ Remark. Lemma 13 is implicitely used in [12, Corollary 2] to prove that the functions computed by max-plus automata with rational entries whose linear representation generates a so-called primitive semigroup (which includes matrices with finite entries) can be computed by a deterministic automaton.

It is also shown (as Corollary 4) that the minimal growth rate of a deterministic automaton, *i.e.* the joint spectral radius of its linear representation, can be computed as the spectral radius of one matrix whose indices are the states of the automaton.

This gives another algorithm to compute the joint spectral radius of a finite set of matrices with finite integers, but not a PSPACE one, since the size of the deterministic automaton is only bounded by $(4b + 1)^{d^2}$, while Equation (3) allows to compute only matrices of size d without storing them.

5 Conclusion and open questions

In this paper, we have proved that the joint spectral radius and the ultimate rank of a finite set of matrices over the tropical semiring are not computable (from the proof it can be seen that they are actually computably-enumerable-complete). To this end, we have proved the undecidability of the comparison of max-plus automata in restricted cases: when all the states are both initial and final and when the number of states is bounded.

As for the restriction on the number of states, we proved that comparison is undecidable when restricted to 553 states. Now, the question is to understand what happens between 2 and 552 states. Even when restricted to 2 states, it seems quite a difficult question to answer. Moreover, the various proofs highlight a link between several universal models: diophantine equations and two-counter machines. Having better size bounds on these models would give a better bound for our undecidable problem, but conversely, getting the decidability of comparison for max-plus automata with at most a certain number of states could lead to improve the known lower bounds on the size of these universal objects.

As for the joint spectral radius, one could ask if it is always rational or if, on the opposite, the set of joint spectral radii of finite families of matrices admits some computability-theoretic characterization. With respect to complexity, the main open question is whether it is PSPACE to approximate the joint spectral radius.

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In *ATVA 2011*, pages 482–491. Springer-Verlag, oct 2011.

2 Yu. A. Al'pin. Bounds for joint spectral radii of a set of nonnegative matrices. *Mathematical Notes*, 87(1):12–14, 2010. doi:10.1134/S0001434610010025.

3 François Louis Baccelli, Geert Jan Olsder, Jean-Pierre Quadrat, and Guy Cohen. *Synchronization and linearity. An algebra for discrete event systems.* Chichester: Wiley, 1992.

4 Vincent D. Blondel, Stéphane Gaubert, and John N. Tsitsiklis. Approximating the spectral radius of sets of matrices in the max-algebra is np-hard. *Automatic Control, IEEE Transactions on*, 45(9):1762–1765, Sep 2000. doi:10.1109/9.880644.

5 Peter Butkovič. *Max-linear systems. Theory and algorithms.* London: Springer, 2010. doi:10.1007/978-1-84996-299-5.

6 Thomas Colcombet. On distance automata and regular cost function. Presented at the Dagstuhl seminar "Advances and Applications of Automata on Words and Trees", 2010.

7 Thomas Colcombet and Laure Daviaud. Approximate comparison of distance automata. In Natacha Portier and Thomas Wilke, editors, *STACS*, volume 20 of *LIPIcs*, pages 574–585. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.574.

8 Thomas Colcombet, Laure Daviaud, and Florian Zuleger. Size-change abstraction and max-plus automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2014. doi:10.1007/978-3-662-44522-8_18.

9 Ludwig Elsner and P. van den Driessche. Bounds for the perron root using max eigenvalues. *Linear Algebra and its Applications*, 428(8):2000–2005, 2008. doi:10.1016/j.laa.2007.11.014.

10 Stéphane Gaubert and Ricardo Katz. Reachability problems for products of matrices in semirings. *International Journal of Algebra and Computation*, 16(3):603–627, jun 2006. URL: http://arxiv.org/abs/math/0310028, arXiv:0310028, doi:10.1142/S021819670600313X.

11 Stéphane Gaubert and Jean Mairesse. Task resource models and $(\max, +)$ automata. In *Idempotency (Bristol, 1994)*, volume 11 of *Publ. Newton Inst.*, pages 133–144. Cambridge Univ. Press, Cambridge, 1998. doi:10.1017/CBO9780511662508.009.

12 Stéphane Gaubert. Performance evaluation of $(\max, +)$ automata. *IEEE Trans. Automat. Control*, 40(12):2014–2025, 1995. doi:10.1109/9.478227.

13 Stéphane Gaubert. On the Burnside problem for semigroups of matrices in the (max,+) algebra. *Semigroup Forum*, 52(1):271–294, 1996. doi:10.1007/BF02574104.

14 Stéphane Gaubert and Jean Mairesse. Modeling and analysis of timed Petri nets using heaps of pieces. *IEEE Trans. Automat. Control*, 44(4):683–697, 1999. doi:10.1109/9.754807.

15 Pierre Guillon, Zur Izhakian, Jean Mairesse, and Glenn Merlet. The ultimate rank of semi-groups of tropical matrices. *Journal of Algebra*, 437:222–248, September 2015. doi:10.1016/j.jalgebra.2015.02.026.

16 Bernd Heidergott, Geert Jan Oldser, and Jacob van der Woude. *Max plus at work. Modeling and analysis of synchronized systems: a course on max-plus algebra and its applications.* Princeton, NJ: Princeton University Press, 2006.

17 James P. Jones. Universal Diophantine equation. *J. Symbolic Logic*, 47(3):549–571, 1982. doi:10.2307/2273588.

18 Raphaël Jungers. *The joint spectral radius*, volume 385 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag, Berlin, 2009. Theory and applications. doi:10.1007/978-3-540-95980-9.

19 Jui-Yi Kao, Narad Rampersad, and Jeffrey Shallit. On NFAs where all states are final, initial, or both. *Theoretical Computer Science*, 410(47):5010–5021, 2009. doi:10.1016/j.tcs.2009.07.049.

20 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In *Automata, languages and programming (Vienna, 1992)*, volume 623 of *Lecture Notes in Comput. Sci.*, pages 101–112. Springer, Berlin, 1992. doi:10.1007/3-540-55719-9_67.

21 Sylvain Lombardy and Jean Mairesse. Max-plus automaton. In *Handbook of Automata*. European Mathematical Society, To appear.

22 Glenn Merlet. Semigroup of matrices acting on the max-plus projective space. *Linear Algebra and its Applications*, 432(8):1923–1935, 2010. doi:10.1016/j.laa.2009.03.029.

23 Marvin L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Ann. of Math. (2)*, 74:437–455, 1961.

24 Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967. Prentice-Hall Series in Automatic Computation.

25 Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

Binary Search in Graphs Revisited*

Argyrios Deligkas[1], George B. Mertzios[2], and Paul G. Spirakis[3]

1 **Faculty of Industrial Engineering and Management, Technion, Israel**
argyris@technion.ac.il
2 **School of Engineering and Computing Sciences, Durham University, UK**
george.mertzios@durham.ac.uk
3 **Department of Computer Science, University of Liverpool, UK, University of Patras, Greece, and CTI, Greece**
p.spirakis@liverpool.ac.uk

Abstract

In the classical binary search in a path the aim is to detect an unknown target by asking as few queries as possible, where each query reveals the direction to the target. This binary search algorithm has been recently extended by [Emamjomeh-Zadeh et al., *STOC*, 2016] to the problem of detecting a target in an arbitrary graph. Similarly to the classical case in the path, the algorithm of Emamjomeh-Zadeh et al. maintains a candidates' set for the target, while each query asks an appropriately chosen vertex – the "median" – which minimises a potential Φ among the vertices of the candidates' set. In this paper we address three open questions posed by Emamjomeh-Zadeh et al., namely (a) detecting a target when the query response is a direction to an *approximately shortest path* to the target, (b) detecting a target when querying a vertex that is an *approximate median* of the current candidates' set (instead of an exact one), and (c) detecting *multiple targets*, for which to the best of our knowledge no progress has been made so far. We resolve questions (a) and (b) by providing appropriate upper and lower bounds, as well as a new potential Γ that guarantees efficient target detection even by querying an approximate median each time. With respect to (c), we initiate a systematic study for detecting two targets in graphs and we identify sufficient conditions on the queries that allow for strong (linear) lower bounds and strong (polylogarithmic) upper bounds for the number of queries. All of our positive results can be derived using our new potential Γ that allows querying approximate medians.

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

Keywords and phrases binary search, graph, approximate query, probabilistic algorithm, lower bound

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.20

1 Introduction

The classical binary search algorithm detects an unknown target (or "treasure") t on a path with n vertices by asking at most $\log n$ queries to an oracle which always returns the direction from the queried vertex to t. To achieve this upper bound on the number of queries, the algorithm maintains a set of candidates for the place of t; this set is always a sub-path, and initially it is the whole path. Then, at every iteration, the algorithm queries the middle vertex ("median") of this candidates' set and, using the response of the query, it excludes

* Partially supported by ISF 2021296, EPSRC grant EP/P020372/1, EPSRC grant EP/P02002X/1, and Liverpool EEECS initiative NeST

either the left or the right half of the set. This way of searching for a target in a path can be naturally extended to the case where t lies on an n-vertex tree, again by asking at most $\log n$ queries that reveal the direction in the (unique) path to t [22]. The principle of the binary search algorithm on trees is based on the same idea as in the case of a path: for every tree there exists a separator vertex such that each of its subtrees contains at most half of the vertices of the tree [14], which can be also efficiently computed.

Due to its prevalent nature in numerous applications, the problem of detecting an unknown target in an arbitrary graph or, more generally in a search space, has attracted many research attempts from different viewpoints. Only recently the binary search algorithm with $\log n$ direction queries has been extended to arbitrary graphs by Emamjomeh-Zadeh et al. [10]. In this case there may exist multiple paths, or even multiple shortest paths form the queried vertex to t. The direction query considered in [10] either returns that the queried vertex q is the sought target t, or it returns an arbitrary direction from q to t, i.e. an arbitrary edge incident to q which lies on a shortest path from q to t. The main idea of this algorithm follows again the same principle as for paths and trees: it always queries a vertex that is the "median" of the current candidates' set and any response to the query is enough to shrink the size of the candidates' set by a factor of at least 2. Defining what the "median" is in the case of general graphs now becomes more tricky: Emamjomeh-Zadeh et al. [10] define the median of a set S as the vertex q that minimizes a potential function Φ, namely the sum of the distances from q to all vertices of S.

Apart from searching for upper bounds on the number of queries needed to detect a target t in graphs, another point of interest is to derive algorithms which, given a graph G, compute the *optimal* number of queries needed to detect an unknown target in G (in the worst case). This line of research was initiated in [18] where the authors studied directed acyclic graphs (DAGs). Although computing a query-optimal algorithm is known to be NP-hard on general graphs [4,8,16], there exist efficient algorithms for trees; after a sequence of papers [1, 13, 17, 19, 26], linear time algorithms were found in [19, 22]. Different models with queries of non-uniform costs or with a probability distribution over the target locations were studied in [5–7, 15].

A different line of research is to search for upper bounds and information-theoretic bounds on the number of queries needed to detect a target t, assuming that the queries incorporate some degree of "noise". In one of the variations of this model [2, 10, 11], each query independently returns with probability $p > \frac{1}{2}$ a direction to a shortest path from the queried vertex q to the target, and with probability $1 - p$ an arbitrary edge (possibly adversarially chosen) incident to q. The study of this problem was initiated in [11], where $\Omega(\log n)$ and $O(\log n)$ bounds on the number of queries were established for a path with n vertices. This information-theoretic lower bound of [11] was matched by an improved upper bound in [2]. The same matching bound was extended to general graphs in [10].

In a further "noisy" variation of binary search, every vertex v of the graph is assigned a fixed edge incident to v (also called the "advice" at v). Then, for a fraction $p > \frac{1}{2}$ of the vertices, the advice directs to a shortest path towards t, while for the rest of the vertices the advice is arbitrary, i.e. potentially misleading or adversarially chosen [3]. This problem setting is motivated by the situation of a tourist driving a car in an unknown country that was hit by a hurricane which resulted in some fraction of road-signs being turned in an arbitrary and unrecognizable way. The question now becomes whether it is still possible to navigate through such a disturbed and misleading environment and to detect the unknown target by asking only few queries (i.e. taking advice only from a few road-signs). It turns out that, apart from its obvious relevance to data structure search, this problem also appears in

artificial intelligence as it can model searching using unreliable heuristics [3, 20, 23]. Moreover this problem also finds applications outside computer science, such as in navigation issues in the context of collaborative transport by ants [12].

Another way of incorporating some "noise" in the query responses, while trying to detect a target, is to have *multiple targets* hidden in the graph. Even if there exist only two unknown targets t_1 and t_2, the response of each query is potentially confusing even if *every* query correctly directs to a shortest path from the queried vertex to one of the targets. The reason of confusion is that now a detecting algorithm does not know to *which* of the hidden targets each query directs. In the context of the above example of a tourist driving a car in an unknown country, imagine there are two main football teams, each having its own stadium. A fraction $0 < p_1 < 1$ of the population supports the first team and a fraction $p_2 = 1 - p_1$ the second one, while the supporters of each team are evenly distributed across the country. The driver can now ask questions of the type "where is the football stadium?" to random local people along the way, in an attempt to visit *both* stadiums. Although every response will be honest, the driver can never be sure which of the two stadiums the local person meant. Can the tourist still detect both stadiums quickly enough? To the best of our knowledge the problem of detecting multiple targets in graphs has not been studied so far; this is one of the main topics of the present paper.

The problem of detecting a target within a graph can be seen as a special case of a two-player game introduced by Renyi [25] and rediscovered by Ulam [27]. This game does not necessarily involve graphs: the first player seeks to detect an element known to the second player in some search space with n elements. To this end, the first player may ask arbitrary yes/no questions and the second player replies to them honestly or not (according to the details of each specific model). Pelc [24] gives a detailed taxonomy for this kind of games. *Group testing* is a sub-category of these games, where the aim is to detect all unknown objects in a search space (not necessarily a graph) [9]. Thus, group testing is related to the problem of detecting multiple targets in graphs, which we study in this paper.

1.1 Our contribution

In this paper we systematically investigate the problem of detecting one or multiple hidden targets in a graph. Our work is driven by the open questions posed by the recent paper of Emamjomeh-Zadeh et al. [10] which dealt with the detection of a single target with and without "noise". More specifically, Emamjomeh-Zadeh et al. [10] asked for further fundamental generalizations of the model which would be of interest, namely (a) detecting a single target when the query response is a direction to an *approximately shortest path*, (b) detecting a single target when querying a vertex that is an *approximate median* of the current candidates' set S (instead of an exact one), and (c) detecting *multiple targets*, for which to the best of our knowledge no progress has been made so far.

We resolve question (a) in Section 2.1 by proving that *any* algorithm requires $\Omega(n)$ queries to detect a single target t, assuming that a query directs to a path with an approximately shortest length to t. Our results hold essentially for any approximation guarantee, i.e. for 1-additive and for $(1+\varepsilon)$-multiplicative approximations.

Regarding question (b), we first prove in Section 2.2 that, for any constant $0 < \varepsilon < 1$, the algorithm of [10] requires at least $\Omega(\sqrt{n})$ queries when we query each time an $(1+\varepsilon)$-approximate median (i.e. an $(1+\varepsilon)$-approximate minimizer of the potential Φ over the candidates' set S). Second, to resolve this lower bound, we introduce in Section 2.3 a new potential Γ. This new potential can be efficiently computed and, in addition, guarantees that, for any constant $0 \leq \varepsilon < 1$, the target t can be detected in $O(\log n)$ queries even when an $(1+\varepsilon)$-approximate median (with respect to Γ) is queried each time.

Regarding question (c), we initiate in Section 3 the study for detecting multiple targets on graphs by focusing mainly to the case of two targets t_1 and t_2. We assume throughout that every query provides a correct answer, in the sense that it always returns a direction to a shortest path from the queried vertex either to t_1 or to t_2. The "noise" in this case is that the algorithm does not know whether a query is returning a direction to t_1 or to t_2. Initially we observe in Section 3 that *any* algorithm requires $\frac{n}{2} - 1$ (resp. $n - 2$) queries in the worst case to detect one target (resp. both targets) if each query directs adversarially to one of the two targets. Hence, in the remainder of Section 3, we consider the case where each query independently directs to the first target t_1 with a constant probability p_1 and to the second target t_2 with probability $p_2 = 1 - p_1$. For the case of trees, we prove in Section 3 that both targets can be detected with high probability within $O(\log n)$ queries.

For general graphs, we distinguish between *biased* queries ($p_1 > p_2$) in Section 3.1 and *unbiased* queries ($p_1 = p_2 = \frac{1}{2}$) in Section 3.2. For biased queries, we observe that we can utilize the algorithm of Emamjomeh-Zadeh et al. [10] to detect the first target t_1 with high probability in $O(\log n)$ queries; this can be done by considering the queries that direct to t_2 as "noise". Thus our objective becomes to detect the target t_2 in a polylogarithmic number of queries. Notice here that we cannot apply the "noisy" framework of [10] to detect the second target t_2 , since now the "noise" is larger than $\frac{1}{2}$. We derive a probabilistic algorithm that overcomes this problem and detects the target t_2 with high probability in $O(\Delta \log^2 n)$ queries, where Δ is the maximum degree of a vertex in the graph. Thus, whenever $\Delta = O(poly \log n)$, a polylogarithmic number of queries suffices to detect t_2. In contrast, we prove in Section 3.2 that, for unbiased queries, *any* deterministic (possibly adaptive) algorithm that detects at least one of the targets requires at least $\frac{n}{2} - 1$ queries, even in an unweighted cycle. Extending this lower bound for two targets, we prove that, assuming $2c \geq 2$ different targets and unbiased queries, *any* deterministic (possibly adaptive) algorithm requires at least $\frac{n}{2} - c$ queries to detect one of the targets.

Departing from the fact that our best upper bound on the number of biased queries in Section 3.1 is not polylogarithmic when the maximum degree Δ is not polylogarithmic, we investigate in Section 4 several variations of queries that provide more informative responses. In Section 4.1 we turn our attention to "direction-distance" biased queries which return with probability p_i both the direction to a shortest path to t_i and the distance between the queried vertex and t_i. In Section 4.2 we consider another type of a biased query which combines the classical "direction" query and an edge-variation of it. For both query types of Sections 4.1 and 4.2 we prove that the second target t_2 can be detected with high probability in $O(\log^3 n)$ queries. Furthermore, in Sections 4.3 and 4.4 we investigate two further generalizations of the "direction" query which make the target detection problem trivially hard and trivially easy to solve, respectively.

1.2 Our Model and Notation

We consider connected, simple, and undirected graphs. A graph $G = (V, E)$, where $|V| = n$, is given along with a *weight function* $w : E \to \mathbb{R}^+$ on its edges; if $w(e) = 1$ for every $e \in E$ then G is *unweighted*. An edge between two vertices v and u of G is denoted by vu, and in this case v and u are said to be *adjacent* . The distance $d(v, u)$ between vertices v and u is the length of a shortest path between v and u with respect to the weight function w. Since the graphs we consider are undirected, $d(u, v) = d(v, u)$ for every pair of vertices v, u. Unless specified otherwise, all logarithms are taken with base 2. Whenever an event happens with probability at least $1 - \frac{1}{n^\alpha}$ for some $\alpha > 0$, we say that it happens *with high probability*.

The *neighborhood* of a vertex $v \in V$ is the set $N(v) = \{u \in V : vu \in E\}$ of its adjacent vertices. The cardinality of $N(v)$ is the *degree* $\deg(v)$ of v. The maximum degree among

all vertices in G is denoted by $\Delta(G)$, i.e. $\Delta(G) = \max\{\deg(v) : v \in V\}$. For two vertices v and $u \in N(v)$ we denote by $N(v,u) = \{x \in V : d(v,x) = w(vu) + d(u,x)\}$ the set of vertices $x \in V$ for which there exists a shortest path from v to x, starting with the edge vu. Note that, in general, $N(u,v) \neq N(v,u)$. Let $T = \{t_1, t_2, \cdots, t_{|T|}\} \subseteq V$ be a set of (initially unknown) *target vertices* . A *direction query* (or simply *query*) at vertex $v \in V$ returns with probability p_i a neighbor $u \in N(v)$ such that $t_i \in N(u,v)$, where $\sum_{i=1}^{|T|} p_i = 1$. If there exist more than one such vertices $u \in N(v)$ leading to t_i via a shortest path, the direction query returns an arbitrary one among them, i.e. possibly chosen adversarially, unless specified otherwise. Moreover, if the queried vertex v is equal to one of the targets $t_i \in T$, this is revealed by the query with probability p_i.

2 Detecting a Unique Target

In this section we consider the case where there is only one unknown target $t = t_1$, i.e. $T = \{t\}$. In this case the direction query at vertex v always returns a neighbor $u \in N(v)$ such that $t \in N(u,v)$. For this problem setting, Emamjomeh-Zadeh et al. [10] provided a polynomial-time algorithm which detects the target t in at most $\log n$ direction queries. During its execution, the algorithm of [10] maintains a "candidates' set" $S \subseteq V$ such that always $t \in S$, where initially $S = V$. At every iteration the algorithm computes in polynomial time a vertex v (called the *median* of S) which minimizes a potential $\Phi_S(v)$ among all vertices of the current set S. Then it queries a median v of S and it reduces the candidates' set S to $S \cap N(v,u)$, where u is the vertex returned by the direction query at v. The upper bound $\log n$ of the number of queries in this algorithm follows by the fact that always $|S \cap N(v,u)| \leq \frac{|S|}{2}$, whenever v is the median of S.

2.1 Bounds for Approximately Shortest Paths

We provide lower bounds for both additive and multiplicative approximation queries. A *c-additive approximation query* at vertex $v \in V$ returns a neighbor $u \in N(v)$ such that $w(vu) + d(u,t) \leq d(v,t) + c$. Similarly, an $(1+\varepsilon)$-*multiplicative approximation query* at vertex $v \in V$ returns a neighbor $u \in N(v)$ such that $w(vu) + d(u,t) \leq (1+\varepsilon) \cdot d(v,t)$.

It is not hard to see that in the unweighted clique with n vertices any algorithm requires in worst case $n-1$ 1-additive approximation queries to detect the target t. Indeed, in this case $d(v,t) = 1$ for every vertex $v \neq t$, while every vertex $u \notin \{v,t\}$ is a valid response of an 1 -additive approximation query at v. Since in the case of the unweighted clique an additive 1-approximation is the same as a multiplicative 2-approximation of the shortest path, it remains unclear whether 1 -additive approximation queries allow more efficient algorithms for graphs with large diameter. In the next theorem we strengthen this result to graphs with unbounded diameter.

▶ **Theorem 1.** *Assuming* 1*-additive approximation queries, any algorithm requires at least* $n-1$ *queries to detect the target* t*, even in graphs with unbounded diameter.*

In the next theorem we extend Theorem 1 by showing a lower bound of $n \cdot \frac{\varepsilon}{4}$ queries when we assume $(1+\varepsilon)$-multiplicative approximation queries.

▶ **Theorem 2.** *Let* $\varepsilon > 0$*. Assuming* $(1+\varepsilon)$ *-multiplicative approximation queries, any algorithm requires at least at least* $n \cdot \frac{\varepsilon}{4}$ *queries to detect the target* t*.*

2.2 Lower Bound for querying the Approximate Median

The potential $\Phi_S : V \to \mathbb{R}^+$ of [10], where $S \subseteq V$, is defined as follows. For any set $S \subseteq V$ and any vertex $v \in V$, the potential of v is $\Phi_S(v) = \sum_{u \in S} d(v,u)$. A vertex $x \in V$ is an $(1+\varepsilon)$-approximate minimizer for the potential Φ over a set S (i.e. an $(1+\varepsilon)$-median of S) if $\Phi_S(x) \leq (1+\varepsilon)\min_{v \in V} \Phi_S(v)$, where $\varepsilon > 0$. We prove that an algorithm querying at each iteration always an $(1+\varepsilon)$-median of the current candidates' set S needs $\Omega(\sqrt{n})$ queries.

▶ **Theorem 3.** *Let $\varepsilon > 0$. If the algorithm of [10] queries at each iteration an $(1+\varepsilon)$-median for the potential function Φ, then at least $\Omega(\sqrt{n})$ queries are required to detect the target t in a graph G with n vertices, even if the graph G is a tree.*

2.3 Upper Bound for querying the Approximate Median

In this section we introduce a new potential function $\Gamma_S : V \to \mathbb{N}$ for every $S \subseteq V$, which overcomes the problem occured in Section 2.2. This new potential guarantees efficient detection of t in at most $O(\log n)$ queries, even when we always query an $(1+\varepsilon)$-median of the current candidates' set S (with respect to the new potential Γ), for any constant $0 < \varepsilon < 1$. Our algorithm is based on the approach of [10], however we now query an approximate median of the current set S with respect to Γ (instead of an exact median with respect to Φ of [10]).

▶ **Definition 4** (Potential Γ). Let $S \subseteq V$ and $v \in V$. Then $\Gamma_S(v) = \max\{|N(v,u) \cap S| : u \in N(v)\}$.

▶ **Theorem 5.** *Let $0 \leq \varepsilon < 1$. There exists an efficient adaptive algorithm which detects the target t in at most $\frac{\log n}{1-\log(1+\varepsilon)}$ queries, by querying at each iteration an $(1+\varepsilon)$-median for the potential function Γ.*

Proof. Our proof closely follows the proof of Theorem 3 of [10]. Let $S \subseteq V$ be an arbitrary set of vertices of G such that $t \in S$. We will show that there exists a vertex $v \in V$ such that $\Gamma_S(v) \leq \frac{|S|}{2}$. First recall the potential $\Phi_S(v) = \sum_{x \in S} d(v,x)$. Let now $v_0 \in V$ be a vertex such that $\Phi_S(v_0)$ is minimized, i.e. $\Phi_S(v_0) \leq \Phi_S(v)$ for every $v \in V$. Let $u \in N(v_0)$ be an *arbitrary* vertex adjacent to v_0. We will prove that $|N(v_0,u) \cap S| \leq \frac{|S|}{2}$. Denote $S^+ = N(v_0,u) \cap S$ and $S^- = S \setminus S^+$. By definition, for every $x \in S^+$, the edge v_0u lies on a shortest path from v_0 to x, and thus $d(u,x) = d(v_0,x) - w(v_0u)$. On the other hand, trivially $d(u,x) \leq d(v_0,x) + w(v_0u)$ for every $x \in S$, and thus in particular for every $x \in S^-$. Therefore $\Phi_S(v_0) \leq \Phi_S(u) \leq \Phi_S(v_0) + (|S^-| - |S^+|) \cdot w(v_0u)$, and thus $|S^+| \leq |S^-|$. That is, $|N(v_0,u) \cap S| = |S^+| \leq \frac{|S|}{2}$, since $S^- = S \setminus S^+$. Therefore which then implies that $\Gamma_S(v_0) \leq \frac{|S|}{2}$ as the choice of the vertex $u \in N(v_0)$ is arbitrary.

Let $v_m \in V$ be an exact median of S with respect to Γ. That is, $\Gamma_S(v_m) \leq \Gamma_S(v)$ for every $v \in V$. Note that $\Gamma_S(v_m) \leq \Gamma_S(v_0) \leq \frac{|S|}{2}$. Now let $0 \leq \varepsilon < 1$ and let $v_a \in V$ be an $(1+\varepsilon)$-median of S with respect to Γ. Then $\Gamma_S(v_a) \leq (1+\varepsilon)\Gamma_S(v_m) \leq \frac{1+\varepsilon}{2}|S|$. Our adaptive algorithm proceeds as follows. Similarly to the algorithm of [10] (see Theorem 3 of [10]), our adaptive algorithm maintains a candidates' set S, where initially $S = V$. At every iteration our algorithm queries an arbitrary $(1+\varepsilon)$-median $v_m \in V$ of the current set S with respect to the potential Γ. Let $u \in N(v_m)$ be the vertex returned by this query; the algorithm updates S with the set $N(v,u) \cap S$. Since $\Gamma_S(v_a) \leq \frac{1+\varepsilon}{2}|S|$ as we proved above, it follows that the updated candidates' set has cardinality at most $\frac{1+\varepsilon}{2}|S|$. Thus, since initially $|S| = n$, our algorithm detects the target t after at most $\log_{(\frac{2}{1+\varepsilon})} n = \frac{\log n}{1-\log(1+\varepsilon)}$ queries. ◀

Notice in the statement of Theorem 5 that for $\varepsilon = 0$ (i.e. when we always query an exact median) we get an upper bound of $\log n$ queries, as in this case the size of the candidates' set decreases by a factor of at least 2. Furthermore notice that the reason that the algorithm of [10] is not query-efficient when querying an $(1+\varepsilon)$-median is that the potential $\Phi_S(v)$ of [10] can become quadratic in $|S|$, while on the other hand the value of our potential $\Gamma_S(v)$ can be at most $|S|$ by Definition 4, for every $S \subseteq V$ and every $v \in V$. Furthermore notice that, knowing only the value $\Phi_S(v)$ for some vertex $v \in V$ is not sufficient to provide a guarantee for the proportional reduction of the set S when querying v. In contrast, just knowing the value $\Gamma_S(v)$ directly provides a guarantee that, if we query vertex v the set S will be reduced by a proportion of $\frac{\Gamma_S(v)}{|S|}$, regardless of the response of the query. Therefore, in practical applications, we may not need to necessarily compute an (exact or approximate) median of S to make significant progress.

3 Detecting Two Targets

In this section we consider the case where there are two unknown targets t_1 and t_2, i.e. $T = \{t_1, t_2\}$. In this case the direction query at vertex v returns with probability p_1 (resp. with probability $p_2 = 1 - p_1$) a neighbor $u \in N(v)$ such that $t_1 \in N(v,u)$ (resp. $t_2 \in N(v,u)$). Detecting more than one unknown targets has been raised as an open question by Emamjomeh-Zadeh et al. [10], while to the best of our knowledge no progress has been made so far in this direction. Here we deal with both problems of detecting at least one of the targets and detecting both targets. We study several different settings and derive both positive and negative results for them. Each setting differs from the other ones on the "freedom" the adversary has on responding to queries, or on the power of the queries themselves. We will say that the response to a query *directs to* t_i, where $i \in \{1,2\}$, if the vertex returned by the query lies on a shortest path between the queried vertex and t_i.

It is worth mentioning here that, if an adversary would be free to arbitrarily choose which t_i each query directs to (i.e. instead of directing to t_i with probability p_i), then any algorithm would require at least $\lfloor \frac{n}{2} \rfloor$ (resp. $n-2$) queries to detect at least one of the targets (resp. both targets), even when the graph is a path. Indeed, consider a path $v_1, \ldots, v_n$ where $t_1 \in \{v_1, \ldots, v_{\lfloor \frac{n}{2} \rfloor}\}$ and $t_2 \in \{v_{\lfloor \frac{n}{2} \rfloor + 1}, \ldots, v_n\}$. Then, for every $i \in \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$, the query at v_i would return v_{i+1}, i.e. it would direct to t_2. Similarly, for every $i \in \{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$, the query at v_i would return v_{i-1}, i.e. it would direct to t_1. It is not hard to verify that in this case the adversary could "hide" the target t_1 at any of the first $\lfloor \frac{n}{2} \rfloor$ vertices which is not queried by the algorithm and the target t_2 on any of the last $n - \lfloor \frac{n}{2} \rfloor$ vertices which is not queried. Hence, at least $\lfloor \frac{n}{2} \rfloor$ queries (resp. $n-2$ queries) would be required to detect one of the targets (resp. both targets) in the worst case.

As a warm-up, we provide in the next theorem an efficient algorithm that detects with high probability both targets in a tree using $O(\log^2 n)$ queries.

▶ **Theorem 6.** *For any constant $0 < p_1 < 1$, we can detect with probability at least $\left(1 - \frac{\log n}{n}\right)^2$ both targets in a tree with n vertices using $O(\log^2 n)$ queries.*

Since in a tree both targets t_1, t_2 can be detected with high probability in $O(\log^2 n)$ queries by Theorem 6, we consider in the remainder of the section arbitrary graphs instead of trees. First we consider in Section 3.1 *biased* queries, i.e. queries with $p_1 > \frac{1}{2}$. Second we consider in Section 3.2 *unbiased* queries, i.e. queries with $p_1 = p_2 = \frac{1}{2}$.

Algorithm 1 Given t_1, detect t_2 with high probability with $O(\Delta \log^2 n)$ queries

$S \leftarrow V$; $c \leftarrow \frac{7(1+p_1)^2}{p_1(1-p_1)^2}$
while $|S| > 1$ **do**
  Compute an (approximate) median v of S with respect to potential Γ; Compute $E_{t_1}(v)$
  Query $c\Delta \log n$ times vertex v; Compute the multiset $Q(v)$ of these query responses
  if $Q(v) \setminus E_{t_1}(v) \neq \emptyset$ **then**
    Pick a vertex $u \in Q(v) \setminus E_{t_1}(v)$ and set $S \leftarrow S \cap N(v, u)$
  else
    Pick a most frequent vertex $u \in Q(v)$ and set $S \leftarrow S \cap N(v, u)$
return the unique vertex in S

3.1 Upper Bounds for Biased Queries

In this section we consider biased queries which direct to t_1 with probability $p_1 > \frac{1}{2}$ and to t_2 with probability $p_2 = 1 - p_1 < \frac{1}{2}$. As we can detect in this case the first target t_1 with high probability in $O(\log n)$ queries by using the "noisy" framework of [10], our aim becomes to detect the second target t_2 with the fewest possible queries, once we have already detected t_1.

For every vertex v and every $i \in \{1, 2\}$, denote by $E_{t_i}(v) = \{u \in N(v) : t_i \in N(v, u)\}$ the set of neighbors of v such that the edge uv lies on a shortest path from v to t_i. Note that the sets $E_{t_1}(v)$ and $E_{t_2}(v)$ can be computed in polynomial time, e.g. using Dijkstra's algorithm. We assume that, once a query at vertex v has chosen which target t_i it directs to, it returns each vertex of $E_{t_i}(v)$ equiprobably and independently from all other queries. Therefore, each of the vertices of $E_{t_1}(v) \setminus E_{t_2}(v)$ is returned by the query at v with probability $\frac{p_1}{|E_{t_1}(v)|}$, each vertex of $E_{t_2}(v) \setminus E_{t_1}(v)$ is returned with probability $\frac{1-p_1}{|E_{t_2}(v)|}$, and each vertex of $E_{t_1}(v) \cap E_{t_2}(v)$ is returned with probability $\frac{p_1}{|E_{t_1}(v)|} + \frac{1-p_1}{|E_{t_2}(v)|}$. We will show in Theorem 8 that, under these assumptions, we detect the second target t_2 with high probability in $O(\Delta \log^2 n)$ queries where Δ is the maximum degree of the graph.

The high level description of our algorithm (Algorithm 1) is as follows. Throughout the algorithm we maintain a candidates' set S of vertices in which t_2 belongs with high probability. Initially $S = V$. In each iteration we first compute an (exact or approximate) median v of S with respect to the potential Γ (see Section 2.3). Then we compute the set $E_{t_1}(v)$ (this can be done as t_1 has already been detected) and we query $c\Delta \log n$ times vertex v, where $c = \frac{7(1+p_1)^2}{p_1(1-p_1)^2}$ is a constant. Denote by $Q(v)$ the multiset of size $c\Delta \log n$ that contains the vertices returned by these queries at v. If at least one of these $O(\Delta \log n)$ queries at v returns a vertex $u \notin E_{t_1}(v)$, then we can conclude that $u \in E_{t_2}(v)$, and thus we update the set S by $S \cap N(v, u)$. Assume otherwise that all $O(\Delta \log n)$ queries at v return vertices of $E_{t_1}(v)$. Then we pick a vertex $u_0 \in N(v)$ that has been returned most frequently among the $O(\Delta \log n)$ queries at v, and we update the set S by $S \cap N(v, u_0)$. As it turns out, $u_0 \in E_{t_2}(v)$ with high probability. Since we always query an (exact or approximate) median v of the current candidates' set S with respect to the potential Γ, the size of S decreases by a constant factor each time. Therefore, after $O(\log n)$ updates we obtain $|S| = 1$. It turns out that, with high probability, each update of the candidates' set was correct, i.e. $S = \{t_2\}$. Since for each update of S we perform $O(\Delta \log n)$ queries, we detect t_2 with high probability in $O(\Delta \log^2 n)$ queries in total.

Recall that every query at v returns a vertex $u \in E_{t_1}(v)$ with probability p_1 and a vertex $u \in E_{t_2}(v)$ with probability $1 - p_1$. Therefore, for every $v \in V$ the multiset $Q(v)$ contains at

least one vertex $u \in E_{t_2}(v)$ with probability at least $1 - p_1^{|Q(v)|} = 1 - p_1^{|c\Delta \log n|}$. In the next lemma we prove that, every time we update S using Step 8, the updated set contains t_2 with high probability.

▶ **Lemma 7.** *Let $S \subseteq V$ such that $t_2 \in S$ and let $S' = S \cap N(v, u)$ be the updated set at Step 8 of Algorithm 1. Then $t_2 \in S'$ with probability at least $1 - \frac{2}{n}$.*

Proof. Let $\delta = \frac{1-p_1}{1+p_1}$ and $c = \frac{7(1+p_1)^2}{p_1(1-p_1)^2}$ be two constants. Recall that each of the vertices of $E_{t_1}(v) \setminus E_{t_2}(v)$ is returned by the query at v with probability $\frac{p_1}{|E_{t_1}(v)|}$, each vertex of $E_{t_2}(v) \setminus E_{t_1}(v)$ is returned with probability $\frac{1-p_1}{|E_{t_2}(v)|}$, and each vertex of $E_{t_1}(v) \cap E_{t_2}(v)$ is returned with probability $\frac{p_1}{|E_{t_1}(v)|} + \frac{1-p_1}{|E_{t_2}(v)|}$. Observe that these probabilities are the expected frequencies for these vertices in $Q(v)$. Recall that Step 8 is executed only in the case where $Q(v) \subseteq E_{t_1}(v)$. To prove the lemma it suffices to show that, whenever $Q(v) \subseteq E_{t_1}(v)$, the most frequent element of $Q(v)$ belongs to $E_{t_1}(v) \cap E_{t_2}(v)$ with high probability.

First note that, for the chosen value of δ,

$$(1+\delta)\frac{p_1}{|E_{t_1}(v)|} < (1-\delta)\left(\frac{p_1}{|E_{t_1}(v)|} + \frac{1-p_1}{|E_{t_2}(v)|}\right) \tag{1}$$

Let $u \in E_{t_1}(v) \setminus E_{t_2}(v)$, i.e. the query at v directs to t_1 but not to t_2. We define the random variable $Z_i(u)$, such that $Z_i(u) = 1$ if u is returned by the i-th query at v and $Z_i(u) = 0$ otherwise. Furthermore define $Z(u) = \sum_{i=1}^{c\Delta \log n} Z_i(u)$. Since $\Pr(Z_i(u) = 1) = \frac{p_1}{|E_{t_1}(v)|}$, it follows that $E(Z(u)) = c\Delta \log n \frac{p_1}{|E_{t_1}(v)|}$ by the linearity of expectation. Then, using Chernoff's bounds we can prove that

$$\Pr(Z(u) \geq (1+\delta)E(Z(u))) \leq \frac{1}{n^2}. \tag{2}$$

Thus (2) implies that the probability that there exists at least one $u \in E_{t_1}(v) \setminus E_{t_2}(v)$ such that $Z(u) \geq (1+\delta)E(Z(u))$ is

$$\Pr\left(\exists u \in E_{t_1}(v) \setminus E_{t_2}(v) : Z(u) \geq (1+\delta)\frac{p_1}{|E_{t_1}(v)|}\right) < (\Delta - 1)\frac{1}{n^2} < \frac{1}{n}. \tag{3}$$

Now let $u' \in E_{t_1}(v) \cap E_{t_2}(v)$. Similarly to the above we define the random variable $Z'_i(u')$, such that $Z'_i(u') = 1$ if u' is returned by the i -th query at v and $Z'_i(u') = 0$ otherwise. Furthermore define $Z'(u') = \sum_{i=1}^{c\Delta \log n} Z'_i(u')$. Since $\Pr(Z'_i(u') = 1) = \frac{p_1}{|E_{t_1}(v)|} + \frac{1-p_1}{|E_{t_2}(v)|}$, it follows that $E(Z(u)) = c\Delta \log n \left(\frac{p_1}{|E_{t_1}(v)|} + \frac{1-p_1}{|E_{t_2}(v)|}\right)$ by the linearity of expectation. Then we obtain similarly to (2) that

$$\Pr(Z'(u') \leq (1-\delta)E(Z'(u'))) < \frac{1}{n^2} \tag{4}$$

Thus, it follows by the union bound and by (1), (3), and (4) that

$$\Pr(\exists u \in E_{t_1}(v) \setminus E_{t_2}(v) : Z(u) \geq Z'(u')) \leq \frac{2}{n}. \tag{5}$$

That is, the most frequent element of $Q(v)$ belongs to $E_{t_1}(v) \cap E_{t_2}(v)$ with probability at least $1 - \frac{2}{n}$. This completes the proof of the lemma. ◀

With Lemma 7 in hand we can now prove the main theorem of the section.

▶ **Theorem 8.** *Given t_1, Algorithm 1 detects t_2 in $O(\Delta \log^2 n)$ queries with probability at least $(1 - \frac{2}{n})^{O(\log n)}$.*

Note by Theorem 8 that, whenever $\Delta = O(poly \log n)$ we can detect both targets t_1 and t_2 in $O(poly \log n)$ queries. However, for graphs with larger maximum degree Δ, the value of the maximum degree dominates any polylogarithmic factor in the number of queries. The intuitive reason behind this is that, for an (exact or approximate) median v of the current set S, whenever $\deg(v)$ and $E_{t_1}(v)$ are large and $E_{t_2}(v) \subseteq E_{t_1}(v)$, we can not discriminate with a polylogarithmic number of queries between the vertices of $E_{t_2}(v)$ and the vertices of $E_{t_1}(v) \setminus E_{t_2}(v)$ with large enough probability. Although this argument does not give any lower bound for the number of queries in the general case (i.e. when Δ is unbounded), it seems that more informative queries are needed to detect both targets with polylogarithmic queries in general graphs. We explore such more informative queries in Section 4.

3.2 Lower Bounds for Unbiased Queries

In this section we consider unbiased queries, i.e. queries which direct to each of the targets t_1, t_2 with equal probability $p_1 = p_2 = \frac{1}{2}$. In this setting every query is indifferent between the two targets, and thus the "noisy" framework of [10] cannot be applied for detecting any of the two targets. In particular we prove in the next theorem that any deterministic (possibly adaptive) algorithm needs at least $\frac{n}{2} - 1$ queries to detect one of the two targets.

▶ **Theorem 9.** *Let $p_1 = p_2 = \frac{1}{2}$. Then any deterministic (possibly adaptive) algorithm needs at least $\frac{n}{2} - 1$ queries to detect one of the two targets, even in an unweighted cycle.*

In the next theorem we generalize the lower bound of Theorem 9 to the case of $2c \geq 2$ different targets $T = \{t_1, t_2, \ldots, t_{2c}\}$ and the query to any vertex $v \notin T$ is unbiased, i.e. $p_i = \frac{1}{2c}$ for every $i \in \{1, 2, \ldots, 2c\}$.

▶ **Theorem 10.** *Suppose that there are $2c$ targets in the graph and let $p_i = \frac{1}{2c}$ for every $i \in \{1, 2, \ldots, 2c\}$. Then, any deterministic (possibly adaptive) algorithm requires at least $\frac{n}{2} - c$ queries to locate at least one target, even in an unweighted cycle.*

4 More Informative Queries for Two Targets

A natural alternative to obtain query-efficient algorithms for multiple targets, instead of restricting the maximum degree Δ of the graph (see Section 3.1), is to consider queries that provide more informative responses in general graphs. As we have already observed in Section 3.1, it is not clear whether it is possible to detect multiple targets with $O(poly \log n)$ *direction queries* in an arbitrary graph. In this section we investigate natural variations and extensions of the direction query for multiple targets which we studied in Section 3.

4.1 Direction-Distance Biased Queries

In this section we strengthen the direction query in a way that it also returns the value of the distance between the queried vertex and one of the targets. More formally, a *direction-distance query* at vertex $v \in V$ returns with probability p_i a pair (u, ℓ), where $u \in N(v)$ such that $t_i \in N(u, v)$ and $d(v, t_i) = \ell$. Note that here we impose again that all p_i's are constant and that $\sum_{i=1}^{|T|} p_i = 1$, where $T = \{t_1, t_2, \ldots, t_{|T|}\}$ is the set of targets. We will say that the response (u, ℓ) to a direction-distance query at vertex v *directs to* t_i if $t_i \in N(v, u)$ and $\ell = d(v, t_i)$. Similarly to our assumptions on the direction query, whenever there exist more than one such vertices $u \in N(v)$ leading to t_i via a shortest path, the direction-distance query returns an arbitrary vertex u among them (possibly chosen adversarially). Moreover,

Algorithm 2 Given t_1, detect t_2 with high probability with $O(\log^3 n)$ direction-distance queries

1: $S \leftarrow V$
2: **while** $|S| > 1$ **do**
3: Compute an (approximate) median v of S with respect to potential Γ; Compute $E_{t_1}(v)$
4: Query $\log n$ times vertex v; Compute the set $Q(v)$ of different query responses
5: **if** there exists a pair $(u, \ell) \in Q(v)$ such that $u \notin E_{t_1}(v)$ or $\ell \neq d(v, t_1)$ **then**
6: $S \leftarrow S \cap N(v, u)$
7: **else**
8: **for** every $(u, \ell) \in Q(v)$ **do**
9: Query $\log n$ times vertex u; Compute the set $Q(u)$ of different query responses
10: **if** for every $(z, \ell') \in Q(u)$ we have $\ell' = \ell - w(vu)$ **then**
11: $S \leftarrow S \cap N(v, u)$; Goto line 2
12: **return** the unique vertex of S

if the queried vertex v is equal to one of the targets $t_i \in T$, this is revealed by the query with probability p_i. These direction-distance queries have also been used in [10] for detecting one single target in directed graphs.

Here we consider the case of two targets and *biased queries*, i.e. $T = \{t_1, t_2\}$ where $p_1 > p_2$. Similarly to Section 3.1, initially we can detect the first target t_1 with high probability in $O(\log n)$ queries using the "noisy" model of [10]. Thus, in what follows we assume that t_1 has already been detected. We will show that the second target t_2 can be detected with high probability with $O(\log^3 n)$ additional direction-distance queries using Algorithm 2. The high level description of our algorithm is the following. We maintain a candidates' set S such that at every iteration $t_2 \in S$ with high probability. Each time we update the set S, its size decreases by a constant factor. Thus we need to shrink the set S at most $\log n$ times. In order to shrink S one time, we first compute an $(1 + \varepsilon)$ -median v of the current set S and we query $\log n$ times this vertex v. Denote by $Q(v)$ the set of all different responses of these $\log n$ direction-distance queries at v. As it turns out, the responses in $Q(v)$ might not always be enough to shrink S such that it still contains t_2 with high probability. For this reason we also query $\log n$ times each of the $\log n$ neighbors $u \in N(v)$, such that $(u, \ell) \in Q(v)$ for some $\ell \in \mathbb{N}$. After these $\log^2 n$ queries at v and its neighbors, we can safely shrink S by a constant factor, thus detecting the target t_2 with high probability in $\log^3 n$ queries.

For the description of our algorithm (see Algorithm 2) recall that, for every vertex v, the set $E_{t_1}(v) = \{u \in N(v) : t_1 \in N(v, u)\}$ contains all neighbors of v such that the edge uv lies on a shortest path from v to t_1.

▶ **Theorem 11.** *Given t_1, Algorithm 2 detects t_2 in at most $O(\log^3 n)$ queries with probability at least $1 - O\left(\log n \cdot p_1^{\log n}\right)$.*

4.2 Vertex-Direction and Edge-Direction Biased Queries

An alternative natural variation of the direction query is to *query an edge* instead of querying a vertex. More specifically, the direction query (as defined in Section 1.2) queries a vertex $v \in V$ and returns with probability p_i a neighbor $u \in N(v)$ such that $t_i \in N(u, v)$. Thus, as this query always queries a vertex, it can be also referred to as a *vertex-direction query*. Now

we define the *edge-direction query* as follows: it queries an ordered pair of adjacent vertices (v, u) and it returns with probability p_i `YES` (resp. `NO`) if $t_i \in N(v, u)$ (resp. if $t_i \notin N(v, u)$). Similarly to our notation in the case of vertex-direction queries, we will say that the response `YES` (resp. `NO`) to an edge-direction query at the vertex pair (v, u) *refers* to t_i if $t_i \in N(v, u)$ (resp. if $t_i \notin N(v, u)$). Similar but different edge queries for detecting one single target on trees have been investigated in [10, 13, 21, 26].

Here we consider the case where both vertex-direction and edge-direction queries are available to the algorithm, and we focus again to the case of two targets and *biased queries*, i.e. $T = \{t_1, t_2\}$ where $p_1 > p_2$. Similarly to Sections 3.1 and 4.1, we initially detect t_1 with high probability in $O(\log n)$ vertex-direction queries using the "noisy" model of [10]. Thus, in the following we assume that t_1 has already been detected.

▶ **Theorem 12.** *Given t_1, it is possible to detect t_2 in at most $O(\log^2 n)$ vertex-direction queries and $O(\log^3 n)$ edge–direction queries with probability at least $1 - O(\log n \cdot p_1^{\log n})$.*

4.3 Two-Direction Queries

In this section we consider another variation of the direction query that was defined in Section 1.2 (or "vertex-direction query" in the terminology of Section 4.2), which we call *two-direction query*. Formally, a two-direction query at vertex v returns an *unordered pair* of (not necessarily distinct) vertices $\{u, u'\}$ such that $t_1 \in N(v, u)$ and $t_2 \in N(v, u')$. Note here that, as $\{u, u'\}$ is an unordered pair, the response of the two-direction query does not clarify which of the two targets belongs to $N(v, u)$ and which to $N(v, u')$.

Although this type of query may seem at first to be more informative than the standard direction query studied in Section 3, we show that this is not the case. Intuitively, this type of query resembles the unbiased direction query of Section 3.2. To see this, consider e.g. the unweighted cycle where the two targets are placed at two anti-diametrical vertices; then, applying many times the unbiased direction query of Section 3.2 at any specific vertex v reveals with high probability the same information as applying a single two-direction query at v. Based on this intuition the next theorem can be proved with exactly the same arguments as Theorem 9 of Section 3.2.

▶ **Theorem 13.** *Any deterministic (possibly adaptive) algorithm needs at least $\frac{n}{2} - 1$ two-direction queries to detect one of the two targets, even in an unweighted cycle.*

4.4 Restricted Set Queries

The last type of queries we consider is when the query is applied not only to a vertex v of the graph, but also to a subset $S \subseteq V$ of the vertices, and the response of the query is a vertex $u \in N(v)$ such that $t \in N(v, u)$ for at least one of the targets t that belong to the set S. Formally, let T be the set of targets. The *restricted-set query* at the pair (v, S), where $v \in V$ and $S \subseteq V$ such that $T \cap S \neq \emptyset$, returns a vertex $u \in N(v)$ such that $t \in N(v, u)$ for at least one target $t \in T \cap S$. If there exist multiple such vertices $u \in N(v)$, the query returns one of them adversarially. Finally, if we query a pair (v, S) such that $T \cap S = \emptyset$, then the query returns adversarially an arbitrary vertex $u \in N(v)$, regardless of whether the edge vu leads to a shortest path from v to any target in T. That is, the response of the query can be considered in this case as "noise". In the next theorem we prove that this query is very powerful, as $|T| \cdot \log n$ restricted-set queries suffice to detect all targets of the set T.

▶ **Theorem 14.** *Let T be the set of targets. There exists an adaptive deterministic algorithm that detects all targets of T with at most $|T| \cdot \log n$ restricted-set queries.*

References

1 Yosi Ben-Asher, Eitan Farchi, and Ilan Newman. Optimal search in trees. *SIAM J. Comput.*, 28(6):2090–2102, 1999.

2 Michael Ben-Or and Avinatan Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 221–230, 2008.

3 Lucas Boczkowski, Amos Korman, and Yoav Rodeh. Searching on trees with noisy memory. *CoRR*, abs/1611.01403, 2016.

4 Renato Carmo, Jair Donadelli, Yoshiharu Kohayakawa, and Eduardo Sany Laber. Searching in random partially ordered sets. *Theor. Comput. Sci.*, 321(1):41–57, 2004.

5 Ferdinando Cicalese, Tobias Jacobs, Eduardo Sany Laber, and Marco Molinaro. On the complexity of searching in trees and partially ordered structures. *Theor. Comput. Sci.*, 412(50):6879–6896, 2011.

6 Ferdinando Cicalese, Tobias Jacobs, Eduardo Sany Laber, and Caio Dias Valentim. The binary identification problem for weighted trees. *Theor. Comput. Sci.*, 459:100–112, 2012.

7 Constantinos Daskalakis, Richard M. Karp, Elchanan Mossel, Samantha Riesenfeld, and Elad Verbin. Sorting and selection in posets. *SIAM J. Comput.*, 40(3):597–622, 2011.

8 Dariusz Dereniowski. Edge ranking and searching in partial orders. *Discrete Applied Mathematics*, 156(13):2493–2500, 2008.

9 Dingzhu Du and Frank K. Hwang. *Combinatorial Group Testing and its Applications*. World Scientific, Singapore, 1993.

10 Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 519–532, 2016.

11 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994.

12 Ehud Fonio, Yael Heyman, Lucas Boczkowski, Aviram Gelblum, Adrian Kosowski, Amos Korman, and Ofer Feinerman. A locally-blazed ant trail achieves efficient collective navigation despite limited information. *eLife*, page 23 pages, 2016.

13 Ananth V. Iyer, H. Donald Ratliff, and Gopalakrishnan Vijayan. Optimal node ranking of trees. *Inf. Process. Lett.*, 28(5):225–229, 1988.

14 C. Jordan. Sur les assemblages de lignes. *Journal f"ur die reine und angewandte Mathematik*, 70:195–190, 1869.

15 Eduardo Sany Laber, Ruy Luiz Milidiú, and Artur Alves Pessoa. On binary searching with non-uniform costs. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA.*, pages 855–864, 2001.

16 Tak Wah Lam and Fung Ling Yue. Edge ranking of graphs is hard. *Discrete Applied Mathematics*, 85(1):71–86, 1998.

17 Tak Wah Lam and Fung Ling Yue. Optimal edge ranking of trees in linear time. *Algorithmica*, 30(1):12–33, 2001.

18 Nathan Linial and Michael E. Saks. Searching ordered structures. *J. Algorithms*, 6(1):86–103, 1985.

19 Shay Mozes, Krzysztof Onak, and Oren Weimann. Finding an optimal tree searching strategy in linear time. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 1096–1105, 2008.

20 Nils J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill Pub. Co., 1971.

21 Robert Nowak. Noisy generalized binary search. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1366–1374. Curran Associates, Inc., 2009.

22 Krzysztof Onak and Pawel Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 379–388, 2006.

23 Judea Pearl. *Heuristics - intelligent search strategies for computer problem solving*. Addison-Wesley series in artificial intelligence. Addison-Wesley, 1984.

24 Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.

25 Alfred Renyi. On a problem in information theory. *Magyar Tud. Akad. Mat. Kutato Int. Kozl*, 6(B):505–516, 1961.

26 Alejandro A. Sch"affer. Optimal node ranking of trees in linear time. *Information Processing Letters*, 33(2):91–96, 1989.

27 Stanislaw Ulam. *Adventures of a Mathematician*. University of California Press, 1991.

A Formal Semantics of Influence in Bayesian Reasoning

Bart Jacobs[1] **and Fabio Zanasi**[2]

1 **Radboud Universiteit, Nijmegen, The Netherlands**
2 **University College London, London, United Kingdom**

Abstract

This paper proposes a formal definition of influence in Bayesian reasoning, based on the notions of state (as probability distribution), predicate, validity and conditioning. Our approach highlights how conditioning a joint entwined/entangled state with a predicate on one of its components has 'crossover' influence on the other components. We use the total variation metric on probability distributions to quantitatively measure such influence. These insights are applied to give a rigorous explanation of the fundamental concept of d-separation in Bayesian networks.

1998 ACM Subject Classification Probabilistic computation, F. 1.2

Keywords and phrases probability distribution, Bayesian network, influence

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.21

1 Introduction

A key feature of Bayesian (probabilistic) reasoning is that an observation leads to an update of knowledge. This is best seen in Bayesian networks: in these graph-like models, dependency relations between events are visually depicted as arcs between nodes. Information about a node-event A will update knowledge of all the nodes connected by an arc to A. However, influence may act also in more indirect ways, classified by Pearl [13] as the following "d-separation" scenarios:

- **(i)** in a **serial** connection $\boxed{\text{A}} \to \boxed{\text{B}} \to \boxed{\text{C}}$, event A influences C through B (and viceversa), but knowledge of B "blocks" this mutual influence – one also says that B *d-separates* A and C.
- **(ii)** in a **fork** connection $\boxed{\text{A}} \leftarrow \boxed{\text{B}} \to \boxed{\text{C}}$, information on A will influence C and viceversa, but this flow is blocked once B is known.
- **(iii)** in a **collider** situation $\boxed{\text{A}} \to \boxed{\text{B}} \leftarrow \boxed{\text{C}}$, any evidence about B (and its descendants) will make A and C depend on each other.

In these three scenarios one may observe many phenomena at work which are usually explained informally in terms of influence, dependence, blocking and evidence. But what is the formal semantics underpinning these concepts? The basic language of conditional probability, based on the reading of $\Pr(A|B)$ as "the probability of A given B", appears to be unsuitable for such an account. For instance, it cannot express that, in the collider situation, *any* evidence on the occurrence of B will make A and C dependent, whereas the blocking of the first two scenarios only occurs when B is known with certainty (probability 1).

This paper proposes a rigorous formal treatment of influence in Bayesian reasoning, yielding an expressive and firmly established language for describing the above scenarios. Our methodology draws inspiration from the area of programming language semantics, and in particular from *Effectus theory* [4, 2], a comprehensive logical framework for probabilistic and quantum computation. At the foundation of our approach there is a conceptual distinction

between the knowledge of an event, called a *state*, and an observation/evidence of such event, called a *predicate*. Concretely, a state on a 'sample' space X will be a (finite) discrete probability distributions ω on X, whereas a (fuzzy) predicate p on X is a function $X \to [0,1]$. The 'knowledge update' is then given by a conditioned distribution, which we write as $\omega|_p$, pronounced as: ω given p. Moreover, our approach includes predicate and state transformers, adding expressive power to the language.

Our first contribution (§ 3) is a semantic description of d-separation in the serial (i) and fork connection (ii): we reduce these scenarios into formal statements, whose proofs are made straightforward by our formalism. Here the phenomenon at stake is influence *blocking*, for which the basic language of states and predicates suffices. However, the collider scenario (iii), in which influence is not blocked but rather *enabled*, demands a deeper analysis.

This leads to our second contribution (§ 4), namely the concept of *state entwinedness*. Intuitively, for a joint state/distribution being entwined means, by analogy with the quantum world, that its components are entangled or, in the Bayesian jargon, they model dependent events. In order to capture the collider situation, the key observation is that the join (tensor product) of non-entwined states (say, in (iii), the join of A and C) may become entwined after conditioning (information about B); from that moment on, any new information on one component of the joint state will have influence also on the other component.

As a third contribution (§ 5), we introduce a formal, quantitative definition of such influence: we call it *crossover influence*, as it measures the non-local action between components of a joint states. We also define a notion of *direct influence*, which measures the local action of a predicate (an information update) on a certain state. Both definitions take as a parameter a notion of 'distance' between states: for our scenarios we pick the *total variation* metric on probability distributions, which coincides with the Kantorovich metric [17] on discrete metric spaces (sets). We make no claim on total variation being 'canonical' in the sense of [10]. Our emphasis is rather on the abstract definition of influence: this is independent of the choice of the underlying metric, which is not itself an essential part of our analysis, see also § 7. As far as we know, probabilistic influence has not been formalised and investigated in this quantitative form before.

We conclude our developments with a reprise of the collider scenario (§ 6), which we are now able to adequately describe using the toolkit introduced in § 4 and § 5. Our analysis clarifies that the commonly used description in the literature (see *e.g.* [14, 8, 15]) for describing the serial (i) and fork (ii) scenarios only works for very special 'singleton' predicates – which we call Dirac predicates, whereas in the collider scenario (iii) *any* predicate on B creates dependence (entwinedness) between A and C.

2 Background: states, predicates, and conditional probability

In this background section we introduce the notation, terminology and basic definitions for several constructions in (finite) discrete probability. There is a categorical formalisation using monads behind this, see e.g. [5], but we prefer to keep constructions more concrete.

States, predicates and validity. A (finite, discrete) distribution over a 'sample' set A is a weighted combination of elements of A, where weights are probabilities from the unit interval $[0,1]$ that add up to 1. We call such a distribution a *state*, as it expresses knowledge the occurrence of elements of A. As mentioned in §1, we pursue an analogy with quantum states, emphasised by the use of the 'ket' notation: a state ω is written as $\omega = r_1 \,|a_1\rangle + \cdots + r_n \,|a_n\rangle$, where $a_i \in A$, $r_i \in [0,1]$ and $\sum_i r_i = 1$. Also, $\mathcal{D}(A)$ is the set of states/distributions on

A. We will sometimes treat $\omega \in \mathcal{D}(A)$ equivalently as a function $\omega\colon A \to [0,1]$ with finite support $\mathrm{supp}(\omega) = \{a \in A \mid \omega(a) \neq 0\}$ and with $\sum_{a\in A} \omega(a) = 1$.

An *event* is a subset $E \subseteq A$ of the sample space. We prefer to use a more general 'fuzzy' kind of predicate, namely functions $p\colon A \to [0,1]$. In this discrete case, states (distributions) are predicates, but not the other way around. Events can be identified with 'sharp' predicates taking values in the subset of booleans $\{0,1\} \subseteq [0,1]$. For $x \in A$, we write ∂_x for the (sharp) *Dirac* predicate over x, defined as $\partial_x(a) = 1$ if $x = a$ and $\partial_x(a) = 0$ otherwise.

For predicates $p, q \in [0,1]^A$ and scalar $r \in [0,1]$ we define $p \,\&\, q$ as $a \mapsto p(a) \cdot q(a)$ and $r \cdot p$ as $a \mapsto r \cdot p(a)$. States and predicates are most effectively reasoned about using the language of *Kleisli categories*. We call a function of shape $f\colon A \to \mathcal{D}(B)$ a 'Kleisli' map from A to B and write its type as $A \kto B$. Kleisli maps can be understood as *channels*, or as *stochastic matrices*, especially when A, B are finite sets. The (Kleisli) composition of maps $f\colon A \kto B$ and $g\colon B \kto C$ is written as $g \bullet f\colon A \kto C$. It is essentially matrix multiplication:

$$\big(g \bullet f\big)(a) = \sum_{c\in C} \big(\textstyle\sum_{b\in B} f(a)(b) \cdot g(b)(c)\big) \big| c \big\rangle. \tag{1}$$

We write $\mathcal{K}\ell(\mathcal{D})$ for the Kleisli category whose objects are sets, and whose arrows from A to B are the Kleisli maps $A \kto B$. The identity map $A \kto A$ in $\mathcal{K}\ell(\mathcal{D})$ is the function $a \mapsto 1\,|a\rangle$. Note that arrows $1 \kto B$ in $\mathcal{K}\ell(\mathcal{D})$ identify elements of $\mathcal{D}(B)$, *i.e.* the states on B, and arrows $B \kto 2$ are elements of $[0,1]^B$, *i.e.* the predicates on B.

Each (ordinary) function $g\colon A \to B$ gives a trivial (diagonal) matrix map $\langle g \rangle\colon A \kto B$ via $\langle g \rangle(a) = 1\,|g(a)\rangle$. Then: $\langle h \rangle \bullet \langle g \rangle = \langle h \circ g \rangle$.

We will see later, in Example 3, how Bayesian networks can be seen as graphs of Kleisli maps in $\mathcal{K}\ell(\mathcal{D})$. For this interpretation, it is of importance that $\mathcal{K}\ell(\mathcal{D})$ forms a monoidal category. The monoidal product $\otimes$ is defined on objects as the cartesian product $\times$ of sets, with tensor unit the one-element set 1. On Kleisli maps $f\colon A \kto X$ and $g\colon B \kto Y$ the map $f \otimes g\colon A \otimes B \kto X \otimes Y$ is defined as $(f \otimes g)(a,b)(x,y) = f(a)(x) \cdot g(b)(y)$.

▶ **Definition 1.** Let $\omega \in \mathcal{D}(A)$ be a state and $p \in [0,1]^A$ be a predicate, both on the same set A. We write $\omega \models p$ for the *validity* or *expected value* of p in state ω. This validity is a number in the unit interval $[0,1]$ defined as:

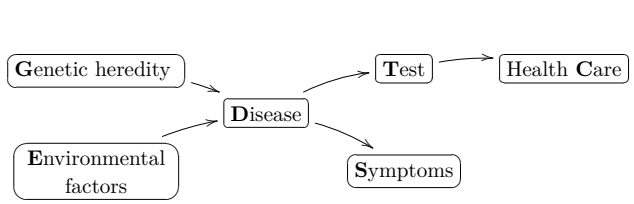
$$\omega \models p \;:=\; \textstyle\sum_{a\in A} \omega(a) \cdot p(a) \;=\; \big(A \overset{p}{\kto} 2\big) \bullet \big(1 \overset{\omega}{\kto} A\big). \tag{2}$$

If this validity is non-zero, it yields a *conditioning* operation on ω. We write $\omega|_p$ or for the conditional state "ω given p", defined as formal convex sum:

$$\omega|_p := \sum_{a\in A} \frac{\omega(a) \cdot p(a)}{\omega \models p} \big| a \big\rangle. \tag{3}$$

▶ **Lemma 2** (From [5]).
(a) $p \,\&\, \partial_x = p(x) \cdot \partial_x$ *and* $\omega \models \partial_x = \omega(x)$ *and* $\omega|_{\partial_x} = 1\,|x\rangle$;
(b) $\omega|_{r\cdot p} = \omega|_p$ *for* $r \neq 0$ *and* $\omega|_{p\&\partial_x} = 1\,|x\rangle$ *when* $p(x) \neq 0$ *and* $\omega(x) \neq 0$;
(c) *Bayes' rule holds for fuzzy predicates:* $\omega|_p \models q = \dfrac{\omega \models p \,\&\, q}{\omega \models p}$.

▶ **Example 3.** As a running example we will use the situation of a disease that can be caused by environmental factors or by genetic heredity. The presence of the disease in a patient will determine whether she manifests symptoms and also whether she tests positively. The test outcome will also influence whether she receives health care. We express these data with a Bayesian network, consisting of a graph together with conditional probability tables.



$\Pr(G) = 1/50$
$\Pr(E) = 1/10$

D	$\Pr(T)$
t	$9/10$
f	$1/20$

G	E	$\Pr(D)$
t	t	$9/10$
t	f	$8/10$
f	t	$4/10$
f	f	0

D	$\Pr(S)$
t	$9/10$
f	$1/15$

T	$\Pr(C)$
t	$4/5$
f	$1/10$

(4)

As illustrated in [7] (*cf.* also [3]), there is a canonical way to interpret our Bayesian network (4) as an arrow in the Kleisli category $\mathcal{KL}(\mathcal{D})$. Each node N of the graph, say with k incoming edges from nodes $N_1, N_2, \ldots, N_k$, is associated with an arrow $N\colon 2^k \nrightarrow 2$ in $\mathcal{KL}(\mathcal{D})$; as a stochastic matrix, N is defined by the probability table of the node N. It will be convenient to write $2_N := \{n, n^\perp\}$ for the two-element target set of the node-arrow N, where n represents occurrence and $n^\perp$ non-occurrence of the event N. For instance, the arrow $D\colon 2_G \otimes 2_E \nrightarrow 2_D$ for the disease node is defined by the channel $2_G \times 2_E \to \mathcal{D}(2_D)$

$$\begin{array}{ll} (g,e) \mapsto \frac{9}{10}\,|d\rangle + \frac{1}{10}\,|d^\perp\rangle & (g,e^\perp) \mapsto \frac{8}{10}\,|d\rangle + \frac{2}{10}\,|d^\perp\rangle \\ (g^\perp,e) \mapsto \frac{4}{10}\,|d\rangle + \frac{6}{10}\,|d^\perp\rangle & (g^\perp,e^\perp) \mapsto 1\,|d^\perp\rangle\,. \end{array}$$

Another example is the initial map $G\colon 1 \nrightarrow 2_G$ for the genetic heredity node, which amounts to the distribution $1/50\,|g\rangle + 49/50\,|g^\perp\rangle$ in $\mathcal{D}(2_G) \cong [0,1]$. In order to recover the whole network (4), one pastes node-arrows together using the monoidal structure of $\mathcal{KL}(\mathcal{D})$. Nodes in (4) that have multiple outgoing edges are modeled by composing the corresponding arrow $2^k \nrightarrow 2$ with the pairing map $\Delta\colon 2 \nrightarrow 2 \otimes 2$ defined by $x \mapsto 1\,|(x,x)\rangle$. The Bayesian network (4) in its entirety is then expressed as the following arrow in $\mathcal{KL}(\mathcal{D})$.

$$1 \xrightarrow{G\otimes E} 2_G \otimes 2_E \xrightarrow{D} 2_D \xrightarrow{\Delta} 2_D \otimes 2_D \xrightarrow{T\otimes S} 2_T \otimes 2_S \xrightarrow{C\otimes \mathrm{id}} 2_C \otimes 2_S \tag{5}$$

Inference via predicate/state transformers. Associated with a Kleisli map $f\colon A \nrightarrow B$ there are *state transformer* and *predicate transformer* maps f_* and f^*. For a state $\omega \in \mathcal{D}(A)$ and a predicate $p \in [0,1]^B$ we define $f_*(\omega) \in \mathcal{D}(B)$ and $f^*(p) \in [0,1]^A$ as:

$$f_*(\omega) = \sum_{b\in B} \big(\textstyle\sum_{a\in A} f(a)(b)\cdot\omega(a)\big)\,|b\rangle \qquad f^*(p)(a) = \sum_{b\in B} f(a)(b)\cdot p(b). \tag{6}$$

Notice that f_* works forwardly, transforming a state on A into a state on B, whereas f^* works backwardly, transforming a predicate on B into a predicate on A. One can understand these definitions in terms of Kleisli composition: $f_*(\omega) = f \bullet \omega$ and $f^*(p) = p \bullet f$. We collect a few basic results from [5].

▶ **Lemma 4.**

(a) *For a Kleisli map $f\colon A \nrightarrow B$, a state $\omega \in \mathcal{D}(A)$ and a predicate $p \in [0,1]^B$, $f_*(\omega) \models p = p \bullet f \bullet \omega = \omega \models f^*(p)$.*

(b) *Predicate transformers f^* preserve $\mathbf{1}$, $\mathbf{0}$, negation $(-)^\perp$ and scalar multiplication $r\cdot(-)$.*

(c) *For an ordinary function $g\colon A \to B$ we have $\langle g\rangle_*(\omega)|_p = \langle g\rangle_*(\omega|_{\langle g\rangle^*(p)})$.*

Using transformers and conditioning one can formulate Bayesian inference (learning). We illustrate the relevant constructions with an example and refer to [7] for more details.

▶ **Example 5** (Backward inference.)**.** A typical learning task wrt. a Bayesian network is backward inference: how the occurrence of a certain event changes the likelihood of its causes. A formalisation of backward inference is proposed in [7] as "predicate transformation followed

by conditioning". We illustrate this for Example 3, focusing on the part of the graph that describes the influence of having the disease on receiving health care. First, we compute our *a priori* knowledge on the likelihood of a disease. In the formalisation (5), this is the Kleisli arrow $D \bullet (G \otimes E) \colon 1 \nrightarrow 2_D$, *i.e.* a state on 2_D.

$$\begin{aligned} G \otimes E &= 0.002\,|g, e\rangle + 0.018\,|g, e^\perp\rangle + 0.098\,|g^\perp, e\rangle + 0.882\,|g^\perp, e^\perp\rangle \\ D \bullet (G \otimes E) &= 0.055\,|d\rangle + 0.945\,|d^\perp\rangle \end{aligned} \tag{7}$$

The event of a positive test is interpreted as the Dirac predicate $\partial_t \in [0,1]^{2_T}$ on 2_T, *i.e.* it maps t to 1 and $t^\perp$ to 0. We can now ask a backward inference question: if the patient tested positive, what is the likelihood that she had the disease? The answer is enclosed in the state $(D \bullet (G \otimes E))|_{T^*(\partial_t)} \colon 1 \nrightarrow 2_D$, obtained by first using $T \colon 2_D \nrightarrow 2_T$ to transform the predicate ∂_t on 2_T into a predicate $T^*(\partial_t)$ on 2_D, and then conditioning the state $D \bullet (G \otimes E)$ over $T^*(\partial_t)$. The latter predicate maps d to $9/10$ and $d^\perp$ to $1/20$. Next,

$$\begin{aligned} D \bullet (G \otimes E) \models T^*(\partial_t) &= 0.097\,|d\rangle + 0.903\,|d^\perp\rangle \\ (D \bullet (G \otimes E))|_{T^*(\partial_t)} &= \textstyle\sum_{x \in 2_D} \frac{D \bullet (G \otimes E)(x) \cdot T^*(\partial_t)(x)}{D \bullet (G \otimes E) \models T^*(\partial_t)}\,|x\rangle \\ &= \tfrac{0.055 \cdot 9/10}{0.097}\,|d\rangle + \tfrac{0.945 \cdot 1/20}{0.097}\,|d^\perp\rangle = 0.51\,|d\rangle + 0.49\,|d^\perp\rangle\,. \end{aligned}$$

Thus evidence of a positive test raises the chances of a disease from 0.055 to 0.51.

Forward inference. A second kind of learning task is *forward inference*: how the occurrence of an event changes the likelihood of its effects. Again following [7], forward inference is formalised as "conditioning and then state transformation". To illustrate this in our leading example, consider a predicate p on 2_G given by $g \mapsto 88\%$ and $g^\perp \mapsto 0.1\%$: it expresses that medical records of a patient show high likelihood of a genetic transmission of the disease. Our forward inference question is: "how does the knowledge update given by predicate p influence the positivity of the test?" For the answer, one first extends p to a (weakened) predicate p' on $2_G \otimes 2_E$, then conditions $G \otimes E$ over p'. Finally, one applies $T \bullet D$ as state transformer to $(G \otimes E)|_{p'}$. Conditioning over p' makes a positive test much more likely:

$$(T \bullet D)_*(G \otimes E) = 0.1\,|t\rangle + 0.9\,|t^\perp\rangle \qquad (T \bullet D)_*((G \otimes E)|_{p'}) = 0.505\,|t\rangle + 0.495\,|t^\perp\rangle\,.$$

3 Influence in d-separation

This section applies the language introduced in § 2 to give a precise explanation of the fundamental concept of 'd-separation' in Bayesian networks, which is used as a criterion for independence, via connections between nodes. These connections can be of three forms, namely 'serial', 'fork', and 'collider'. As we shall see, the language introduced so far is only adapted to describe the first two scenarios. The third scenario needs a richer formalism, which justifies the developments in the next sections.

3.1 Serial connections

$$\boxed{A} \xrightarrow{f} \boxed{B} \xrightarrow{g} \boxed{C} \tag{8}$$

Consider a 'serial connection' Bayesian network as on the right. Clearly, what we know about A influences our knowledge about C, and vice-versa. In the context of d-separation one considers the special cases when there is evidence about the state of B, so that the

mutual influencing between A and B is blocked. We first quote how this is formulated in standard references (names of the nodes in the second quote are adapted to make them consistent with diagram (8)).

(i) [8, §1.2]: Obviously, evidence on A will influence the certainty of B, which then influences the certainty of C. Similarly, evidence on C will influence the certainty on A through B. On the other hand, if the state of B is known, then the channel is blocked, and A and C become independent.

(ii) [14, §1.2.3]: Figuratively, conditioning on B appears to "block" the flow of information along the path, since learning about A has no effect on the probability of C, given B.

These descriptions are rather informal. (i) speaks about (mutual) independence, and (ii) only about having no effect in the forward direction. We will make precise what is going on. Consider the same diagram (8), but now with f, g interpreted as maps in the Kleisli category $\mathcal{K}\ell(\mathcal{D})$ and with predicates as below. The three predicates are inhabitants $p \in [0,1]^A$, $\partial_x \in [0,1]^B$, $q \in [0,1]^C$.

$$\begin{array}{ccccc} A & \xrightarrow{f} & B & \xrightarrow{g} & C \\ {\scriptstyle p}\downarrow & & {\scriptstyle \partial_x}\downarrow & & \downarrow{\scriptstyle q} \\ 2 & & 2 & & 2 \end{array} \tag{9}$$

▶ **Proposition 6** (Blocking I). *Consider the serial connection* (9)*, with Dirac evidence ∂_x on the middle node B, for some fixed $x \in B$. Then there is no influence from A to C, nor from C to A, in the sense that for each distribution/state $\omega \in \mathcal{D}(A)$,*

(a) *for any predicate p on A with $\omega \models p \neq 0$, there is an equality of states on C:*

$$g_*\big(f_*(\omega)|_{\partial_x}\big) = g_*\big(f_*(\omega|_p)|_{\partial_x}\big).$$

(b) *for any predicate q on C there is an equality of states on A:*

$$\omega|_{f^*(\partial_x)} = \omega|_{f^*(\partial_x \& g^*(q))}.$$

We recall how to read the equation in point (a): given a state ω on A, we can transform it to a state $f_*(\omega)$ on B. We can also first condition ω to $\omega|_p$ and then push forward to $f_*(\omega|_p)$ on B. These different states $f_*(\omega)$ and $f_*(\omega|_p)$ become equal when we condition with the Dirac predicate ∂_x, and then push them forward to C via g_*. Thus, the influence of p is 'annihilated' or 'blocked' via the knowledge $x \in B$ used in conditioning with ∂_x.

Proof. For the first point it suffices to prove $f_*(\omega)|_{\partial_x} = f_*(\omega|_p)|_{\partial_x}$. But this equation follows directly from Lemma 2 (a) since both sides are equal to $1\,|x\rangle$.

For the second point we have $f^*(\partial_x \,\&\, g^*(q)) = f^*(g^*(q)(x) \cdot \partial_x)$ by Lemma 2(a), which is then equal to $g^*(q)(x) \cdot f^*(\partial_x)$ by Lemma 4(b). Finally, by Lemma 2(b), $\omega|_{f^*(\partial_x \& g^*(q))} = \omega|_{g^*(q)(x)\cdot f^*(\partial_x)} = \omega|_{f^*(\partial_x)}$. ◀

▶ **Example 7.**

$$\begin{array}{ccccc} 2_D & \xrightarrow{T} & 2_T & \xrightarrow{C} & 2_C \\ {\scriptstyle \omega}\uparrow & & \downarrow{\scriptstyle \partial_t} & & \\ 1 & & 2 & & \end{array} \tag{10}$$

Nodes 'Disease', 'Test' and 'Health Care' in the network of Example 3 form a serial connection, with Kleisli interpretation given by solid arrows as in (10) below. Clearly, new information about the Disease will impact the likelihood of receiving Health Care,

and viceversa, via the intermediate Test node. We examined these phenomena as forward and backward inference in Example 5, following [7]. We now show that, as prescribed by d-separation, mutual influence may be blocked: a positive test will determine the availability of health care, disregarding whether the patient actually has the disease or not. Viceversa, a positive test will nullify any influence of receiving health care on having the disease, as health care is entirely determined by the test outcome. The dotted arrows in (10) describe a state $\omega = \frac{1}{100}\,|d\rangle + \frac{99}{100}\,|d^\perp\rangle$ on 2_D, giving a 1% disease probability, and the Dirac predicate $\partial_t \in [0,1]^{2_T}$, asserting the positivity of the test. For the transformed predicate $T^*(\partial_x)$ on 2_D we have:

$$\begin{cases} T^*(\partial_t)(d) = \frac{9}{10} \\ T^*(\partial_t)(d^\perp) = \frac{1}{20} \end{cases} \qquad \omega \models T^*(\partial_t) = \tfrac{117}{2000} \qquad \omega|_{T^*(\partial_t)} = \tfrac{18}{117}\,|d\rangle + \tfrac{99}{117}\,|d^\perp\rangle\,.$$

The latter distribution $\omega|_{T^*(\partial_t)}$ equals $\omega|_{T^*(\partial_t \& C^*(q))}$ for each predicate $q \in [0,1]^{2_C}$ on 2_C, by Proposition 6 ((b)).

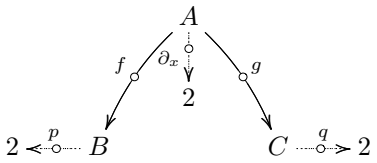
▶ Remark. We emphasise that, if we replace the predicate ∂_t on 2_D by a non-Dirac predicate $p \in [0,1]^{2_D}$, then there is *no blocking*, in general. For instance, take: $p(t) = \frac{1}{3}$, $p(t^\perp) = \frac{1}{4}$, $q(c) = \frac{1}{5}$ and $q(c^\perp) = 1$. Then we compute a difference between the following states on 2_D.

$$\omega|_{T^*(p)} = 0.013\,|d\rangle + 0.987\,|d^\perp\rangle \qquad \omega|_{T^*(p \& c^*(q))} = 0.006\,|d\rangle + 0.994\,|d^\perp\rangle$$

Hence, influence from right to left in (10) does exist for non-sharp predicates.

3.2 Fork connections

Next we consider a "fork" Bayesian network with predicates p, ∂_x, q, for a given element $x \in A$, as below. The informal description of this situation is: influence can pass between the children B and C via A, unless the state of A is known, as formulated *e.g.* in [8].



(11)

▶ **Example 8.** The Bayesian network of Example 3 contains a fork, given by 'Disease', 'Test' and 'Symptoms'. If a patient tests positively, it gets more likely that she has the disease, and thus shows symptoms. However, if one gets to know with certainty that she has the disease, then any evidence about the test will not change the likelihood of showing symptoms.

▶ **Proposition 9 (Blocking II).** *In the fork network* (11)*, with Dirac evidence on the middle node A, there is no influence from B to C, nor from C to B. This lack of influence from B to C is expressed via the equation:*

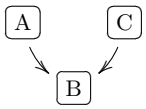
$$g_*\big(\omega|_{\partial_x}\big) = g_*\big(\omega|_{f^*(p)\&\partial_x}\big) \tag{12}$$

for each state ω on A and predicate p on B, and $x \in A$. The other direction is analogous.

Proof. The state transformer g_* is irrelevant, as $\omega|_{\partial_x} = 1\,|x\rangle = \omega|_{f^*(p)\&\partial_x}$. The first equation is in point (a) in Lemma 2, and the second one in point (b). ◀

3.3 Collider connections

The last d-separation scenario is the one of a collider:



which becomes in $\mathcal{KL}(\mathcal{D})$, with the addition of a predicate q,

$$\begin{array}{c} A \otimes C \\ {\scriptstyle f}\downarrow \\ B \xrightarrow{q} 2. \end{array} \tag{13}$$

In [14] one can read about this situation: "if the two extreme variables are (marginally) independent, they will become dependent (*i.e.* connected through unblocked path) once we condition on the middle variable (*i.e.* the common effect) or any of its descendants."

In our formalisms, this explanation unravels as follows. We fix states $\sigma \in \mathcal{D}(A)$ and $\tau \in \mathcal{D}(C)$, giving rise to a product state $\sigma \otimes \tau \in \mathcal{D}(A \otimes C)$. If we have evidence $q\colon B \to 2$ on B, then we can pull it back to evidence $f^*(q)\colon A \otimes C \to 2$. Now, in order to complete our formalisation, we would like to express that σ and τ are initially independent of each other when joint in $\sigma \otimes \tau$, but they get correlated after conditioning $(\sigma \otimes \tau)|_{f^*(q)}$. This correlation should be witnessed by the fact that from now on any predicate on the A-component σ will also have influence on the C-component τ, and viceversa. However, our formalisms of § 2 still lacks the means of expressing such 'crossover' properties, which echo the entanglement phenomena commonly studied in quantum theory. We devote the next two sections to rigorously describe them within our approach, and return to the collider scenario in § 6.

4 Joint states and entwinedness

We now commence the formal investigation of correlation phenomena which will lead to the notion of *crossover* influence. We give an elementary illustration first.

▶ **Example 10.** Consider two diseases A_1 and A_2 which may occur together, as given by the prior joint probability distribution: $\omega = \frac{1}{6}\,|a_1 a_2\rangle + \frac{1}{4}\,|a_1 a_2^\perp\rangle + \frac{1}{3}\,|a_1^\perp a_2\rangle + \frac{1}{4}\,|a_1^\perp a_2^\perp\rangle$. Assume that there is a test for disease A_1 with sensitivity 90% positive when a patient has the disease A_1, and 5% positive when the patient does not. It turns out the prior probability of A_2 is $\frac{1}{2}$, but decreases to $\frac{40}{97}$ after a A_1-positive test. We shall see how this works in Example 15.

For two states/distributions $\sigma \in \mathcal{D}(A_1)$ and $\tau \in \mathcal{D}(A_2)$ we can form the joint 'product' distribution $\sigma \otimes \tau \in \mathcal{D}(A_1 \otimes A_2)$ as $(\sigma \otimes \tau)(a_1, a_2) = \sigma(a_1) \cdot \tau(a_2)$, as already used in (7). The two original states σ and τ can be recovered as marginals of this product state: $\mathsf{M}_1(\sigma \otimes \tau) = \sigma$ and $\mathsf{M}_2(\sigma \otimes \tau) = \tau$. Marginalisation (of states) and weakening (of predicates) are special cases of state and predicate transformation, namely for the (Kleisli) projection maps $\pi_i\colon A_1 \otimes A_2 \to A_i$, given by $\pi_i(a_1, a_2) = 1\,|a_i\rangle$. Marginalisation moves a 'joint' state on a product to one of the components, and weakening moves a predicate on a component to the product. These two operations play a special role in the sequel, and therefore we introduce explicit notation M and W. First, for a joint state $\omega \in \mathcal{D}(A_1 \otimes A_2)$ we have first and second marginalisation $\mathsf{M}_i(\omega) = (\pi_i)_*(\omega) \in \mathcal{D}(A_i)$ determined by (6) as:

$$\mathsf{M}_1(\omega)(a_1) = \textstyle\sum_{a_2 \in A_2} \omega(a_1, a_2) \qquad \mathsf{M}_2(\omega)(a_2) = \textstyle\sum_{a_a \in A_1} \omega(a_1, a_2). \tag{14}$$

Similarly we have weakening operations $\mathsf{W}_i(p_i) = (\pi_i)^*(p_i) \in [0,1]^{A_1 \otimes A_2}$ for predicates $p_i \in [0,1]^{A_i}$ given by:

$$\mathsf{W}_1(p_1)(a_1, a_2) = p_1(a_1) \qquad \mathsf{W}_2(p_2)(a_1, a_2) = p_2(a_2). \tag{15}$$

Also, for two predicates $p_i \in [0,1]^{A_i}$, we introduce their *parallel conjunction* $p_1 \odot p_2 \in [0,1]^{A_1 \times A_2}$, mapping (a_1, a_2) to $p_1(a_1) \cdot p_2(a_2)$. The following definition describes the interaction – dependence, in Bayesian jargon – between the components of a joint state.

▶ **Definition 11.** A joint state $\omega \in \mathcal{D}(A_1 \otimes A_2)$ is called *non-entwined* if it is the product of its marginals: $\omega = \mathsf{M}_1(\omega) \otimes \mathsf{M}_2(\omega)$. It is called *entwined* otherwise.

▶ **Lemma 12.**
(a) $\mathsf{M}_1(\omega) \models p \;=\; \omega \models \mathsf{W}_1(p)$ *and* $\mathsf{M}_2(\omega) \models p \;=\; \omega \models \mathsf{W}_2(p)$.
(b) $\mathsf{W}_1(p) = p \odot \mathbf{1}$ *and* $\mathsf{W}_2(q) = \mathbf{1} \odot q$ *and* $p \odot q = \mathsf{W}_1(p) \,\&\, \mathsf{W}_2(q)$.
(c) $(\sigma \otimes \tau) \models (p \odot q) = (\sigma \models p) \cdot (\tau \models q)$ *and* $(\sigma \otimes \tau)|_{p \odot q} = (\sigma|_p) \otimes (\tau|_q)$.

The next result plays an important role in the sequel. The first equation below says that if one takes the marginal of a joint state conditioned with a weakened predicate, then one may as well condition the marginal directly. This holds if the weakening and marginalisation use the same component. But it fails if the components are different, see the subsequent inequality $\neq$ below. The latter fact is remarkable, because it involves a form of influence between components. This is also called 'signalling' in the quantum world, but apparently already appears in the current probabilistic setting – only for entwinted states.

▶ **Proposition 13.** *Let $p \in [0,1]^A$ be a predicate on a set A.*
(a) *For an arbitrary joint state $\omega \in \mathcal{D}(A \otimes B)$,*

$$\mathsf{M}_1\big(\omega|_{\mathsf{W}_1(p)}\big) \;=\; \mathsf{M}_1(\omega)|_p \qquad \textit{but in general} \qquad \mathsf{M}_2\big(\omega|_{\mathsf{W}_1(p)}\big) \;\neq\; \mathsf{M}_2(\omega).$$

(b) *For the special case of a (non-entwined) product state $\sigma \otimes \tau \in \mathcal{D}(A \otimes B)$,*

$$\mathsf{M}_1\big((\sigma \otimes \tau)|_{\mathsf{W}_1(p)}\big) \;=\; \sigma|_p \qquad \mathsf{M}_2\big((\sigma \otimes \tau)|_{\mathsf{W}_1(p)}\big) \;=\; \tau.$$

Proof. We only prove the equality in point (a), and refer to Example 14 (b) for the inequality in point (b), where a counterexample is given.

$$\begin{aligned}
\mathsf{M}_1\big(\omega|_{\mathsf{W}_1(p)}\big)(a) &\overset{(14)}{=} \textstyle\sum_b \omega|_{\mathsf{W}_1(p)}(a,b) \overset{(3)}{=} \sum_b \frac{\omega(a,b) \cdot \mathsf{W}_1(p)(a,b)}{\omega \models \mathsf{W}_1(p)} \overset{(15)}{=} \frac{\sum_b \omega(a,b) \cdot p(a)}{\omega \models \mathsf{W}_1(p)} \\
&\overset{\text{Lem.12(a)}}{=} \frac{\mathsf{M}_1(\omega)(a) \cdot p(a)}{\mathsf{M}_1(\omega) \models p} \\
&\overset{(3)}{=} \mathsf{M}_1(\omega)|_p(a). \qquad \blacktriangleleft
\end{aligned}$$

We illustrate two significant related phenomena via an example.

▶ **Example 14.** Given sets $X = \{x, y\}$ and $A = \{a, b\}$, one can prove that a state $\omega = r_1 \,|x,a\rangle + r_2 \,|x,b\rangle + r_3 \,|y,a\rangle + r_4 \,|y,b\rangle \in \mathcal{D}(X \otimes A)$, where $r_1 + r_2 + r_3 + r_4 = 1$, is non-entwined if and only if $r_1 \cdot r_4 = r_2 \cdot r_3$. This fact also holds in the quantum case, see *e.g.* [12, §1.5]. It plays a role in the next two illustrations.

(a) **Conditioning creates entwinedness**. Recall from Example 3 the joint state $G \otimes E$ on $2_G \otimes 2_E$, defined as in (7). Consider now a predicate $p \in [0,1]^{2_D}$ defined by $d \mapsto 85\%$ and $d^\perp \mapsto 20\%$. It models, for instance, the information that pallor appears as a symptom in 85% of patients with the disease, but also healthy patients may be pale for other reasons, 20% of the times. Using D as a predicate transformer, we can form the conditioned state $\omega = (G \otimes E)|_{D^*(p)} = 0.007 \,|g,e\rangle + 0.055 \,|g,e^\perp\rangle + 0.191 \,|g^\perp,e\rangle + 0.747 \,|g^\perp,e^\perp\rangle$. This state is now entwined, see the above characterisation of non-entwinedness.

(b) Influence between marginals of entwined states. Let's now start with an entwined state $\sigma = \frac{1}{3}\,|g,e\rangle + \frac{1}{4}\,|g,e^\perp\rangle + \frac{1}{6}\,|g^\perp,e\rangle + \frac{1}{4}\,|g^\perp,e^\perp\rangle \in \mathcal{D}(2_G \otimes 2_E)$ and a predicate $q = \partial_g \in [0,1]^{2_G}$. By weakening we get $\mathsf{W}_1(q) = q \bullet \pi_1 \in [0,1]^{2_G \otimes 2_E}$. Then: $\sigma \models \mathsf{W}_1(q) = \frac{1}{3}\cdot 1 + \frac{1}{4}\cdot 1 = \frac{7}{12}$, so that:

$$\sigma|_{\mathsf{W}_1(q)} \;=\; \tfrac{1/3}{7/12}\,|g,e\rangle + \tfrac{1/4}{7/12}\,|g,e^\perp\rangle \;=\; \tfrac{4}{7}\,|g,e\rangle + \tfrac{3}{7}\,|g,e^\perp\rangle\,.$$

Below, the second marginal of the original state σ differs from the second marginal of this conditioned state, illustrating the inequality $\neq$ in Proposition 13 (a).

$$\mathsf{M}_2(\sigma) \;=\; \tfrac{1}{2}\,|e\rangle + \tfrac{1}{2}\,|e^\perp\rangle \qquad \text{whereas} \qquad \mathsf{M}_2\big(\sigma|_{\mathsf{W}_1(q)}\big) \;=\; \tfrac{4}{7}\,|e\rangle + \tfrac{3}{7}\,|e^\perp\rangle\,.$$

▶ **Example 15.** We conclude with the formal description of the two-disease scenario with which we started this section (Example 10). The test is a map $T\colon 2_{A_1} \to 2_T$ given by $T(a_1) = \frac{9}{10}\,|t\rangle + \frac{1}{10}\,|t^\perp\rangle$ and $T(a_1^\perp) = \frac{1}{20}\,|t\rangle + \frac{19}{20}\,|t^\perp\rangle$. The impact of a positive test on the disease A_2 is given by the marginal of the conditional: $\mathsf{M}_2(\omega|_{\mathsf{W}_1(T^*(\partial_t))}) = \frac{40}{97}\,|a_2\rangle + \frac{57}{97}\,|a_2^\perp\rangle$.

5 A quantitative definition of influence

Last section showed how evidence on one component of an entwined state may influence the other component. But *how much* did it change the latter component with respect to our prior belief? This section addresses such aspect by introducing a quantitative semantics for our influence vocabulary. We begin by recalling the total variation metric on distributions.

▶ **Definition 16.** Let $\sigma, \tau \in \mathcal{D}(X)$ be two distributions on a set X. Their *total variation distance* $\mathsf{d}(\sigma,\tau)$ is defined as the following number in the unit interval $[0,1]$.

$$\mathsf{d}(\sigma,\tau) \;=\; \frac{1}{2}\sum_{x\in X} \big|\sigma(x) - \tau(x)\big|.$$

▶ **Lemma 17.** *Let $f\colon X \nrightarrow Y$ be a Kleisli map. The associated state transformer $f_*\colon \mathcal{D}(X) \to \mathcal{D}(Y)$ from (6) is non-expansive: $\mathsf{d}(f_*(\sigma), f_*(\tau)) \leq \mathsf{d}(\sigma,\tau)$. This yields a functor $\mathcal{K}\ell(\mathcal{D}) \to \mathbf{Met}_1$, where $\mathbf{Met}_1$ is the category of 1-bounded metric spaces and non-expansive maps.*

Proof.

$$\begin{aligned}
\mathsf{d}(f_*(\sigma), f_*(\tau)) \;&=\; \tfrac{1}{2}\textstyle\sum_{y\in Y} \big|\, f_*(\sigma)(y) - f_*(\tau)(y) \,\big| \\
&\overset{(6)}{=}\; \tfrac{1}{2}\textstyle\sum_{y\in Y} \Big|\, \sum_{x\in X} f(x)(y)\cdot\sigma(x) - \sum_{x\in X} f(x)(y)\cdot\tau(x) \,\Big| \\
&\leq\; \tfrac{1}{2}\textstyle\sum_{x\in X}\sum_{y\in Y} f(x)(y)\cdot \big|\, \sigma(x) - \tau(x) \,\big| \\
&=\; \tfrac{1}{2}\textstyle\sum_{x\in X} 1\cdot \big|\, \sigma(x) - \tau(x) \,\big| \;=\; \mathsf{d}(\sigma,\tau).
\end{aligned}$$

◀

We refer to [6] for more information about total variation (and Kantorovich) distance and the distribution monad $\mathcal{D}$, and turn to our formal definition of influence. First we define it in direct form, as a number indicating how much a predicate p influences a state σ via conditioning $\sigma|_p$, given by the (total variation) distance between σ and $\sigma|_p$. This seems fairly simple. But, as we have seen in Section 4, there may also be indirect, 'crossover' influence between the components of a joint entwined state: this is the content of our second definition.

▶ **Definition 18.** Let $p \in [0,1]^A$ be a predicate on a set A with discrete metric.

1. For a state $\sigma \in \mathcal{D}(A)$ on A the *direct influence* of p on σ is defined as:

$$\mathcal{I}_d(p,\sigma) \;:=\; \mathsf{d}\big(\sigma, \sigma|_p\big) \qquad \text{provided} \;\; \sigma \models p \neq 0.$$

2. For a joint state $\omega \in \mathcal{D}(A \otimes B)$ the *crossover influence* of p on ω is:

$$\mathcal{I}_c(p, \omega) := \mathsf{d}\big(\mathsf{M}_2(\omega), \mathsf{M}_2(\omega|_{\mathsf{W}_1(p)})\big) \qquad \text{provided } \omega \models \mathsf{W}_1(p) \neq 0.$$

In general we say that a predicate has *no* (direct or crossover) influence on a state if the corresponding influence function ($\mathcal{I}_d$ or $\mathcal{I}_c$) has outcome zero.

▶ **Example 19.** We give an example of *direct* influence, postponing a detailed illustration of *crossover* influence to the collider scenario in Section 6. Recall the Kleisli map (5) modeling the Bayesian network of Example 3. We fix three different states on $2_D = \{d, d^\perp\}$:

$$\omega = \tfrac{4}{5}\,|d\rangle + \tfrac{1}{5}\,|d^\perp\rangle \qquad \rho = \tfrac{1}{2}\,|d\rangle + \tfrac{1}{2}\,|d^\perp\rangle \qquad \sigma = \tfrac{1}{5}\,|d\rangle + \tfrac{4}{5}\,|d^\perp\rangle\,.$$

Intuitively, in state ω it is likely that the patient has the disease, in state σ it is rather unlikely, and ρ sits in the middle. Consider the Dirac predicate $\partial_t \in [0,1]^{2_T}$ expressing positivity of the test: we first use the predicate transformer T^* associated with the Kleilsi map $T\colon 2_D \nrightarrow 2_T$ to obtain a predicate $T^*(\partial_t) \in [0,1]^{2_D}$; subsequently, we compute the influence of $T^*(\partial_t)$ on the above three states. This is done via a script.

$$\mathcal{I}_d\big(T^*(\partial_t), \omega\big) = 0.19 \qquad \mathcal{I}_d\big(T^*(\partial_t), \rho\big) = 0.45 \qquad \mathcal{I}_d\big(T^*(\partial_t), \sigma\big) = 0.62$$

Influence measures how radically the positivity of the test challenges our belief on the disease: a positive test does not come at surprise in state ω, but it is more unexpected in state σ.

▶ **Example 20.** Clearly, $\mathcal{I}_d(\mathbf{1}, \omega) = 0$, for the truth predicate $\mathbf{1}$, since $\omega|_{\mathbf{1}} = \omega$. Is there also an example where the (direct and crossover) influence reaches the maximal distance 1? We show how to approximate it. Take $A = \{a, b\}$ with predicate $p(a) = 1, p(b) = 0$ and state $\sigma = \epsilon\,|a\rangle + (1-\epsilon)\,|b\rangle$. The direct influence $\mathcal{I}_d(p, \sigma)$ goes to 1 as $\epsilon \to 0$. Similarly, by taking $\omega = \epsilon\,|aa\rangle + (1-\epsilon)\,|bb\rangle \in \mathcal{D}(A \times A)$ we get $\mathcal{I}_c(p, \omega) \to 1$ as $\epsilon \to 0$ for crossover influence.

We now establish some facts on crossover influence: (1) it only makes sense if the state is entwined, since for a product state the crossover influence is zero; (2) weakening and marginalisation must work in different components, since otherwise we have direct influence; (3) crossover influences is always less than direct influence. In the context of Definition 18:

▶ **Lemma 21.**

1. $\mathcal{I}_c(p, \sigma \otimes \tau) = 0$;
2. $\mathsf{d}\big(\mathsf{M}_1(\omega), \mathsf{M}_1(\omega|_{\mathsf{W}_1(p)})\big) = \mathcal{I}_d(p, \mathsf{M}_1(\omega))$;
3. $\mathcal{I}_c(p, \omega) \leq \mathcal{I}_d(\mathsf{W}_1(p), \omega)$
4. *For each function* $g\colon X \to Y$, *considered as a Kleisli map* $\langle g \rangle\colon X \to \mathcal{D}(Y)$, *we have:* $\mathcal{I}_d(p, \langle g \rangle_*(\sigma)) \leq \mathcal{I}_d(\langle g \rangle^*(p), \sigma)$, *where* $\sigma \in \mathcal{D}(X)$.

Proof. The first two points follow directly from Proposition 13 (b) and (a). The inequality in point (3) from the fact that marginalisation is a special form of state transformation, which, as we know from Lemma 17, is non-expansive:

$$\begin{aligned}\mathcal{I}_c(p, \omega) = \mathsf{d}\big(\mathsf{M}_2(\omega), \mathsf{M}_2(\omega|_{\mathsf{W}_1(p)})\big) &= \mathsf{d}\big((\pi_2)_*(\omega), (\pi_2)_*(\omega|_{\mathsf{W}_1(p)})\big) \\ &\leq \mathsf{d}\big(\omega, \omega|_{\mathsf{W}_1(p)}\big) = \mathcal{I}_d(\mathsf{W}_1(p), \omega).\end{aligned}$$

Finally, for point (4) we use both Lemma 4 (c) and Lemma 17 in:

$$\begin{aligned}\mathcal{I}_d(p, \langle g \rangle_*(\omega)) &= \mathsf{d}\big(\langle g \rangle_*(\omega), \langle g \rangle_*(\omega)|_p\big) \\ &= \mathsf{d}\big(\langle g \rangle_*(\omega), \langle g \rangle_*(\omega|_{\langle g \rangle^*(p)})\big) \leq \mathsf{d}\big(\omega, \omega|_{\langle g \rangle^*(p)}\big) = \mathcal{I}_d(\langle g \rangle^*(p), \omega).\end{aligned}$$

◀

▶ Remark. Crossover and direct influence are instances of a more general definition of influence of a predicate $p \in [0,1]^{A_j}$ on the i-th marginal A_i of a joint state $\omega \in \mathcal{D}(A_1 \otimes \ldots \otimes A_n)$. For $n = 2$ and $i \neq j$, this corresponds to crossover influence, whereas for $n = i = j = 1$ it would be direct influence. We chose not to work within this uniform approach as we believe that it is more insightful to think of the two notions of Definition 18 as conceptually distinct.

As observed in §3, the blocking action of Dirac predicates plays a key role in d-separation. We can use Definition 18 to express that no predicate p has any influence on a Dirac-conditioned state $\omega|_{\partial_x}$ – by Lemma 2, $(\omega|_{\partial_x})|_p = (\omega|_p)|_{\partial_x} = 1\,|x\rangle = \omega|_{\partial_x}$, so $\mathcal{I}_d(p, \omega|_{\partial_x}) = 0$.

▶ **Example 22.** For instance, we can reformulate the fork scenario as follows. Because conditioning is commutative, (12) is the same as: $\omega|_{\partial_x} = \big(\omega|_{\partial_x}\big)|_{f^*(p)}$. Thus Proposition 9 says that $\mathcal{I}_d(f^*(p), \omega|_{\partial_x}) = 0$, *i.e.* $f^*(p)$ has no influence on $\omega|_{\partial_x}$.

In the same vein, one may also revisit Example 8, an instance of the serial connection scenario: in short, from (5), use states D, $T \bullet D$ and $S \bullet T \bullet D$ to construct a joint state on $2_D \otimes 2_T \otimes 2_S$; check that a 'positive test' predicate $\partial_t \in [0,1]^{2_T}$ has crossover influence on the marginal 2_S, then prove that a 'disease' predicate $\partial_d \in [0,1]^{2_D}$ blocks such influence.

6 Influence in d-separation (reprise)

We conclude with a return on the collider scenario, left unfinished at the end of § 3. With the notation introduced therein, we now explain the collider situation in Diagram (13): the initial joint (product) state $\sigma \otimes \tau$ is non-entwined, but it becomes entwined after conditioning with evidence q on B, as in $(\sigma \otimes \tau)|_{f^*(q)}$. Now any new evidence $p \in [0,1]^A$ on A may have *crossover* influence on C – *cf.* Example 14 (b). It can be explicitly quantified by computing $\mathcal{I}_c\big(p, (\sigma \otimes \tau)|_{f^*(q)}\big)$.

A conceptual insight stemming from our analysis is the asymmetry between blocking and enabling influence: while in the serial and fork scenarios only Dirac predicates are able to block, in a collider *any* predicate may enable. We give a concrete example below.

▶ **Example 23.** The Bayesian network of Example 3 includes a collider, given by nodes 'Genetic Heredity' and 'Environmental Factors' both pointing to 'Disease'. The two possible causes for the disease are represented as a joint state $G \otimes E$ on $2_G \otimes 2_E$, see (7). *A priori*, they are unrelated. For instance, a Dirac predicate $\partial_{g^\perp} \in [0,1]^{2_G}$ that excludes any genetic disorder of the patient has no effect on the chances that she has been exposed to the environmental factors: formally, the crossover influence $\mathcal{I}_c(\partial_{g^\perp}, G \otimes E)$ is 0, as guaranteed by Lemma 21.1. However, let's include the information that pallor is a symptom of the disease, modeled as a predicate $p \in [0,1]^{2_D}$ as in Example 14(a): it turns $G \otimes E$ into an entwined state $(G \otimes E)|_{D^*(p)}$. In this changed scenario, d-separation tells that ruling out genetic heredity (predicate $\partial_{g^\perp}$) *does* influence the belief that environment was the cause. We can formally expressed it with crossover influence:

$$\mathcal{I}_c(\partial_{g^\perp}, G \otimes E) = 0 \qquad \mathcal{I}_c(\partial_{g^\perp}, (G \otimes E)|_{D^*(p)}) = 0.006.$$

Note that a Dirac predicate $\partial_d \in [0,1]^{2_D}$ expressing certainty of the disease entwines G and E much more: indeed $\mathcal{I}_c(\partial_{g^\perp}, (G \otimes E)|_{D^*(\partial_d)}) = 0.26 > \mathcal{I}_c(\partial_{g^\perp}, (G \otimes E)|_{D^*(p)})$.

7 Discussion

Our ambition in this paper was to develop a framework where grounding concepts of Bayesian reasoning (influence, dependence, blocking, evidence, ...) are given a clear, completely formal meaning, building on [7], and can be reasoned about in an abstract and flexible

manner. As a proof of concept, we analysed d-separation: the intention was to show how event interactions with a subtle and potentially ambiguous natural language description can be reduced to elementary formulas of our language, with a simple and transparent proof.

We based our approach on Kleisli categories, in harmony with the increasing importance of algebraic methods from program semantics in the analysis of probabilistic systems [11, 16, 10]. The highlight of our developments is the notion of crossover influence, which we believe may foster research in two directions. First, it draws a parallelism with non-locality phenomena of quantum theory, see also [6]: we plan investigate the meaning of our definitions in that setting, exploiting the formal bridge offered by Effectus theory [4, 2]. Second, our definition is abstract enough to accommodate different choices for the underlying notion of distance between states. The total variation metric suits the applications of this paper, but other choices are also worth investigating: we think in particular of the Kantorovich metric [17], for when the sample set has a non-discrete metric, and quantitative analyses of information leakage [1]. Also connections with Kullback-Leibler divergence [9], focussing on loss of information, in Shannon style, and to mutual information, remain to be investigated.

A related point concerns the relationship between the total variation distance and Bayesian influence. In our choice, we simply aimed at the most basic additive distance which does not (unlike Kantorovich) builds on a pre-existing metric, as our sample sets have none. Admittedly, the suitability of total variation is only empirically justified by examples. In future work we aim at a more satisfactory investigation: recent advances on an axiomatic treatment of metrics [10] appear to be very suitable for the purpose.

References

1 M. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith. Measuring information leakage using generalized gain functions. In *Computer Security Foundations Symposium (CSF 2012)*, pages 265–279, 2012.

2 K. Cho, B. Jacobs, A. Westerbaan, and B. Westerbaan. An introduction to effectus theory. see `arxiv.org/abs/1512.05813`, 2015.

3 B. Fong. Causal theories: A categorical perspective on Bayesian networks. Master's thesis, Univ. of Oxford, 2012. see `arxiv.org/abs/1301.6201`.

4 B. Jacobs. New directions in categorical logic, for classical, probabilistic and quantum logic. *Logical Methods in Comp. Sci.*, 11(3):1–76, 2015.

5 B. Jacobs. From probability monads to commutative effectuses. *Journ. of Logical and Algebraic Methods in Programming*, 156, 2016. See DOI:10.1016/j.jlamp.2016.11.006.

6 B. Jacobs. A note on distances between probabilistic and quantum distributions. In A. Silva, editor, *Math. Found. of Programming Semantics*, Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2017.

7 B. Jacobs and F. Zanasi. A predicate/state transformer semantics for Bayesian learning. In L. Birkedal, editor, *Math. Found. of Programming Semantics*, number 325 in Elect. Notes in Theor. Comp. Sci., pages 185–200. Elsevier, Amsterdam, 2016.

8 F. Jensen. *Bayesian Networks and Decision Graphs*. Statistics for Engineering and Information Science. Springer, 2001.

9 S. Kullback. *Information Theory and Statistics*. John Wiley & Sons, 1959.

10 R. Mardare, P. Panangaden, and G. Plotkin. Quantitative algebraic reasoning. In *Logic in Computer Science, LICS '16*, pages 700–709. IEEE, Computer Science Press, 2016.

11 A. McIver, C. Morgan, and T. Rabehaja. Abstract hidden Markov models: A monadic account of quantitative information flow. In *Logic in Computer Science*, pages 597–608. IEEE, Computer Science Press, 2015.

12 N.D. Mermin. *Quantum Computer Science: An Introduction*. Cambridge Univ. Press, 2007.

13 J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

14 J. Pearl. *Causality. Models, Reasoning, and Inference*. Cambridge Univ. Press, 2nd ed. edition, 2009.

15 S. Russel and P. Norvig. *Artificial Intelligence. A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 2003.

16 S. Staton, H. Yang, C. Heunen, O. Kammar, and F. Wood. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Logic in Computer Science, LICS '16*, pages 525–534, 2016.

17 C. Villani. *Optimal Transport – Old and New*. Springer Berlin Heidelberg, 2009.

The Complexity of SORE-definability Problems

Ping Lu[*1], Zhilin Wu[†2], and Haiming Chen[‡3]

1 **BDBC, Beihang University, Beijing, China**
luping@buaa.edu.cn

2 **State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China**
wuzl@ios.ac.cn

3 **State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China**
chm@ios.ac.cn

Abstract

Single occurrence regular expressions (SORE) are a special kind of deterministic regular expressions, which are extensively used in the schema languages DTD and XSD for XML documents. In this paper, with motivations from the simplification of XML schemas, we consider the *SORE-definability problem*: Given a regular expression, decide whether it has an equivalent SORE. We investigate extensively the complexity of the SORE-definability problem: We consider both (standard) regular expressions and regular expressions with counting, and distinguish between the alphabets of size at least two and unary alphabets. In all cases, we obtain tight complexity bounds. In addition, we consider another variant of this problem, the bounded SORE-definability problem, which is to decide, given a regular expression E and a number M (encoded in unary or binary), whether there is an SORE, which is equivalent to E on the set of words of length at most M. We show that in several cases, there is an exponential decrease in the complexity when switching from the SORE-definability problem to its bounded variant.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Single occurrence regular expressions, Definability, Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.22

1 Introduction

Background. Deterministic regular expressions are a special kind of regular expressions, which are mainly used in DTD and XML Schema [11]. Intuitively, deterministic regular expressions require that when reading from left to right a word in the language, each symbol in the word can directly match the symbol in the expression without knowing the next symbol or the length of the word [35, 1]. For example, $(a+b)a^*$ is deterministic. Since for each word w in $\mathcal{L}((a+b)a^*)$, if the first symbol in w is a, then it matches the first a in $(a+b)a^*$, and the other occurrences of a match the second one. On the other hand, $a^*(a+b)$ is not deterministic. Because given a word $w = aa$, without knowing the length of the word, we do not know whether the first a in w should match the first a in $a^*(a+b)$ or the second one.

* Ping Lu is supported by the NSFC grant under No. 61472405. Ping Lu is also partially funded by Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University.
† Zhilin Wu is supported by the the NSFC grants under No. 61472474 and 61572478.
‡ Haiming Chen is supported by the NSFC grant under No. 61472405.

Although deterministic regular expressions ensure the effective processing of XML documents [11, 16], it is not an easy task to design schema for XML documents with deterministic regular expressions. The major obstacle is that they are defined in a semantic manner, and do not have syntax rules to guide the design [1]. Considerable efforts have been made to solve this problem [2, 3, 4, 13, 23, 21, 7]. Among these is the introduction of single occurrence regular expressions (SORE) [2, 3, 4, 13]. SORE are regular expressions where each alphabet symbol appears at most once [2]. For example, $E = abc$ is an SORE, since there is only one of a, b, and c in E. While $E = aa^*b$ is not, because the symbol a appears twice in E.

Motivation. XML schemas defined with SORE are desirable, since they are very simple and intuitive to comprehend, and it is easy to check the conformance of XML documents with respect to them. As a result, though SORE constitute a restricted class of regular expressions, it turns out that deterministic regular expressions used in DTDs of XML documents from the practice are mostly SORE [29]. In view of this, XML schema designers may tempt to know whether the schema they proposed can in fact be changed into an equivalent, but simpler, schema defined with SORE. This brings the *SORE-definability problem*: *Given a regular expression, decide whether there is an SORE defining the same language.*

In this paper, we start an extensive investigation on the complexity of the SORE-definability problem: We consider both (standard) regular expressions and regular expressions with counting, and distinguish between the alphabets of size at least two and unary alphabets. In all cases, we obtain tight complexity bounds.

In addition, we consider another variant of the SORE-definability problem, the bounded SORE-definability problem, which is to decide, given a regular expression E (without or with counting) and a number M (encoded in unary or binary), whether there is an SORE, which is equivalent to E on the set of words of length at most M. The motivation of this problem comes from the observation that in practice, there may exist some additional information about the number of children that a node in XML documents can have, and this information can be utilised to ease the design of XML schemas. Given a node n, suppose that its content model (the format of its child elements) is defined by a regular expression E. Although $\mathcal{L}(E)$, the language defined by E, might be not deterministic (thus not in SORE), if there is a bound M on the number of children of n, then we may still construct a SORE E_1 such that E and E_1 are equivalent over the set of words up to length M. In this way, we can obtain a regular expression E_1 with a simpler structure to define its content model. Moreover, E_1 covers all the possible XML documents. Checking the existence of SORE E_1 such that $E =_{<=M} E_1$ is the bounded-definability problem. In addition, the bounded-definability problem is related to the bounded nonuniversality problem investigated in [10]. The bounded nonuniversality problem is to decide for a given nondeterministic finite-state automaton A and a number M encoded in unary, whether $\mathcal{L}(A) \cap \Sigma^{\leq M} \neq \Sigma^{\leq M}$. Note that this problem can be rephrased as $\mathcal{L}(A) \neq_{\leq M} \Sigma^*$, where $\Sigma = \{a_1, \ldots, a_n\}$ and Σ^* is the abbreviation of $(a_1 + \ldots + a_n)^*$, which is obviously an SORE.

Contributions. The results obtained in this work are summarised in Table 1. In the table, R stands for the set of (standard) regular expressions, and $R(\#)$ stands for the set of regular expressions with counting. We highlight some of them below.

(1) We first show that the SORE-definability problem is **PSPACE**-complete for regular expressions, and **EXPSPACE**-complete for regular expressions with counting.

(2) We then consider the special case of unary alphabets. We show that the SORE-definability problem becomes **coNP**-complete for regular expressions and $\mathbf{\Pi_2^p}$-complete for regular

Table 1 The results of this paper: An overview.

The SORE-definability problem			
		$\|\Sigma\| = 1$	$\|\Sigma\| \geq 2$
R		**coNP**-c (Thm 6)	**PSPACE**-c (Thm 4)
$R(\#)$		$\mathbf{\Pi_2^P}$-c (Thm 7)	**EXPSPACE**-c (Thm 5)
The bounded SORE-definability problem			
R	M is unary	**PTIME** (Thm 15)	**coNP**-c (Cor 14)
	M is binary	**coNP**-c (Cor 19)	**PSPACE**-c (Cor 17)
$R(\#)$	M is unary	**PTIME** (Thm 15)	**coNP**-c (Thm 13)
	M is binary	$\mathbf{\Pi_2^P}$-c (Cor 18)	**coNEXPTIME**-c (Thm 16)

expressions with counting. Moreover, by using the same idea of the lower bound proof, we can show that the definability problem of deterministic regular expressions is $\mathbf{\Pi_2^P}$-complete for regular expressions with counting, which solves an open problem in [27].

(3) For the bounded SORE-definability problem, a bit surprisingly, we show that the complexity is **coNP**-complete for both regular expressions and regular expressions with counting, if the length bound M is encoded in unary. On the other hand, if M is encoded in binary, the complexity is **PSPACE**-complete for regular expressions and **coNEXPTIME**-complete for regular expressions with counting. In addition, when unary alphabets are considered and M is encoded in unary, the bounded SORE-definability problem can be solved in *polynomial time*, even for regular expressions with counting. These results show that if the length bound M is encoded in unary, there is an exponential decrease when switching from the SORE-definability problem to its bounded variant.

Related work. Single occurrence regular expressions were introduced in [2, 3, 4]. Most works on SORE focus on inferring an SORE from a set of sample words. The basic idea of these works is that given a set of sample words, we first construct a single occurrence automaton (SOA) A, then derive an SORE E from A. The main technical difficulty is how to construct E from A. To this end, Bex et al. [3, 4] developed an $O(|A|^5)$ algorithm. Recently, Freydenberger et al. [13] reduced the complexity of the construction to linear time.

There are also works investigating how to automatically construct deterministic regular expressions from (non-deterministic) regular expressions given by users. This problem entails the definability problem of deterministic regular expressions: Given a regular expression, decide whether there exists an equivalent deterministic regular expression. It was shown in [1, 12, 26] that this problem is **PSPACE**-complete. For regular expressions with counting, there are two notations of determinism, *i.e.,* weak and strong determinism (see [14]). The definability problem of weakly (resp. strongly) deterministic regular expressions with counting can also be defined similarly. It was shown in [24] that the definability problem of weakly deterministic regular expressions with counting is in **2-EXPSPACE** when the inputs are regular expressions, and in **3-EXPSPACE** when the inputs are regular expressions with counting, whereas the exact complexity of these problems are still open.

Researchers also investigated how to decide whether a given regular expression is deterministic [5, 6, 23, 21, 7, 16] (Note that this problem is different from the definability problem of deterministic regular expressions in the sense that we do not check the existence of an equivalent albeit potentially different deterministic regular expression). Remarkably, it

was shown in [16] that the problem of whether a regular expression is deterministic can be decided in linear time. In addition, in [16], it was also shown that the problem of whether a regular expression with counting is weakly deterministic can be decided with the same complexity bound. On the other hand, the work [8] provided a linear time algorithm to decide whether a regular expression with counting is strongly deterministic. Moreover, in [18, 19], efficient matching algorithms were provided for strongly deterministic regular expressions with counting by using finite automata with counters.

Outline. This paper is structured as follows. Section 2 fixes some notations. Section 3 is devoted to the SORE-definability problem. The bounded SORE-definability problem is investigated in Section 4. We conclude this paper in Section 5.

2 Preliminaries

For a natural number $n \in \mathbb{N}$, let $[n]$ denote $\{1, \cdots, n\}$. In addition, for two natural numbers $m, n \in \mathbb{N}$ such that $m \leq n$, let $[m, n]$ denote the set $\{m, m+1, \cdots, n\}$.

An *alphabet* Σ is a finite set of symbols $\{a_1, a_2, \ldots, a_n\}$. We will use $a, b, \cdots$ to denote symbols from Σ. A *word* over Σ is a sequence of symbols from Σ. We will use $u, v, w, \cdots$ to denote words and ε to denote the empty word. A *language* over Σ is a set of words on Σ. For two languages L_1 and L_2, we use $L_1 \cdot L_2$ to denote the language $\{uv \mid u \in L_1, v \in L_2\}$. In addition, for a language L, we define $L^0 = \{\varepsilon\}$, and $L^{n+1} = L \cdot L^n$ for each natural number n. We also use L^* to denote $\bigcup_{n \in \mathbb{N}} L^n$. A (standard) *regular expression* over Σ is inductively defined as follows: ε and a are regular expressions for any $a \in \Sigma$; for any regular expressions E_1 and E_2, the disjunction $E_1 + E_2$, the concatenation $E_1 \cdot E_2$ (or $E_1 E_2$), and the star E_1^* are also regular expressions. The semantics of a regular expression E is given by a language $\mathcal{L}(E)$ defined as follows: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$, $\mathcal{L}(a) = \{a\}$, $\mathcal{L}(E_1 + E_2) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2)$, $\mathcal{L}(E_1 \cdot E_2) = \mathcal{L}(E_1) \cdot \mathcal{L}(E_2)$, and $\mathcal{L}(E_1^*) = (\mathcal{L}(E_1))^*$. Let R denote the set of all regular expressions. For a regular expression E, let Σ_E denote the set of all symbols from Σ that appear in E.

Next, we define deterministic regular expressions. Before that, we introduce some notations. Given a regular expression E, we replace every symbol a in E by a subscripted symbol a_i such that a_i appears only once. Let $\overline{E}$ denote the resulting expression obtained by this replacement. For instance, let $E = (\varepsilon + a) \cdot a^*$, then $\overline{E} = (\varepsilon + a_1) \cdot a_2^*$. A regular expression E is *deterministic* iff the following condition holds: for every two words $uxw, uyv \in \mathcal{L}(\overline{E})$, if $x = a_i$ and $y = a_j$, then $i = j$. A regular language L is deterministic, if there exists a deterministic regular expression E such that $\mathcal{L}(E) = L$. For $E = (\varepsilon + a) \cdot a^*$, consider two words $a_1 a_2, a_2 \in \mathcal{L}(\overline{E})$. Let $u = \varepsilon$, $x = a_1$, $w = a_2$, $y = a_2$, and $v = \varepsilon$, then $uxw, uyv \in \mathcal{L}(\overline{E})$, $x = a_1$, $y = a_2$, but $1 \neq 2$. By the definition, $(\varepsilon + a) \cdot a^*$ is not deterministic. But the language $\mathcal{L}((\varepsilon + a) \cdot a^*)$ is deterministic, since $\mathcal{L}((\varepsilon + a) \cdot a^*) = \mathcal{L}(a^*)$ and a^* is deterministic.

Next, we define the Glushkov automata of regular expressions [15, 30]. Given a regular expression E, we first define the following sets: $first(E) = \{a \mid aw_1 \in \mathcal{L}(E)\}$, $follow(E, a) = \{b \mid w_1 a b w_2 \in \mathcal{L}(E)\}$, and $last(E) = \{a \mid w_1 a \in \mathcal{L}(E)\}$. Intuitively, $first(E)$ comprises the first symbols of words in $\mathcal{L}(E)$, $follow(E, a)$ is the set of symbols, which can appear immediately after a in a word from $\mathcal{L}(E)$, and $last(E)$ contains the last symbols of words in $\mathcal{L}(E)$. Then the Glushkov automaton of E, denoted by $G_E = (Q_E \cup \{q_I\}, \Sigma, \delta_E, q_I, F_E)$, is constructed as follows:

1. Q_E is the set of symbols in $\overline{E}$, and q_I is the initial state;
2. For any $a \in \Sigma_E$, let $\delta_E(q_I, a) = \{a_i \mid a_i \in first(\overline{E})\}$;

3. For any $a, b \in \Sigma_E$, let $\delta_E(a_i, b) = \{b_j \mid b_j \in follow(\overline{E}, a_i)\}$;
4. $F_E = \begin{cases} \{a_i \mid a_i \in last(\overline{E})\} \cup \{q_I\} & \text{if } \varepsilon \in L(E), \\ \{a_i \mid a_i \in last(\overline{E})\} & \text{otherwise.} \end{cases}$

Given a regular expression E, the Glushkov automaton G_E can be constructed in polynomial time [6]. See Example 1 in the next section for an example of Glushkov automata.

Single occurrence regular expressions (SORE)[3, 4] are a special kind of deterministic regular expressions, which require that every symbol in the alphabet appears at most once. Moreover, SORE use the following operators: the disjunction $(+)$, the concatenation $(\cdot)$, the iteration $(^+)$, and the optional $(?)$, where the iteration E^+ and the optional $E?$ are the abbreviations of $E \cdot E^*$ and $\varepsilon + E$, respectively. Since $\mathcal{L}(E^*) = \mathcal{L}((E^+)?)$, SORE do not use the star operation. For example, a^+bc is an SORE, but aa^+ is not, since the symbol a appears twice. Additionally, we require that the iteration (resp. the optional) cannot be nested in an SORE, that is, the expressions of the form $(E?)?$ or $(E^+)^+$ are forbidden. We also forbid the expressions of the form $((E^+)?)^+$ or $((E?)^+)?$. These constraints ensure that for a fixed alphabet Σ, there are only a fixed number of SORE over Σ. Nevertheless, these constraints do not affect the expressive power of SORE. For an SORE, its Glushkov automaton can also be constructed in polynomial time [2].

A single occurrence automaton (SOA) $S = (Q, \Sigma, \delta, q_I, F)$ over an alphabet Σ is defined as follows [13]: $Q \subseteq \Sigma \cup \{q_I\}$, where q_I is the initial state; $\delta : Q \times \Sigma \to Q$ is the transition function such that whenever $\delta(q_1, b) = q_2$, we have $q_2 = b$; and $F \subseteq Q$ is the set of final states. Although SOA defined here are slightly different from those in [13], one can easily add a sink state to each SOA defined in this paper to obtain an SOA in [13]. Later on, we will ignore this difference and apply the algorithms in [13] directly on SOA in this paper.

A regular expression with counting is an extension of regular expressions, which additionally allows using the counting modalities $E^{[m,n]}$ or $E^{[m,\infty]}$. For a regular expression with counting E, the language $\mathcal{L}(E^{[m,n]}) = \bigcup_{i \in [m,n]} (\mathcal{L}(E))^i$. The language $\mathcal{L}(E^{[m,\infty]})$ can be defined similarly. Let $R(\#)$ denote the set of regular expressions with counting.

3 The SORE-definability problem

In this section, we study the complexity of the SORE-definability problem: Given a regular expression E, decide whether there exists an SORE E^c such that $\mathcal{L}(E^c) = \mathcal{L}(E)$. We first consider the general case of non-unary alphabets, then the special case of unary alphabets.

3.1 Non-unary alphabets

We start with regular expressions and consider regular expressions with counting later on[1].

To solve the SORE-definability problem, our main idea is to construct for a regular expression E, an SORE E^c such that $\mathcal{L}(E) = \mathcal{L}(E^c)$ iff there exists an SORE E_1 satisfying that $\mathcal{L}(E) = \mathcal{L}(E_1)$. With such an SORE E^c, we can solve the SORE-definability problem by checking whether $\mathcal{L}(E) = \mathcal{L}(E^c)$.

The construction of the SORE E^c is divided into two steps: We first construct an SOA S_E from E, then an SORE E^c from S_E.

Given a regular expression E, let $G_E = (Q_E \cup \{q_I\}, \Sigma, \delta_E, q_I, F_E)$ be the Glushkov automaton of E, we construct the SOA $S_E = (\Sigma_E \cup \{q_I\}, \Sigma_E, \delta'_E, q_I, F'_E)$ as follows:

[1] Several results in this section are based on Chapter 7 of the PhD. thesis of Ping Lu (cf. [25]).

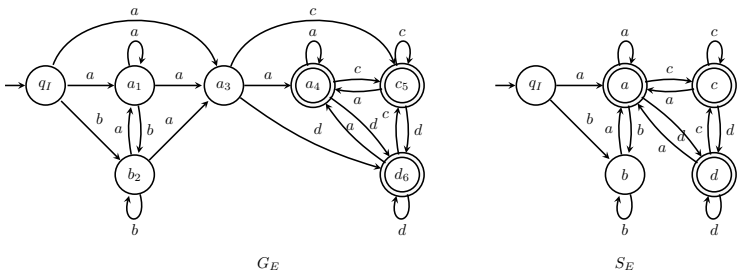


Figure 1 G_E and S_E.

1. For any $a \in \Sigma_E$, let $\delta'_E(q_I, a) = \{a \mid \exists i.\ a_i \in \delta_E(q_I, a)\}$;
2. For any $a, b \in \Sigma_E$, let $\delta'_E(a, b) = \{b \mid \exists i, j.\ b_j \in \delta_E(a_i, b)\}$;
3. $F'_E = \begin{cases} \{a \mid \exists i.\ a_i \in F_E\} \cup \{q_I\} & \text{if } q_I \in F_E, \\ \{a \mid \exists i.\ a_i \in F_E\} & \text{otherwise.} \end{cases}$

Intuitively, the SOA S_E is constructed from G_E by merging for each $a \in \Sigma_E$, all the states a_i of G_E into one state a and modifying the transition relation correspondingly.

▶ **Example 1.** Let $E = (a + b)^* a(a + c + d)^*$. Then $\overline{E} = (a_1 + b_2)^* a_3 (a_4 + c_5 + d_6)^*$. The Glushkov automaton G_E and the SOA S_E constructed from G_E are given in Figure 1. Note that the states a_1, a_3, and a_4 in G_E are merged into one state a in S_E, and the other states remain the same (modulo names).

The SOA S_E enjoys the following property.

▶ **Lemma 2.** *Given a regular expression E, $\mathcal{L}(E)$ can be defined by an SOA iff $\mathcal{L}(G_E) = \mathcal{L}(S_E)$.*

To construct the desired SORE E^c, we apply the algorithm REWRITE in [3, 4] or Soa2Sore in [13] to S_E, and get an SORE E^c enjoying the following property: S_E can be represented by an SORE iff $\mathcal{L}(S_E) = \mathcal{L}(E^c)$. From Lemma 2, we deduce the following fact.

▶ **Proposition 3.** *$\mathcal{L}(E)$ can be defined by an SORE iff $\mathcal{L}(E) = \mathcal{L}(E^c)$.*

The arguments for Proposition 3 proceed as follows: The "if" direction is trivial. For the "only if" direction, suppose that $\mathcal{L}(E)$ is defined by an SORE E_1. Then it can also be defined by an SOA, since the Glushkov automaton of E_1 is an SOA. By Lemma 2, we must have that $\mathcal{L}(E_1) = \mathcal{L}(E) = \mathcal{L}(G_E) = \mathcal{L}(S_E)$. Hence, S_E is represented by the SORE E_1. From the property of E^c, we know that $\mathcal{L}(S_E) = \mathcal{L}(E^c)$. We conclude that $\mathcal{L}(E) = \mathcal{L}(S_E) = \mathcal{L}(E^c)$.

Given a regular expression E, by using Proposition 3, we solve the SORE-definability problem of E as follows:
(1) Construct the SOA S_E;
(2) Construct the candidate SORE E^c;
(3) Check whether $\mathcal{L}(E) = \mathcal{L}(E^c)$. If so, return **true**; otherwise, return **false**.

The correctness of the algorithm follows from Proposition 3. In the following, we analyze the complexity of this algorithm. From the results in [6, 3, 4, 13], we know that steps (1) and (2) can be done in polynomial time. Since the equivalence problem of regular expressions is PSPACE-complete [31], we know that step (3) can be fulfilled in polynomial space. It follows that the SORE-definability problem of regular expressions is in **PSPACE**. For the lower bound, we apply a reduction from the complement of the acceptance problem of polynomial-space bounded Turing machines. Then we get the following result.

▶ **Theorem 4.** *The SORE-definability problem is* **PSPACE***-complete for regular expressions.*

When E is a regular expression in $R(\#)$, *i.e.,* a regular expression with counting, we can expand E into a (standard) regular expression E', and then use the above algorithm to decide the SORE-definability problem. Since expanding E into E' takes exponential time, and the equivalence problem of regular expressions is **PSPACE**-complete [31], we conclude that the SORE-definability problem of $R(\#)$ is in **EXPSPACE**. For the **EXPSPACE** lower bound, we prove it by a reduction from the universality problem of regular expressions in $R(\#)$, which is known to be **EXPSPACE**-complete [31].

▶ **Theorem 5.** *The SORE-definability problem is* **EXPSPACE***-complete for $R(\#)$.*

3.2 Unary alphabets

In this section, we mainly consider the complexity of the SORE-definability problem for unary alphabets. As a by-product, we also solve an open problem in [27].

Let $\Sigma = \{a\}$. One can verify that an SORE E_1 over Σ must satisfy one of the following constraints: $\mathcal{L}(E_1) = \mathcal{L}(\varepsilon)$, $\mathcal{L}(E_1) = \mathcal{L}(a)$, $\mathcal{L}(E_1) = \mathcal{L}(a?)$, $\mathcal{L}(E_1) = \mathcal{L}(a^+)$, or $\mathcal{L}(E_1) = \mathcal{L}((a^+)?)$. Then to check whether $\mathcal{L}(E)$ can be defined by an SORE, we only need to check whether $\mathcal{L}(E) = \mathcal{L}(\varepsilon)$, $\mathcal{L}(E) = \mathcal{L}(a)$, $\mathcal{L}(E) = \mathcal{L}(a?)$, $\mathcal{L}(E) = \mathcal{L}(a^+)$, or $\mathcal{L}(E) = \mathcal{L}((a^+)?)$. Since the equivalence problems of regular expressions and regular expressions with counting over a unary alphabet are **coNP**-complete [34] and $\mathbf{\Pi_2^P}$-complete [34, 20] respectively, we get the **coNP** and $\mathbf{\Pi_2^P}$ upper bounds respectively for their SORE-definability problems over a unary alphabet. Moreover, we show the corresponding lower bounds by a reduction from the universality problem of regular expressions over a unary alphabet and the connectedness problem of integer expressions respectively, which are known to be **coNP**-complete [34] and $\mathbf{\Pi_2^P}$-complete [36, 33] respectively. Therefore, we get the following results.

▶ **Theorem 6.** *Over a unary alphabet, the SORE-definability problem is* **coNP***-complete for regular expressions.*

▶ **Theorem 7.** *Over a unary alphabet, the SORE-definability problem is $\mathbf{\Pi_2^P}$-complete for $R(\#)$.*

The work in [27] showed that the definability problem of deterministic regular expressions over a unary alphabet is in $\mathbf{\Pi_2^P}$ for $R(\#)$, but left the lower bound open. By using a reduction similar to the proof of the lower bound in Theorem 7, we can solve the open problem and get the following result.

▶ **Theorem 8.** *Over a unary alphabet, the definability problem of deterministic regular expressions is $\mathbf{\Pi_2^P}$-complete for $R(\#)$.*

Although the lower bound in Theorem 7 can also be proved by using the construction in [9], that construction cannot be used to prove the lower bound in Theorem 8, since the language defined by the regular expression constructed in [9] is already deterministic.

4 The Bounded SORE-definability problem

In this section, we will study the complexity of the bounded SORE-definability problem: Given a regular expression E (without or with counting) and a number M, whether there exists an SORE E_1 such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E_1)$, *i.e.*, for every word w such that $|w| \leq M$,

$w \in \mathcal{L}(E)$ iff $w \in L(E_1)$. As mentioned in the introduction, if the content model E in a DTD or XML Schema does not denote a deterministic regular language, if there is a bound M on the maximum number of children of nodes in XML documents, then we only need to give a deterministic content model E_1 to ensure that the language $\mathcal{L}(E_1)$ is equivalent to $\mathcal{L}(E)$ within the bound M.

We assume that in the bounded SORE-definability problem, the given regular expression (without or with counting) E and the number M satisfy that $M \geq 2 \cdot |\Sigma_E|$. This assumption is essential for the results in this section and it is open whether this assumption can be lifted.

Similar to the SORE-definability problem, the decision procedure for the bounded SORE-definability problem proceeds as follows:

1. At first, an SOA S_E is constructed from E such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(S_E)$ iff there exists an SOA A such that $\mathcal{L}(A) =_{\leq M} \mathcal{L}(E)$.
2. Then by using the algorithm Soa2Sore in [13], an SORE E^c is constructed from S_E such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$ iff there exists an SORE E' such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E')$.
3. Finally, decide whether $L(E) =_{\leq M} L(E^c)$.

In the following, we will first show how to construct S_E from E and E^c from S_E in Section 4.1, then based on these constructions, we derive the complexity results of the bounded SORE-definability problem in Section 4.2.

4.1 The construction of S_E and E^c

In this section, we assume that E is in $R(\#)$ and demonstrate how to construct an SOA S_E and an SORE E^c from E. The construction for regular expressions without counting is relatively easy and taken as a special case.

For the construction of S_E, one naive approach is to expand the expression E into a regular expression (without counting) E', and construct $S_{E'}$ from E'. But since the expansion of E incurs an exponential blow-up, we would not be able to achieve the tight complexity bounds (see, *e.g.,* Theorem 13 in Section 4.2). In the following, we show how we can circumvent the exponential blow-up and construct a desired SOA S_E in polynomial time.

▶ **Lemma 9.** *Given a regular expression E in $R(\#)$ and a number M encoded in binary, an SOA S_E satisfying the following constraint can be computed in polynomial time: $\mathcal{L}(S_E) =_{\leq M} \mathcal{L}(E)$ iff there exists an SOA A such that $\mathcal{L}(A) =_{\leq M} \mathcal{L}(E)$.*

Before presenting the construction of S_E, we introduce some additional notations. Let us assume that $M > 0$ in the following. For an expression E in $R(\#)$ and a natural number M, we define $first_M(E) = \{a \in \Sigma_E \mid \exists w_1.\ aw_1 \in \mathcal{L}(E) \cap (\Sigma_E)^{\leq M}\}$, $follow_M(E) = \{(a,b) \in \Sigma_E \times \Sigma_E \mid \exists w_1, w_2.\ w_1abw_2 \in \mathcal{L}(E) \cap (\Sigma_E)^{\leq M}\}$, and $last_M(E) = \{a \in \Sigma_E \mid \exists w_1.\ w_1a \in \mathcal{L}(E) \cap (\Sigma_E)^{\leq M}\}$.

For an expression E in $R(\#)$ and a natural number M, we construct an SOA $S_E = (\Sigma_E \cup \{q_I\}, \Sigma_E, \delta'_E, q_I, F'_E)$ satisfying the following constraints:

- $\{a \in \Sigma_E \mid \delta'_E(q_I, a) = a\} = first_M(E)$;
- $\{(a,b) \in \Sigma_E \times \Sigma_E \mid \delta'_E(a,b) = b\} = follow_M(E)$;
- if $\varepsilon \in L(E)$, then $F'_E = last_M(E) \cup \{q_I\}$; otherwise, $F'_E = last_M(E)$.

Therefore, the construction of S_E is equivalent to the computations of $first_M(E)$, $follow_M(E)$ and $last_M(E)$, which, we will show, can be done in polynomial time.

For the computations of $first_M(E)$, $follow_M(E)$ and $last_M(E)$, for each subexpression E' of E, we will compute an *up-to-M counting abstraction* of E' over Σ_E, denoted by $\mathsf{Abs}_{\Sigma_E,M}(E') = (x, T, F, L)$, in polynomial time, such that

- $x \in \{\mathbf{true}, \mathbf{false}\}$ denotes whether $\varepsilon \in \mathcal{L}(E')$;
- T is a function from $\Sigma_E \times \Sigma_E$ to $[M] \cup \{\infty\}$ such that $T(a,b) = \min(\{|w| \mid w \in \mathcal{L}(E') \cap (\Sigma_E)^{\leq M}, w = w_1abw_2 \text{ for some } w_1, w_2\})$, where $\min(\emptyset) = \infty$ by convention;
- F is a function from Σ_E to $[M] \cup \{\infty\}$ such that $F(a) = \min(\{|w| \mid w \in \mathcal{L}(E') \cap (\Sigma_E)^{\leq M}, w = aw_1 \text{ for some } w_1\})$;
- L is a function from Σ_E to $[M] \cup \{\infty\}$ such that $L(a) = \min(\{|w| \mid w \in \mathcal{L}(E') \cap (\Sigma_E)^{\leq M}, w = w_1a \text{ for some } w_1\})$.

Moreover, given $\mathsf{Abs}_{\Sigma_E,M}(E') = (x,T,F,L)$, we define $N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'))$ as the minimum length of the words in $\mathcal{L}(E') \cap (\Sigma_E)^{\leq M}$, that is, if $x = \mathbf{true}$, then $N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E')) = 0$; otherwise, $N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E')) = \min(\mathsf{rng}(T) \cup \mathsf{rng}(F) \cup \mathsf{rng}(L))$, where $\mathsf{rng}(T)$ denotes the range of T, similarly for $\mathsf{rng}(F)$ and $\mathsf{rng}(L)$. It is assumed that $n < \infty$ for each $n \in [M]$.

Evidently, given $\mathsf{Abs}_{\Sigma_E,M}(E) = (x,T,F,L)$, we have that $first_M(E) = \{a \in \Sigma_E \mid F(a) \neq \infty\}$, $follow_M(E) = \{(a,b) \in \Sigma_E \times \Sigma_E \mid T(a,b) \neq \infty\}$ and $last_M(E) = \{a \in \Sigma_E \mid L(a) \neq \infty\}$.

Next, we show how to compute $\mathsf{Abs}_{\Sigma_E,M}(E')$ from E' by a structural induction on E'.

(1) $E' = \varepsilon$. In this case, $\mathsf{Abs}_{\Sigma_E,M}(E') = (\mathbf{true}, T_\infty, F_\infty, L_\infty)$, where T_∞ denotes the function where $T_\infty(a,b) = \infty$ for each $a,b \in \Sigma_E$.

(2) $E' = a$ for any $a \in \Sigma_E$. In this case, $\mathsf{Abs}_{\Sigma_E,M}(E') = (\mathbf{false}, T_\infty, a \to 1, a \to 1)$, where $a \to 1$ denotes the function that maps a to 1 and maps all the other symbols from Σ_E to ∞.

(3) $E' = E'_1 + E'_2$. In this case, suppose that $\mathsf{Abs}_{\Sigma_E,M}(E'_i) = (x_i, T_i, F_i, L_i)$ for $i = 1,2$, then $\mathsf{Abs}_{\Sigma_E,M}(E') = (x,T,F,L)$, where

- $x = x_1 \vee x_2$ (x is the disjunction of x_1 and x_2);
- for each $(a,b) \in \Sigma_E \times \Sigma_E$, $T(a,b) = \min(\{T_1(a,b), T_2(a,b)\})$;
- for each $a \in \Sigma_E$, $F(a) = \min(\{F_1(a), F_2(a)\})$;
- for each $a \in \Sigma_E$, $L(a) = \min(\{L_1(a), L_2(a)\})$.

(4) $E' = E'_1E'_2$. In this case, suppose that $\mathsf{Abs}_{\Sigma_E,M}(E'_i) = (x_i, T_i, E_i, L_i)$ for $i = 1,2$, then $\mathsf{Abs}_{\Sigma_E,M}(E') = (x,T,F,L)$, where

- $x = x_1 \wedge x_2$ (x is the conjunction of x_1 and x_2);
- for each $(a,b) \in \Sigma_E \times \Sigma_E$, let

$$T(a,b) = \min\left([M] \bigcap \left(\begin{array}{l}\{T_1(a,b) + N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_2))\} \\ \cup \{T_2(a,b) + N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\} \\ \cup \{L_1(a) + F_2(b)\}\end{array}\right)\right),$$

  note that here we assume $\infty + \infty = \infty + n = n + \infty = \infty$ for every natural number n;
- for each $a \in \Sigma_E$, let

$$F(a) = \min\left([M] \bigcap \left(\{F_1(a) + N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_2))\} \cup \{F_2(a) \mid x_1 = \mathbf{true}\}\right)\right);$$

- for each $a \in \Sigma_E$, let

$$L(a) = \min\left([M] \bigcap \left(\{L_2(a) + N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\} \cup \{L_1(a) \mid x_2 = \mathbf{true}\}\right)\right).$$

(5) $E' = (E'_1)^{[m,n]}$ or $E' = (E'_1)^{[m,\infty]}$. Since the analysis of $E' = (E'_1)^{[m,\infty]}$ is almost the same as $E' = (E'_1)^{[m,n]}$, we only show the analysis of $E' = (E'_1)^{[m,n]}$. Let $\mathsf{Abs}_{\Sigma_E,M}(E'_1) = (x_1, T_1, F_1, L_1)$. Then $\mathsf{Abs}_{\Sigma_E,M}(E') = (x,T,F,L)$, where

- if $m \geq 2$, then
  - $x = x_1$,

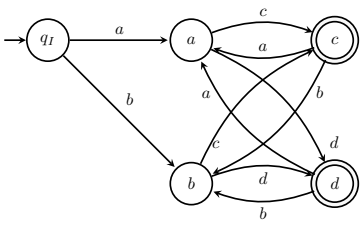


Figure 2 The SOA S_E.

- for each $(a, b) \in \Sigma_E \times \Sigma_E$, let

$$T(a,b) = \min\left([M] \bigcap \left(\begin{array}{l} \{T_1(a,b) + (m-1)N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\} \cup \\ \{L_1(a) + F_1(b) + (m-2)N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\} \end{array} \right)\right),$$

  note that here we assume that $0 \times \infty = 0$ and $n \times \infty = \infty$ for each $n > 0$,
- for each $a \in \Sigma_E$, let

$$F(a) = \min\left([M] \bigcap \{F_1(a) + (m-1)N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\}\right),$$

- for each $a \in \Sigma_E$, let

$$L(a) = \min\left([M] \bigcap \{L_1(a) + (m-1)N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\}\right);$$

- if $m = 1$ and $n = 1$, then $\mathsf{Abs}_{\Sigma_E,M}((E'_1)^{[m,n]}) = \mathsf{Abs}_{\Sigma_E,M}(E'_1)$;
- if $m = 1$ and $n \geq 2$, then $\mathsf{Abs}_{\Sigma_E,M}((E'_1)^{[m,n]}) = \mathsf{Abs}_{\Sigma_E,M}(E'_1 + (E'_1)^{[m+1,n]})$, which can be computed from $\mathsf{Abs}_{\Sigma_E,M}(E'_1)$ and $\mathsf{Abs}_{\Sigma_E,M}((E'_1)^{[m+1,n]})$ by the aforementioned construction for the $+$ operator (note that $(E'_1)^{[m+1,n]}$ satisfies that $m + 1 \geq 2$);
- if $m = 0$ and $n = 0$, then $\mathsf{Abs}_{\Sigma_E,M}((E'_1)^{[m,n]}) = \mathsf{Abs}_{\Sigma_E,M}(\varepsilon)$;
- if $m = 0$ and $n \geq 1$, then $\mathsf{Abs}_{\Sigma_E,M}((E'_1)^{[m,n]}) = \mathsf{Abs}_{\Sigma_E,M}(\varepsilon + (E'_1)^{[m+1,n]})$ (note that $(E'_1)^{[m+1,n]}$ satisfies that $m + 1 \geq 1$).

The above computation of $\mathsf{Abs}_{\Sigma_E,M}(E)$ can be done in polynomial time, since each $\mathsf{Abs}_{\Sigma_E,M}(E')$ occupies only polynomial space and the computation takes at most $O(|E|)$ steps.

▶ **Example 10.** We will use the following example to show all the constructions. Let $E = ((a + b) \cdot (c + d) \cdot (\varepsilon + (ae)^{[5,5]}))^{[2,\infty]}$ and $M = 9$. The computation of $\mathsf{Abs}_{\Sigma_E,M}(E')$ is shown in Table 2, where **T** stands for **true**, **F** stands for **false**, and the pairs (a, b) such that $T(a, b) = \infty$ are omitted, similarly for F and L. Consider the computation of $\mathsf{Abs}_{\Sigma_E,M}((ae)^{[5,5]})$. It is easy to verify that $\mathsf{Abs}_{\Sigma_E,M}(ae) = (\mathbf{false}, \{(a, e)\to 2\}, a\to 2, e\to 2)$. Since $M = 9$, and ae is the shortest word in $\mathcal{L}(ae)$, we have that $2 + (m - 1) \times 2 = 2 + 4 \times 2 = 10 > M$. Therefore, $\mathsf{Abs}_{\Sigma_E,M}((ae)^{[5,5]}) = (\mathbf{false}, T_\infty, T_\infty, T_\infty)$, which means that any word in $\mathcal{L}((ae)^{[5,5]})$ cannot be a sub-word of w in $\mathcal{L}(E)$ such that $|w| \leq M$. Let $\mathsf{Abs}_{\Sigma_E,M}(E) = (x, T, F, L)$. Then $follow_M(E) = \{(a', b') \in \Sigma_E \times \Sigma_E \mid T(a', b') \neq \infty\} = \{(a, c), (b, c), (a, d), (b, d), (c, a), (d, a), (c, b), (d, b)\}$. Similarly, $first_M(E) = \{a' \in \Sigma_E \mid F(a') \neq \infty\} = \{a, b\}$, and $last_M(E) = \{a' \in \Sigma_E \mid L(a') \neq \infty\} = \{c, d\}$. From the sets $first_M$, $follow_M$, and $last_M$, we construct an SOA S_E illustrated in Figure 2.

By using S_E, we can construct the candidate SORE E^c for E in Example 12.

Computation of E^c. Again, we use the algorithm Soa2Sore [13] mentioned in Section 3 to compute an SORE E^c from S_E in polynomial time. As a result of the assumption that $M \geq 2 \cdot |\Sigma_E|$, the SORE E^c enjoys the following property.

Table 2 The computation of $\mathsf{Abs}_{\Sigma_E,M}(E')$.

E'	$\mathsf{Abs}_{\Sigma_E,M}(E')$	E'	$\mathsf{Abs}_{\Sigma_E,M}(E')$
a	$(\mathbf{F}, T_\infty, a\to 1, a\to 1)$	$c+d$	$(\mathbf{F}, T_\infty, \left\{\begin{smallmatrix} c\to 1 \\ d\to 1 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} c\to 1 \\ d\to 1 \end{smallmatrix}\right\})$
b	$(\mathbf{F}, T_\infty, b\to 1, b\to 1)$	ae	$(\mathbf{F}, \{(a,e)\to 2\}, a\to 2, e\to 2)$
c	$(\mathbf{F}, T_\infty, c\to 1, c\to 1)$	$(a+b)(c+d)$	$(\mathbf{F}, \left\{\begin{smallmatrix} (a,c)\to 2 \\ (b,c)\to 2 \\ (a,d)\to 2 \\ (b,d)\to 2 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} a\to 2 \\ b\to 2 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} c\to 2 \\ d\to 2 \end{smallmatrix}\right\})$
d	$(\mathbf{F}, T_\infty, d\to 1, d\to 1)$	$(ae)^{[5,5]}$	$(\mathbf{F}, T_\infty, T_\infty, T_\infty)$
e	$(\mathbf{F}, T_\infty, e\to 1, e\to 1)$	$\varepsilon+(ae)^{[5,5]}$	$(\mathbf{T}, T_\infty, T_\infty, T_\infty)$
$a+b$	$(\mathbf{F}, T_\infty, \left\{\begin{smallmatrix} a\to 1 \\ b\to 1 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} a\to 1 \\ b\to 1 \end{smallmatrix}\right\})$	$(a+b)(c+d)(\varepsilon+(ae)^{[5,5]})$	$(\mathbf{F}, \left\{\begin{smallmatrix} (a,c)\to 2 \\ (b,c)\to 2 \\ (a,d)\to 2 \\ (b,d)\to 2 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} a\to 2 \\ b\to 2 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} c\to 2 \\ d\to 2 \end{smallmatrix}\right\})$
$((a+b)(c+d)(\varepsilon+(ae)^{[5,5]}))^{[2,\infty]}$		$(\mathbf{F}, \left\{\begin{smallmatrix} (a,c)\to 4 & (c,a)\to 4 \\ (b,c)\to 4 & (d,a)\to 4 \\ (a,d)\to 4 & (c,b)\to 4 \\ (b,d)\to 4 & (d,b)\to 4 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} a\to 4 \\ b\to 4 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} c\to 4 \\ d\to 4 \end{smallmatrix}\right\})$	

▶ **Proposition 11.** *$\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$ iff there exists an SORE E' such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E')$.*

The arguments for Proposition 11 proceed as follows: The "only if" direction is trivial. For the "if" direction, suppose that there exists an SORE E' such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E')$. Then from the fact that $\mathcal{L}(E')$ can be defined by an SOA, according to Lemma 9, we know that $\mathcal{L}(S_E) =_{\leq M} \mathcal{L}(E)$. Therefore, we have $\mathcal{L}(S_E) =_{\leq M} \mathcal{L}(E')$. From the assumption $M \geq 2 \cdot |\Sigma_E|$, we can further show that $\mathcal{L}(S_E) = \mathcal{L}(E')$. Moreover, in [13], it was proved that the SORE E^c satisfies the "SORE-descriptive" property, *i.e.* there does not exist an SORE E'' such that $\mathcal{L}(S_E) \subseteq \mathcal{L}(E'') \subset \mathcal{L}(E^c)$. Therefore, we must have $\mathcal{L}(S_E) = \mathcal{L}(E^c)$. From $\mathcal{L}(S_E) =_{\leq M} \mathcal{L}(E)$, we conclude that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$.

▶ **Example 12.** Let us continue Example 10. By using the algorithm Soa2Sore in [13], from the SOA S_E in Figure 2, we obtain the following SORE: $((a+b)(c+d))^+$. It is easy to verify that $\mathcal{L}(E) =_{\leq 9} \mathcal{L}(((a+b)(c+d))^+)$. Then E can be represented by an SORE within the bound $M = 9$. On the other hand, by the results in Section 3, we can check that $\mathcal{L}(E)$ cannot be defined by an SORE.

4.2 The Complexity

In this section, we establish the complexity results of the bounded SORE-definability problem. Given a regular expression E in $R(\#)$ and a number M, by using Proposition 11, we can develop the following algorithm to decide the bounded SORE-definability problem:

(1) Compute the set $\mathsf{Abs}_{\Sigma_E,M}(E)$;
(2) Construct $first_M(E)$, $follow_M(E)$, $last_M(E)$, and the candidate SORE E^c;
(3) Check whether $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$. If so, return **true**; otherwise, return **false**.

The correctness of this algorithm follows from Proposition 11. In the following, we analyse the complexity of the algorithm. From the computations of $\mathsf{Abs}_{\Sigma_E,M}(E)$, $first_M$, $follow_M$, $last_M$, and E^c in Section 4.1, we know that steps (1) and (2) can be done in polynomial time. For step (3), we distinguish between the unary and binary encoding of M.

M is encoded in unary.

Since M is encoded in unary, to check whether $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$, we first guess a word w such that $|w| \leq M$, then check whether $w \in (\mathcal{L}(E) \setminus \mathcal{L}(E^c))$ or $w \in (\mathcal{L}(E^c) \setminus \mathcal{L}(E))$.

Since checking whether $w \in \mathcal{L}(E)$ and $w \in \mathcal{L}(E^c)$ is in **PTIME** [22], we deduce that the bounded SORE-definability problem for $R(\#)$ is in **coNP**. By a reduction from the bounded universality problem, which is known to be **coNP**-complete [10], we also show that the bounded SORE-definability problem for $R(\#)$ is **coNP**-hard.

▶ **Theorem 13.** *The bounded SORE-definability problem is* **coNP***-complete for $R(\#)$ and natural numbers M encoded in unary.*

One may wonder whether the bounded SORE-definability problem would be easier to solve, if E is a regular expression. The answer to this question is negative, since the expression constructed in the lower-bound proof of Theorem 13 is already a regular expression.

▶ **Corollary 14.** *The bounded SORE-definability problem is* **coNP***-complete for regular expressions and natural numbers M encoded in unary.*

If $|\Sigma_E| = 1$, then $(\Sigma_E)^{\leq M}$ contains at most $M + 1$ words. Therefore, in this case, when M is encoded in unary, step (3) can be done in **PTIME** and we have the following result.

▶ **Theorem 15.** *The bounded SORE-definability problem is in* **PTIME** *for unary regular expressions in $R(\#)$ and natural numbers M encoded in unary.*

From the aforementioned results and the ones in Section 3, we can see that if M is encoded in unary, then there is an exponential decrease in the complexity when switching from the SORE-definability problem to its bounded variant. For instance, for $R(\#)$, the SORE-definability problem is **EXPSPACE**-complete, while the bounded SORE-definability problem is **coNP**-complete if M is encoded in unary.

M is encoded in binary.

As mentioned above, $\mathsf{Abs}_{\Sigma_E,M}(E)$ can be computed in polynomial time even if M is encoded in binary. Nevertheless, since M is encoded in binary, the complexity of step (3) may increase exponentially.

Similar to the algorithm for Theorem 13, step (3) can also be done by guessing a word w such that $|w| \leq M$. Since M is encoded in binary, w can be exponentially large. Then checking whether $w \in \mathcal{L}(E)$ and $w \in \mathcal{L}(E^c)$ can be done in exponential time and we get a **coNEXPTIME** upper bound for bounded SORE-definability problem problem of $R(\#)$. For the **coNEXPTIME** lower bound, we get a reduction from the complement of the acceptance problem of nondeterministic exponential-time Turing machines. Therefore, we obtain the following result.

▶ **Theorem 16.** *The bounded SORE-definability problem is* **coNEXPTIME***-complete for $R(\#)$ and natural numbers M encoded in binary.*

Next, we consider the case when E is a regular expression. Step (1) and (2) can still be done in polynomial time. To check whether $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$, we can construct two NFA from E and E^c respectively, guess a word w such that $|w| \leq M$, and check whether $w \in \mathcal{L}(E) \setminus \mathcal{L}(E^c)$ or $w \in \mathcal{L}(E^c) \setminus \mathcal{L}(E)$. The nondeterministic algorithm uses polynomial space. By Savitch's Theorem [32], we conclude that the bounded SORE-definability problem for regular expressions is in PSPACE. For the lower bound, we can directly use the same reduction in Theorem 4, and let $M = 2^{|E|}$. The correctness follows from the following arguments: Since the number of states of the minimum DFA for E is at most M [17], it is easy to check that there exists an SORE E_1 such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E_1)$ iff there exists an SORE E_2 such that $\mathcal{L}(E) = \mathcal{L}(E_2)$.

▶ **Corollary 17.** *The bounded SORE-definability problem is* **PSPACE***-complete for regular expressions and natural numbers M encoded in binary.*

It is interesting to observe that the complexities of the SORE-definability problem and its bounded variant are the same for regular expressions, while the complexities of the two problems are different for $R(\#)$. This distinction is attributed to the following facts: (1) given a regular expression E, $\mathcal{L}(E) = \Sigma_E^*$, if and only if, $\mathcal{L}(E) =_{\leq 2^{|E|}} \Sigma_E^*$; while (2) given $E \in R(\#)$, $\mathcal{L}(E) = \Sigma_E^*$, if and only if, $\mathcal{L}(E) =_{\leq 2^{2^{|E|}}} \Sigma_E^*$ [31]. That is, to decide whether $\mathcal{L}(E) = \Sigma_E^*$ for $E \in R(\#)$, we have to check double-exponentially many words in $\mathcal{L}(E)$, while for regular expressions, we only need to check exponentially many words. So the bounded SORE-definability problem is simpler than the SORE-definability problem for $R(\#)$.

Given a regular expression E in $R(\#)$ over a unary alphabet, since the minimum DFA for E has at most $M = 2^{2\cdot|E|+4} + 1$ states [27], we have the following results.

▶ **Corollary 18.** *Over a unary alphabet, the bounded SORE-definability problem is* $\mathbf{\Pi_2^p}$*-complete for $R(\#)$ and natural numbers M encoded in binary.*

▶ **Corollary 19.** *Over a unary alphabet, the bounded SORE-definability problem is* **coNP***-complete for regular expressions and natural numbers M encoded in binary.*

5 Conclusion

In this paper, we study the complexity of the SORE-definability problem as well as its bounded variant. The results of the paper were summarised in Table 1. As a by-product of the results obtained in this paper, we also solved an open problem in [27] and showed that over a unary alphabet, the definability problem of deterministic regular expressions is $\mathbf{\Pi_2^p}$-complete for regular expressions with counting.

There are several directions for the future work. An obvious question left open in this paper is whether the assumption $M \geq 2 \cdot |\Sigma_E|$ for the bounded SORE-definability problem can be lifted. Without this assumption, given an SOA A, it is unclear whether it is still in **PTIME** to decide whether there exists an SORE E_1 such that $\mathcal{L}(E_1) =_{\leq M} \mathcal{L}(A)$. Another interesting question is to investigate the definability problem for single occurrence regular expressions with counting (SORE(#)). The main technical challenge is how to obtain a candidate SORE(#). Similarly, one can also consider the CHARE-definability problem (see [3, 28] for the definition of CHARE).

References

1. Geert Jan Bex, Wouter Gelade, Wim Martens, and Frank Neven. Simplifying XML Schema: effortless handling of nondeterministic regular expressions. In *SIGMOD*, pages 731–744. ACM, 2009.
2. Geert Jan Bex, Wouter Gelade, Frank Neven, and Stijn Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. *ACM Transactions on the Web*, 4(4):14, 2010.
3. Geert Jan Bex, Frank Neven, Thomas Schwentick, and Karl Tuyls. Inference of concise DTDs from XML data. In *VLDB*, pages 115–126, 2006.
4. Geert Jan Bex, Frank Neven, Thomas Schwentick, and Stijn Vansummeren. Inference of concise regular expressions and DTDs. *ACM Transactions on Database Systems*, 35(2):11, 2010.
5. Anne Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2):197–213, 1993.

6 Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Information and Computation*, 140(2):229–253, 1998.

7 Haiming Chen and Ping Lu. Assisting the design of XML Schema: diagnosing nondeterministic content models. In *Web Technologies and Applications*, pages 301–312. Springer, 2011.

8 Haiming Chen and Ping Lu. Checking determinism of regular expressions with counting. *Information and Computation*, 241:302–320, 2015.

9 Dmitry Chistikov and Rupak Majumdar. Unary pushdown automata and straight-line programs. In *ICALP*, pages 146–157. Springer, 2014.

10 Sang Cho and Dung T Huynh. The parallel complexity of finite-state automata problems. *Information and Computation*, 97(1):1–22, 1992.

11 World Wide Web Consortium. `http://www.w3.org/wiki/UniqueParticleAttribution`.

12 Wojciech Czerwiński, Claire David, Katja Losemann, and Wim Martens. Deciding definability by deterministic regular expressions. In *FOSSACS*, pages 289–304, 2013.

13 Dominik D Freydenberger and Timo Kötzing. Fast learning of restricted regular expressions and DTDs. *Theory of Computing Systems*, 57(4):1114–1158, 2015.

14 Wouter Gelade, Marc Gyssens, and Wim Martens. Regular expressions with counting: Weak versus strong determinism. *SIAM Journal on Computing*, 41(1):160–190, 2012.

15 Victor Mikhaylovich Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16(5):1, 1961.

16 Benoît Groz, Sebastian Maneth, and Sławek Staworko. Deterministic regular expressions in linear time. In *PODS*, pages 49–60. ACM, 2012.

17 John E Hopcroft and Jeffrey D Ullman. *Introduction to automata theory, languages, and computation, first edition*. Pearson, 1979.

18 Dag Hovland. Regular expressions with numerical constraints and automata with counters. In *ICTAC*, pages 231–245, 2009.

19 Dag Hovland. The membership problem for regular expressions with unordered concatenation and numerical constraints. In *LATA*, pages 313–324, 2012.

20 Dung T Huynh. Deciding the inequivalence of context-free grammars with 1-letter terminal alphabet is Σ_2^p-complete. *Theoretical Computer Science*, 33(2-3):305–326, 1984.

21 Pekka Kilpeläinen. Checking determinism of XML Schema content models in optimal time. *Information Systems*, 36(3):596–617, 2011.

22 Pekka Kilpeläinen and Rauno Tuhkanen. Regular Expressions with Numerical Occurrence Indicators-preliminary results. In *SPLST*, pages 163–173, 2003.

23 Pekka Kilpeläinen and Rauno Tuhkanen. One-unambiguity of regular expressions with numeric occurrence indicators. *Information and Computation*, 205(6):890–916, 2007.

24 Markus Latte and Matthias Niewerth. Definability by Weakly Deterministic Regular Expressions with Counters is Decidable. In *MFCS*, pages 369–381, 2015.

25 Ping Lu. *Research on deterministic regular languages (in Chinese)*. PhD thesis, University of Chinese Academy of Sciences, May 2014.

26 Ping Lu, Joachim Bremer, and Haiming Chen. Deciding Determinism of Regular Languages. *Theory of Computing Systems*, 57(1):97–139, 2015.

27 Ping Lu, Feifei Peng, Haiming Chen, and Lixiao Zheng. Deciding determinism of unary languages. *Information and Computation*, 245:181–196, 2015.

28 Wim Martens, Frank Neven, and Thomas Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM Journal on Computing*, 39(4):1486–1530, 2009.

29 Wim Martens, Frank Neven, Thomas Schwentick, and Geert Jan Bex. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems*, 31(3):770–813, 2006.

30 Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 1(EC-9):39–47, 1960.

31 Albert R Meyer and Larry J Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT (FOCS)*, pages 125–129, 1972.

32 Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

33 Marcus Schaefer and Christopher Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT news*, 33(3):32–49, 2002.

34 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973.

35 Eric van der Vlist. *XML Schema*. O'Reilly Media, Inc., 2002.

36 Klaus W Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986.

TC^0 Circuits for Algorithmic Problems in Nilpotent Groups

Alexei Myasnikov[1] and Armin Weiß[2]

1 **Stevens Institute of Technology, Hoboken, NJ, USA**
2 **Universität Stuttgart, Germany**

Abstract

Recently, Macdonald et. al. showed that many algorithmic problems for finitely generated nilpotent groups including computation of normal forms, the subgroup membership problem, the conjugacy problem, and computation of subgroup presentations can be done in LOGSPACE. Here we follow their approach and show that all these problems are complete for the uniform circuit class TC^0 – uniformly for all r-generated nilpotent groups of class at most c for fixed r and c.

Moreover, if we allow a certain binary representation of the inputs, then the word problem and computation of normal forms is still in uniform TC^0, while all the other problems we examine are shown to be TC^0-Turing reducible to the problem of computing greatest common divisors and expressing them as linear combinations.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.0 Discrete Mathematics

Keywords and phrases nilpotent groups, TC^0, abelian groups, word problem, conjugacy problem, subgroup membership problem, greatest common divisors

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.23

1 Introduction

The word problem (given a word over the generators, does it represent the identity?) is one of the fundamental algorithmic problems in group theory introduced by Dehn in 1911 [3]. While for general finitely presented groups all these problems are undecidable [22, 2], for many particular classes of groups decidability results have been established – not just for the word problem but also for a wide range of other problems. Finitely generated nilpotent groups are a class where many algorithmic problems are (efficiently) decidable (with some exceptions like the problem of solving equations – see e. g. [6]). In 1958, Mal'cev [17] established decidability of the word and subgroup membership problem by investigating finite approximations of nilpotent groups. In 1965, Blackburn [1] showed decidability of the conjugacy problem. However, these methods did not allow any efficient (e. g. polynomial time) algorithms. Nevertheless, in 1966 Mostowski provided "practical" algorithms for the word problem and several other problems [18]. In terms of complexity, a major step was the result by Lipton and Zalcstein [15] that the word problem of linear groups is in LOGSPACE. Together with the fact that finitely generated nilpotent groups are linear (see e. g. [7, 10]) this gives a LOGSPACE solution to the word problem of nilpotent groups, which was later improved to uniform TC^0 by Robinson [23]. A typical algorithmic approach to nilpotent groups is using so-called Mal'cev (or Hall–Mal'cev) bases (see e. g. [7, 10]), which allow to carry out group operations by evaluating polynomials (see Lemma 2). This approach was systematically used in [11] and [18] or – in the more general setting of polycyclic presentations – in [24] for solving (among others) the subgroup membership and conjugacy

problem of polycyclic groups. Recently in [19, 20] polynomial time bounds for the equalizer and subgroup membership problems in nilpotent groups have been given. Finally, in [16] the following problems were shown to be in LOGSPACE using the Mal'cev basis approach. Here, $\mathcal{N}_{c,r}$ denotes the class of nilpotent groups of nilpotency class at most c generated by at most r elements.

- The *word problem*: given $G \in \mathcal{N}_{c,r}$ and $g \in G$, is $g = 1$ in G?
- Given $G \in \mathcal{N}_{c,r}$ and $g \in G$, compute the (Mal'cev) normal form of g.
- The *subgroup membership problem*: Given $G \in \mathcal{N}_{c,r}$ and $g, h_1, \ldots, h_n \in G$, decide whether $g \in \langle h_1, \ldots, h_n \rangle$ and, if so, express g as a word over the subgroup generators $h_1, \ldots, h_n$ (in [16] only the decision version was shown to be in LOGSPACE – for expressing g as a word over the original subgroup generators a polynomial time bound was given).
- Given $G, H \in \mathcal{N}_{c,r}$ and $K = \langle g_1, \ldots, g_n \rangle \leq G$, together with a homomorphism $\varphi : K \to H$ specified by $\varphi(g_i) = h_i$, and some $h \in \mathrm{Im}(\varphi)$, compute a generating set for $\ker(\varphi)$ and find $g \in G$ such that $\varphi(g) = h$.
- Given $G \in \mathcal{N}_{c,r}$ and $K = \langle g_1, \ldots, g_n \rangle \leq G$, compute a presentation for K.
- Given $G \in \mathcal{N}_{c,r}$ and $g \in G$, compute a generating set for the centralizer of g.
- The *conjugacy problem*: Given $G \in \mathcal{N}_{c,r}$ and $g, h \in G$, decide whether or not there exists $u \in G$ such that $u^{-1}gu = h$ and if so find such an element u.

Notice that these problems are not only of interest in themselves, but also might serve as building blocks for solving the same problems in polycyclic groups – which are of particular interest because of their possible application in non-commutative cryptography [4]. In this work we follow [16] and extend these results in several ways:

- We give a complexity bound of uniform TC^0 for all the above problems.
- In order to derive this bound, we show that the extended gcd problem with unary coefficients is in TC^0.
- Our description of circuits is for the uniform setting where $G \in \mathcal{N}_{c,r}$ is part of the input (in [16] the uniform setting is also considered; however, only in some short remarks).
- Since nilpotent groups have polynomial growth, it is natural to allow compressed inputs: we give a uniform TC^0 solution for the word problem allowing words with binary exponents as input – this contrasts with the situation with straight-line programs (i. e., context-free grammars which produces precisely one word – another method of exponential compression) as input: then the word problem is hard for $\mathsf{C}_=\mathsf{L}$ [12]. Thus, the difficulty of the word problem with straight-line programs is not due to their compression but rather due to the difficulty of evaluating a straight-line program.
- We show that the other of the above problems are uniform-TC^0-Turing-reducible to the extended gcd problem (compute the greatest common divisor and express it as a linear combination) when the inputs (both the ambient group and the subgroup etc.) are given as words with binary exponents.

Thus, in the unary case we settle the complexity of the above problems completely. Moreover, it also seems rather unlikely that the subgroup membership problem can be solved without computing gcds – in this case our results on binary inputs would be also optimal. Altogether, our results mean that many algorithmic problems are no more complicated in nilpotent groups than in abelian groups. Notice that while in [16] explicit length bounds on the outputs for all these problems are proven, we obtain polynomial length bounds simply by the fact that everything can be computed in uniform TC^0 (for which in the following we only write TC^0). Throughout the paper we follow the outline of [16]. For a concise presentation, we copy many definitions from [16]. Most of our theorems involve two statements: one for unary encoded inputs and one for binary encoded inputs. In order to have a concise presentation, we always put them in one statement. We only consider finitely generated nilpotent groups without mentioning that further.

Outline. We start with basic definitions on complexity as well as on nilpotent groups. In Section 2 we describe how subgroups of nilpotent groups can be represented and develop a "nice" presentation for all groups in $\mathcal{N}_{c,r}$. Section 3 deals with the word problem and computation of normal forms. Based on this we introduce the so-called matrix reduction and solve the subgroup membership problem. Finally, in Section 5 we present our result for the remaining of the above problems – the proofs are essentially repeated applications of the matrix reduction. Due to space constraints many of the proofs are omitted – they can be found in the full version on arXiv [21].

1.1 Preliminaries on Complexity

For a finite *alphabet* Σ, the set of *words* over Σ is denoted by Σ^*. Computation or decision problems are given by functions $f : \Delta^* \to \Sigma^*$ for some finite alphabets Δ and Σ. A decision problem (= formal language) L is identified with its characteristic function $\chi_L : \Delta^* \to \{0, 1\}$ with $\chi_L(x) = 1$ if, and only if, $x \in L$. (In particular, the word and conjugacy problems can be seen as functions $\Sigma^* \to \{0, 1\}$.) We use circuit complexity as described in [25].

Circuit Classes. The class TC^0 is defined as the class of functions computed by families of circuits of constant depth and polynomial size with unbounded fan-in Boolean gates (and, or, not) and majority gates. A majority gate (denoted by Maj) returns 1 if the number of 1s in its input is greater or equal to the number of 0s. In the following we always assume that the alphabets Δ and Σ are encoded over the binary alphabet $\{0, 1\}$ such that each letter uses the same number of bits. We say a function f is TC^0-*computable* if $f \in \mathsf{TC}^0$.

In the following, we only consider Dlogtime-uniform circuit families and we simply write TC^0 as shorthand for Dlogtime-uniform TC^0. Dlogtime-uniform means that there is a deterministic Turing machine which decides in time $\mathcal{O}(\log n)$ on input of two gate numbers (given in binary) and the string 1^n whether there is a wire between the two gates in the n-input circuit and also computes of which type some gates is. Note that the binary encoding of the gate numbers requires only $\mathcal{O}(\log n)$ bits – thus, the Turing machine is allowed to use time linear in the length of the encodings of the gates. For more details on these definitions we refer to [25]. We have the inclusions $\mathsf{AC}^0 \subsetneq \mathsf{TC}^0 \subseteq \mathsf{LOGSPACE} \subseteq \mathsf{P}$ (note that even $\mathsf{TC}^0 \subseteq \mathsf{P}$ is not known to be strict).

Reductions. A function f is TC^0-*Turing-reducible* to a function g if there is a Dlogtime-uniform family of TC^0 circuits computing f which, in addition to the Boolean and majority gates, also may use oracle gates for g (i. e., gates which on input x output $g(x)$). This is expressed by $f \in \mathsf{TC}^0(g)$. Note that if $f_1, \ldots, f_k$ are in TC^0, then $\mathsf{TC}^0(f_1, \ldots, f_k) = \mathsf{TC}^0$.

In particular, if f and g are TC^0-computable functions, then also the composition $g \circ f$ is TC^0-computable. We will extensively make use of this observation – which will also guarantee the polynomial size bound on the outputs of our circuits without additional calculations.

We will also use another fact frequently without giving further reference: on input of two alphabets Σ and Δ (coded over the binary alphabet), a list of pairs (a, v_a) with $a \in \Sigma$ and $v_a \in \Delta^*$ such that each $a \in \Sigma$ occurs in precisely one pair, and a word $w \in \Sigma^*$, the image $\varphi(w)$ under the homomorphism φ defined by $\varphi(a) = v_a$ can be computed in TC^0 [13].

Encoding numbers: unary vs. binary. There are essentially two ways of representing integer numbers: the usual way as a binary number where a string $a_0 \cdots a_n$ with $a_i \in \{0, 1\}$ represents $\sum a_i 2^{n-i}$, and as a unary number where $k \in \mathbb{N}$ is represented by $1^k = \underbrace{11 \cdots 1}_{k}$ (respectively by $0^{n-k}1^k$ if n is the number of input bits).

We will state most results in this paper with both representations. The unary representation corresponds to group elements given as words over the generators, whereas the binary encoding will be used if inputs are given in a compressed form.

Arithmetic in TC^0. Iterated Addition (resp. Iterated Multiplication) are the following computation problems: On input of n binary integers $a_1, \ldots, a_n$ each having n bits (i. e., the input length is $N = n^2$), compute the binary representation of the sum $\sum_{i=1}^{n} a_i$ (resp. product $\prod_{i=1}^{n} a_i$). For Integer Division the input are two binary n-bit integers a, b; the binary representation of the integer $c = \lfloor a/b \rfloor$ has to be computed. The first statement of Theorem 1 is a standard fact, see [25]; the other statements are due to Hesse, [8, 9].

▶ **Theorem 1** ([8, 9, 25]). *The problems* Iterated Addition, Iterated Multiplication, Integer Division *are all in* TC^0 *no matter whether inputs are given in unary or binary.*

Note that if the numbers a and b are encoded in unary (as strings 1^a and 1^b), division can be seen to be in TC^0 very easily: just try for all $0 \leq c \leq a$ whether $0 \leq a - bc < b$.

Representing groups for algorithmic problems. We consider finitely generated groups G together with finite generating sets A. Group elements are represented as words over the generators and their inverses (i. e., as elements of $(A \cup A^{-1})^*$). We make no distinction between words and the group elements they represent. Whenever it might be unclear whether we mean equality of words or of group elements, we write "$g = h$ in G" for equality in G.

Words over the generators ± 1 of $\mathbb{Z}$ correspond to unary representation of integers. As a generalization of binary encoded integers, we introduce the following notion: a *word with binary exponents* is a sequence $w_1, \ldots, w_n$ where the w_i are from a fixed generating set of the group together with a sequence of exponents $x_1, \ldots, x_n$ where the $x_i \in \mathbb{Z}$ are encoded in binary. The word with binary exponents represents the word (or group element) $w = w_1^{x_1} \cdots w_n^{x_n}$. Note that in a fixed nilpotent group *every* word of length n can be rewritten as a word with binary exponents using $\mathcal{O}(\log n)$ bits (this fact is well-known and also a consequence of Theorem 5 below); thus, words with binary exponents are a natural way of representing inputs for algorithmic problems in nilpotent groups.

1.2 Preliminaries on Nilpotent groups and Mal'cev coordinates

Let G be a group. For $x, y \in G$ we write $[x, y] = x^{-1}y^{-1}xy$ for the *commutator* of x and y. For subgroups $H_1, H_2 \leq G$, we have $[H_1, H_2] = \langle \{[h_1, h_2] \mid h_1 \in H_1, h_2 \in H_2\} \rangle$. A group G is called *nilpotent* if it has a finite central series, i.e.

$$G = G_1 \geq G_2 \geq \cdots \geq G_c \geq G_{c+1} = 1 \tag{1}$$

such that $[G, G_i] \leq G_{i+1}$ for all $i = 1, \ldots, c$. If G is finitely generated, so are the abelian quotients G_i/G_{i+1}, $1 \leq i \leq c$. Let $a_{i1}, \ldots, a_{im_i}$ be a basis of G_i/G_{i+1}, i.e. a generating set such that G_i/G_{i+1} has a presentation $\left\langle a_{i1}, \ldots, a_{im_i} \,\middle|\, a_{ij}^{e_{ij}}, [a_{ik}, a_{i\ell}], \text{ for } j \in \mathcal{T}_i,\ k, \ell \in \{1, \ldots, m_i\} \right\rangle$, where $\mathcal{T}_i \subseteq \{1, \ldots, m_i\}$ (here $\mathcal{T}$ stands for torsion) and $e_{ij} \in \mathbb{Z}_{>0}$ (be aware that we explicitly allow $e_{ij} = 1$, which is necessary for our definition of quotient presentations in Section 2). Formally, we put $e_{ij} = \infty$ for $j \notin \mathcal{T}_i$. We call $A = (a_{11}, a_{12}, \ldots, a_{cm_c})$ a *Mal'cev basis associated to the central series* (1). Sometimes we use A interchangeably also for the set $A = \{a_{11}, a_{12}, \ldots, a_{cm_c}\}$.

For convenience, we will also use a simplified notation, in which the generators a_{ij} and exponents e_{ij} are renumbered by replacing each subscript ij with $j + \sum_{\ell<j} m_\ell$, so the generating

sequence A can be written as $A = (a_1, \ldots, a_m)$. We allow the expression ij to stand for $j + \sum_{\ell<j} m_\ell$ in other notations as well. We also denote $\mathcal{T} = \{i \mid e_i < \infty\}$. By the choice of $\{a_1, \ldots, a_m\}$, every element $g \in G$ may be written uniquely in the form $g = a_1^{\alpha_1} \cdots a_m^{\alpha_m}$, where $\alpha_i \in \mathbb{Z}$ and $0 \le \alpha_i < e_i$ whenever $i \in \mathcal{T}$. The m-tuple $(\alpha_1, \ldots, \alpha_m)$ is called the *coordinate vector* or *Mal'cev coordinates* of g and is denoted $\mathrm{Coord}(g)$, and the expression $a_1^{\alpha_1} \cdots a_m^{\alpha_m}$ is called the *(Mal'cev) normal form* of g. We also denote $\alpha_i = \mathrm{Coord}_i(g)$.

To a Mal'cev basis A we associate a presentation of G as follows. For each $1 \le i \le m$, let n_i be such that $a_i \in G_{n_i} \setminus G_{n_i+1}$. If $i \in \mathcal{T}$, then $a_i^{e_i} \in G_{n_i+1}$, hence a relation

$$a_i^{e_i} = a_\ell^{\mu_{i\ell}} \cdots a_m^{\mu_{im}} \tag{2}$$

holds in G for $\mu_{ij} \in \mathbb{Z}$ and $\ell > i$ such that $a_\ell, \ldots, a_m \in G_{n_i+1}$. We call this the *power relation* for a_i. Let $1 \le i < j \le m$. Since the series (1) is central, relations of the form

$$a_j a_i = a_i a_j a_\ell^{\alpha_{ij\ell}} \cdots a_m^{\alpha_{ijm}} \qquad\qquad a_j^{-1} a_i = a_i a_j^{-1} a_\ell^{\beta_{ij\ell}} \cdots a_m^{\beta_{ijm}} \tag{3}$$

hold in G for $\alpha_{ijk}, \beta_{ijk} \in \mathbb{Z}$ and $l > j$ such that $a_\ell, \ldots, a_m \in G_{n_j+1}$. Now, G is the group with generators $\{a_1, \ldots, a_m\}$ subject to the relation of the the form (2)–(3).

A presentation with relations of the form (2)–(3) for all i resp. i and j is called a *nilpotent presentation*. Indeed, any presentation of this form will define a nilpotent group. It is called *consistent* if the order of a_i modulo $\langle a_{i+1}, \ldots, a_m \rangle$ is precisely e_i for all i. While presentations of this form need not, in general, be consistent, those derived from a central series of a group G as above are consistent. Given a consistent nilpotent presentation, there is an easy way to solve the word problem: simply apply the rules of the form (3) to move all occurrences of $a_1^{\pm 1}$ in the input word to the left, then apply the power relations (2) to reduce their number modulo e_1; finally, continue with a_2 and so on.

Multiplication functions. An crucial feature of the coordinate vectors for nilpotent groups is that the coordinates of a product $(a_1^{\alpha_1} \cdots a_m^{\alpha_m})(a_1^{\beta_1} \cdots a_m^{\beta_m})$ may be computed as a "nice" function (polynomial if $\mathcal{T} = \emptyset$) of the integers $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m$.

▶ **Lemma 2** ([7, 10]). *Let G be a nilpotent group with Mal'cev basis $a_1, \ldots, a_m$ and $\mathcal{T} = \emptyset$. There exist $p_1, \ldots, p_m \in \mathbb{Z}[x_1, \ldots, x_m, y_1, \ldots, y_m]$ and $q_1, \ldots, q_m \in \mathbb{Z}[x_1, \ldots, x_m, z]$ such that for $g, h \in G$ with $\mathrm{Coord}(g) = (\gamma_1, \ldots, \gamma_m)$ and $\mathrm{Coord}(h) = (\delta_1, \ldots, \delta_m)$ and $l \in \mathbb{Z}$ we have*

- **(i)** $\mathrm{Coord}_i(gh) = p_i(\gamma_1, \ldots, \gamma_m, \delta_1, \ldots, \delta_m)$,
- **(ii)** $\mathrm{Coord}_i(g^l) = q_i(\gamma_1, \ldots, \gamma_m, l)$,
- **(iii)** $\mathrm{Coord}_1(gh) = \gamma_1 + \delta_1$ *and* $\mathrm{Coord}_1(g^l) = l\gamma_1$.

Notice that an explicit algorithm to construct the polynomials p_i, q_i is given in [14]. For further background on nilpotent groups we refer to [7, 10].

2 Presentation of subgroups

Before we start with algorithmic problems, we introduce a canonical way how to represent subgroups of nilpotent groups. This is important for two reasons: first, of course we need it to solve the subgroup membership problem, and, second, for the uniform setting it allows us to represent nilpotent groups as free nilpotent group modulo a kernel which is represented as a subgroup. Let $h_1, \ldots, h_n$ be elements of G given in normal form by $h_i = a_1^{\alpha_{i1}} \cdots a_m^{\alpha_{im}}$, for

$i = 1, \ldots, n$, and let $H = \langle h_1, \ldots, h_n \rangle$. We associate the *matrix of coordinates*

$$A = \begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1m} \\ \vdots & \ddots & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nm} \end{pmatrix}, \tag{4}$$

to the tuple $(h_1, \ldots, h_n)$ and conversely, to any $n \times m$ integer matrix, we associate an n-tuple of elements of G, whose Mal'cev coordinates are given as the rows of the matrix, and the subgroup H generated by the tuple. For each $i = 1, \ldots, n$ where row i is non-zero, let π_i be the column of the first non-zero entry ('pivot') in row i. The sequence $(h_1, \ldots, h_n)$ is said to be in *standard form* if the matrix of coordinates A is in row-echelon form and its pivot columns are maximally reduced, more specifically, if A satisfies the following properties:

(i) all rows of A are non-zero (i.e. no h_i is trivial),
(ii) $\pi_1 < \pi_2 < \cdots < \pi_s$ (where s is the number of pivots),
(iii) $\alpha_{i\pi_i} > 0$, for all $i = 1, \ldots, n$,
(iv) $0 \leq \alpha_{k\pi_i} < \alpha_{i\pi_i}$, for all $1 \leq k < i \leq s$
(v) if $\pi_i \in \mathcal{T}$, then $\alpha_{i\pi_i}$ divides e_{π_i}, for $i = 1, \ldots, s$.

The sequence (resp. matrix) is called *full* if in addition

(vi) $H \cap \langle a_i, a_{i+1}, \ldots, a_m \rangle$ is generated by $\{h_j \mid \pi_j \geq i\}$, for all $1 \leq i \leq m$.

Note that $\{h_j \mid \pi_j \geq i\}$ consists of those elements having 0 in their first $i-1$ coordinates. It is an easy exercise (see also [16]) to show that 6 holds for a given i if and only if

- for all $1 \leq k < j \leq s$ with $\pi_k < i$, $h_k^{-1} h_j h_k$ and $h_k h_j h_k^{-1}$ are elements of $\langle\, h_l \mid l > k \,\rangle$, and
- for all $1 \leq k \leq s$ with $\pi_k < i$ and $\pi_k \in \mathcal{T}$, $h_k^{e_{\pi_k}/\alpha_{k\pi_k}} \in \langle\, h_l \mid l > k \,\rangle$.

We will use full sequences and the associated matrices in full form interchangeably without mentioning it explicitly. For simplicity we assume that the inputs of algorithms are given as matrices. The importance of full sequences is described in the following lemma – a proof can be found in [24] Propositions 9.5.2 and 9.5.3.

▶ **Lemma 3** ([16, Lem. 3.1]). *Let $H \leq G$. There is a unique full sequence $U = (h_1, \ldots, h_s)$ that generates H. We have $s \leq m$ and $H = \{h_1^{\beta_1} \cdots h_s^{\beta_s} \mid \beta_i \in \mathbb{Z}$ and $0 \leq \beta_i < e_{\pi_i}$ if $\pi_i \in \mathcal{T}\}$.*

Thus, computing a full sequence will be the essential tool for solving the subgroup membership problem. Before we focus on subgroup membership, we will first solve the word problem and introduce how the nilpotent group can be part of the input.

Quotient Mal'cev presentations. Let $c, r \in \mathbb{N}$ be fixed. The free nilpotent group $F_{c,r}$ of class c and rank r is defined as $F_{c,r} = \langle\, a_1, \ldots, a_r \mid [x_1, \ldots, x_{c+1}] = 1 \text{ for } x_1, \ldots, x_{c+1} \in F_{c,r} \,\rangle$ where $[x_1, \ldots, x_{c+1}] = [[x_1, \ldots, x_c], x_{c+1}]$, i. e., $F_{c,r}$ is the r-generated group only subject to the relations that weight $c+1$ commutators are trivial. Throughout, we fix a Mal'cev basis $A = (a_1, \ldots, a_m)$ (which we call the *standard Mal'cev basis*) associated to the lower central series of $F_{c,r}$ such that the associated nilpotent presentation consists only of relations of the form (3) (i. e., $\mathcal{T} = \emptyset$ – such a presentation exists since $F_{c,r}$ is torsion-free), $a_1, \ldots, a_r$ generates $F_{c,r}$, and all other Mal'cev generators are iterated commutators of $a_1, \ldots, a_r$.

Denote by $\mathcal{N}_{c,r}$ the set of r-generated nilpotent groups of class at most c. Every group $G \in \mathcal{N}_{c,r}$ is a quotient of the free nilpotent group $F_{c,r}$, i. e., $G = F_{c,r}/N$ for some normal subgroup $N \leq F_{c,r}$. Assume that $T = (h_1, \ldots, h_s)$ is a full sequence generating N. Adding T to the set of relators of the free nilpotent group yields a new nilpotent presentation. This presentation will be called *quotient presentation* of G. For inputs of algorithms, we assume that a quotient presentation is always given as its matrix of coordinates in full form.

Depending whether the entries of the matrix are encoded in unary or binary, we call the quotient presentation be given in *unary* or *binary*.

▶ **Lemma 4** ([16, Prop. 5.1]). *Let c and r be fixed integers and let $A = (a_1, \dots, a_m)$ be the standard Mal'cev basis of $F_{c,r}$. Moreover, denote by S the set of relators of $F_{c,r}$ with respect to A. Let $G \in \mathcal{N}_{c,r}$ with $G = F_{c,r}/N$ and let T be the full-form sequence for the subgroup N of $F_{c,r}$. Then, $\langle A \mid S \cup T \rangle$ is a consistent nilpotent presentation of G.*

For a proof of Lemma 4 see [21]. For the following we always assume that a quotient presentation is part of the input, but c and r are fixed. Later, we will show how to compute quotient presentations from an arbitrary presentation.

▶ Remark. Lemma 4 ensures that each group element has a unique normal form with respect to the quotient presentation; thus, it guarantees that all our manipulations of Mal'cev coordinates are well-defined.

3 Word problem and computation of Mal'cev coordinates

In this section we deal with the word problem of nilpotent groups, which is well-known to be in TC^0 [23]. Here, we generalize this result by allowing words with binary exponents (recall that *word with binary exponents* is a sequence $w = w_1^{x_1} \cdots w_n^{x_n}$ where $w_i \in \{a_1, \dots, a_m\}$ and the $x_i \in \mathbb{Z}$). By using words with binary exponents the input can be compressed exponentially – making the word problem, a priori, harder to solve. Nevertheless, it turns out that the word problem still can be solved in TC^0 when allowing the input to be given as a word with binary exponents. Note that this contrasts with the situation where the input is given as straight-line program (which like words with binary exponents allow an exponential compression) – then the word problem is complete for the counting class $\mathsf{C_=L}$ [12].

▶ **Theorem 5.** *Let $c, r \geq 1$ be fixed and let $(a_1, \dots, a_m)$ be the standard Mal'cev basis of $F_{c,r}$. The following problem is TC^0-complete: on input of $G \in \mathcal{N}_{c,r}$ given as a binary encoded quotient presentation and a word with binary exponents $w = w_1^{x_1} \cdots w_n^{x_n}$, compute integers $y_1, \dots, y_m$ (in binary) such that $w = a_1^{y_1} \cdots a_m^{y_m}$ in G and $0 \leq y_i < e_i$ for $i \in \mathcal{T}$. Moreover, if the input is given in unary (both G and w), then the output is in unary.*

Note that the statement for unary inputs is essentially the one of [23]. Be aware that in the formulation of the theorem, $\mathcal{T}$ and e_i for $i \in \mathcal{T}$ depend on the input group G. These parameters can be read from the full matrix of coordinates representing G (recall that π_i denotes the column index of the i-th pivot): $\mathcal{T} = \{\pi_i \mid i \in \{1, \dots, n\}\}$ (all columns which have a pivot) and $e_i = \alpha_{ji}$ if $\pi_j = i$. As an immediate consequence of Theorem 5, we obtain:

▶ **Corollary 6.** *Let $c, r \geq 1$ be fixed. The uniform, binary version of the word problem for groups in $\mathcal{N}_{c,r}$ is TC^0-complete (where the input is given as in Theorem 5).*

The proof of Theorem 5 follows the outline given in Section 1.2; however, we cannot apply the rules (2)–(3) one by one. Instead we do only two steps for each generator: first apply all possible rules (3) in one step and then apply the rules (2) in one step.

Proof of Theorem 5. The hardness part is clear since already the word problem of $\mathbb{Z}$ is TC^0-complete. For describing a TC^0 circuit, we proceed by induction along the standard Mal'cev basis $(a_1, \dots, a_m)$ of the free nilpotent group $F_{c,r}$. If w does not contain any letter a_1, we have $y_1 = 0$ and we can compute y_i for $i > 1$ by induction.

Otherwise, we rewrite w as $a_1^{y_1}uv$ (with $0 \leq y_1 < e_1$ if $1 \in \mathcal{T}$) such that u and v are words with binary exponents not containing any a_1s. Once this is completed, uv can be rewritten as $a_2^{y_2} \cdots a_m^{y_m}$ by induction. For computing y_1, u and v, we proceed in two steps:

First, we rewrite w as $a_1^{\tilde{y}_1}v$ with $\tilde{y}_1 = \sum_{w_i=a_1} x_i$ (this is possible by Lemma 2 (iii)). The exponent $\tilde{y}_1$ can be computed by iterated addition, which by Theorem 1 is in TC^0 (in the unary case $\tilde{y}_1$ can be written down in unary). Now, v consists of what remains from w after a_1 has been "eliminated": for every position i in w with $w_i \neq a_1$, we compute $z_i = \sum_{w_j=a_1}^{j>i} x_j$ using iterated addition. Let $w_i = a_k$. By Lemma 2 (i) there are fixed polynomials $p_{k,k+1}, \ldots, p_{k,m} \in \mathbb{Z}[x,y]$ such that in the free nilpotent group holds $a_k^x a_1^y = a_1^y a_k^x a_{k+1}^{p_{k,k+1}(x,y)} \cdots a_m^{p_{k,m}(x,y)}$ for all $x, y \in \mathbb{Z}$. Hence, in order to obtain $\tilde{w}$, it remains to replace every $w_i^{x_i}$ with $w_i = a_1$ by the empty word and every $w_i^{x_i}$ with $w_i = a_k \neq a_1$ by $a_k^{x_i} a_{k+1}^{p_{k,k+1}(x_i,z_i)} \cdots a_m^{p_{k,m}(x_i,z_i)}$, which is a word with binary exponents (resp. as a word of polynomial length in the unary case), for $k = 2, \ldots, m$. The exponents can be computed in TC^0 by Theorem 1. Since the $p_{k,i}$ are bounded by polynomials, in the unary case, $a_k^{x_i} a_{k+1}^{p_{k,k+1}(x_i,z_i)} \cdots a_m^{p_{k,m}(x_i,z_i)}$ can be written as a word without exponents.

The second step is only applied if $1 \in \mathcal{T}$ (as explained above, this can be decided and e_i can be read directly from the quotient presentation by checking whether there is a pivot in the first column) – otherwise $y_1 = \tilde{y}_1$ and u is the empty word. We rewrite $a_1^{\tilde{y}_1}$ to $a_1^{y_1}u$ with $y_1 = \tilde{y}_1 \bmod e_1$ and a word with binary exponents u not containing any a_1. Again y_1 can be computed in TC^0 by Theorem 1. Let $a_1^{e_1} = a_2^{\mu_{12}} \cdots a_m^{\mu_{1m}}$ be the power relation for a_1 (which can be read from the quotient presentation – it is just the row where the pivot is in the first column) and write $\tilde{y}_1 = s \cdot e_1 + y_1$. Now, u should be equal to $(a_2^{\mu_{12}} \cdots a_m^{\mu_{1m}})^s$ in $F_{c,r}$. We use the fixed polynomials $q_i \in \mathbb{Z}[x_1, \ldots, x_m, z]$ from Lemma 2 (ii) for $F_{c,r}$ yielding $u = a_2^{q_2(0,\mu_{12},\ldots,\mu_{1m},s)} \cdots a_m^{q_m(0,\mu_{12},\ldots,\mu_{1m},s)}$ (which, in the binary setting, is a word with binary exponents, and in the unary setting a word without exponents of polynomial length). Now, we have $w = a_1^{y_1}uv$ in G as desired. ◀

4 Matrix reduction and subgroup membership problem

Before we solve the subgroup membership problem, let us take a look at one essential step, namely the problem of computing greatest common divisors. Indeed, consider the nilpotent group $\mathbb{Z}$ and let $a, b, c \in \mathbb{Z}$. Then $c \in \langle a, b \rangle$ if, and only if, $\gcd(a, b) \mid c$.

Binary gcds. The *extended gcd problem* (ExtGCD) is the following problem: on input of binary encoded numbers $a_1, \ldots, a_n \in \mathbb{Z}$, compute $x_1, \ldots, x_n \in \mathbb{Z}$ such that $x_1a_1 + \cdots + x_na_n = \gcd(a_1, \ldots, a_n)$. Clearly this can be done in P using the Euclidean algorithm, but it is not known whether it is actually in NC. Since we need to compute greatest common divisors, we will reduce the subgroup membership problem to the computation of gcds.

Unary gcds. Computing the gcd of numbers encoded in unary is straightforward in TC^0 by an exhaustive search. Also, for just two numbers $a, b \in \mathbb{Z}$ the gcd easily can be expressed as a linear combination in TC^0: there are $x, y \leq \max\{|a|, |b|\}$ such that $ax + by = \gcd(a, b)$. Now, x, y can be computed in TC^0 by simply checking all values with $|x|, |y| \leq \max\{|a|, |b|\}$. Similarly, there are $x_1, \ldots, x_n \leq |\max\{|a_1|, \ldots, |a_n|\}|$ with $x_1a_1 + \cdots + x_na_n = \gcd(a_1, \ldots, a_n)$. However, for computing these x_i, we cannot check all possible combinations of values in TC^0 because there are $|\max\{|a_1|, \ldots, |a_n|\}|^n$ (i. e., exponentially) many. Expressing the gcd as a linear combination can be viewed as a linear equation with integral coefficients. Recently, in [5, Thm. 3.14] it has been shown that, if all the coefficients are given in unary, it can be

decided in TC^0 whether such an equation or a system of a fixed number of equations has a solution. Since from the proof of [5, Thm. 3.14] it is not obvious how to find an actual solution, we prove the following result in our full version on arXiv [21]:

▶ **Proposition 7.** *The following problem is in TC^0: Given integers $a_1, \dots, a_n$ in unary, compute $x_1, \dots, x_n \in \mathbb{Z}$ (either in unary or binary) such that $x_1a_1 + \cdots + x_na_n = \gcd(a_1, \dots, a_n)$ and $|x_i| \leq (n+1)\left(\max\{|a_1|, \dots, |a_n|\}\right)^2$.*

Matrix reduction. The matrix reduction procedure converts an arbitrary matrix of coordinates into its full form and, thus, is an essential step for solving the subgroup membership problem and several other problems. It was first described in [24] – however, without a precise complexity estimate. In this section, we repeat the presentation from [16] and show that for fixed c and r, it can be actually computed uniformly for groups in $\mathcal{N}_{c,r}$ in TC^0 – in the case that the inputs are given in unary (as words). If the inputs are represented as words with binary exponents, then we still can show that it is TC^0-Turing-reducible to ExtGCD. In Section 2, we defined the matrix representation of subgroups of nilpotent groups. We adopt all notation from Section 2.

As before, let $c, r \in \mathbb{N}$ be fixed and let $(a_1, \dots, a_m)$ be the standard Mal'cev basis of $F_{c,r}$. Let $G \in \mathcal{N}_{c,r}$ be given as quotient presentation, i. e., as a matrix in full form (either with unary or binary coefficients). We define the following operations on tuples $(h_1, \dots, h_n)$ (our subgroup generators) of elements of G and the corresponding operations on the associated matrix, with the goal of converting $(h_1, \dots, h_n)$ to a sequence in full form generating the same subgroup $H = \langle h_1, \dots, h_n \rangle$:

1. Swap h_i with h_j. This corresponds to swapping row i with row j.
2. Replace h_i by $h_i h_j^l$ ($i \neq j$, $l \in \mathbb{Z}$). This corresponds to replacing row i by $\mathrm{Coord}(h_i h_j^l)$.
3. Add or remove a trivial element from the tuple. This corresponds to adding or removing a row of zeros; or (3') a row of the form $(0 \ \dots \ 0 \ e_i \ \alpha_{i+1} \ \dots \ \alpha_m)$, where $i \in \mathcal{T}$ and $a_i^{-e_i} = a_{i+1}^{\alpha_{i+1}} \cdots a_m^{\alpha_m}$.
4. Replace h_i with h_i^{-1}. This corresponds to replacing row i by $\mathrm{Coord}(h_i^{-1})$.
5. Append an arbitrary product $h_{i_1}^{l_1} \cdots h_{i_k}^{l_k}$ with $i_1, \dots, i_k \in \{1, \dots, n\}$ and $l_1, \dots, l_k \in \mathbb{Z}$ to the tuple: add a new row with $\mathrm{Coord}(h_{i_1}^{l_1} \cdots h_{i_k}^{l_k})$.

Clearly, all these operations preserve H.

▶ **Lemma 8.** *On input of a quotient presentation of $G \in \mathcal{N}_{c,r}$ in unary (resp. binary) and a matrix of coordinates A given in unary (resp. binary), operations (1)–(5) can be done in TC^0. The output matrix will be also encoded in unary (resp. binary). For operations (2) and (5), we require that the exponents l, $l_1, \dots, l_k$ are given in unary (resp. binary).*

Moreover, as long as the rows in the matrix which are changed are pairwise distinct, a polynomial number of such steps can be done in parallel in TC^0.

Proof. Operations (1) and (3), clearly can be done in TC^0. Notice that operation (3') means simply that a row of the quotient presentation of G is appended to the matrix.

In the unary case, it follows directly from Theorem 5 that operations (2), (4), and (5) are in TC^0 because, since l, $l_1, \dots, l_k$ are given in unary, the respective group elements can be written down as words.

In the case of binary inputs, (5) works as follows ((2) and (4) analogously): by Lemma 2 (ii), there are functions $q_1, \dots, q_m \in \mathbb{Z}[x_1, \dots, x_m, z]$ such that for every $h \in F_{c,r}$ with $\mathrm{Coord}(h) = (\gamma_1, \dots, \gamma_m)$ anda $l \in \mathbb{Z}$, we have $\mathrm{Coord}_i(h^l) = q_i(\gamma_1, \dots, \gamma_m, l)$ in $F_{c,r}$. These functions can be used to compute $\mathrm{Coord}(h_{i_j}^{l_j})$ for $j = 1, \dots, k$. After that, $h_{i_1}^{l_1} \cdots h_{i_k}^{l_k}$ can be written down as word with binary exponents and Theorem 5 can be applied. ◀

Using the row operations defined above, in [16] it is shown how to reduce any coordinate matrix to its unique full form. Let us repeat these steps:

Let A_0 be a matrix of coordinates, as in (4) in Section 2. Recall that π_k denotes the column index of the k-th pivot (of the full form of A_0). We produce matrices $A_1, \ldots, A_s$, where s is the number of pivots in the full form of A_0, such that for every $k = 1, \ldots, s$ the first π_k columns of A_k form a matrix satisfying conditions 2-5 of being a full sequence, condition 6 is satisfied for all $i < \pi_{k+1}$, and A_s is the full form of A_0. Here we formally denote $\pi_{s+1} = m + 1$. Set $\pi_0 = 0$ and assume that A_{k-1} has been constructed for some $k \geq 1$. In the steps below we construct A_k. We let n and m denote the number of rows and columns, respectively, of A_{k-1}. At all times during the computation, h_i denotes the group element corresponding to row i of A_k and α_{ij} denotes the (i,j)-entry of A_k, which is $\mathrm{Coord}_j(h_i)$. These may change after every operation.

Step 1. Locate the column π_k of the next pivot, which is the minimum integer $\pi_{k-1} < \pi_k \leq m$ such that $\alpha_{i\pi_k} \neq 0$ for at least one $k \leq i \leq n$. If no such integer exists, then $k-1 = s$ and A_s is already constructed. Otherwise, set A_k to be a copy of A_{k-1} and denote $\pi = \pi_k$. Compute a linear expression of $d = \gcd(\alpha_{k\pi}, \ldots, \alpha_{n\pi}) = l_k\alpha_{k\pi} + \cdots + l_n\alpha_{n\pi}$. Let $h_{n+1} = h_k^{l_k} \cdots h_n^{l_n}$ and note that h_{n+1} has coordinates of the form $\mathrm{Coord}(h_{n+1}) = (0, \ldots, 0, d, \ldots)$ with d occurring in position π. Perform operation 5 to append h_{n+1} as row $n+1$ of A_k.

Step 2. For each $i = k, \ldots, n$, perform operation 2 to replace row i by $\mathrm{Coord}(h_i \cdot h_{n+1}^{-\alpha_{i\pi}/d})$. and for each $i = 1, \ldots, k-1$, use 2 to replace row i by $\mathrm{Coord}(h_i \cdot h_{n+1}^{-\lfloor \alpha_{i\pi}/d \rfloor})$. After that, swap row k with row $n+1$ using 1. At this point, properties 2-4 hold on the first k columns of A_k.

Step 3. If $\pi \in \mathcal{T}$, we additionally ensure condition 5 as follows. Perform row operation (3'), with respect to π, to append a trivial element h_{n+2} with $\mathrm{Coord}(h_{n+2}) = (0, \ldots, 0, e_\pi, \ldots)$ to A_k. Let $\delta = \gcd(d, e_\pi)$ and compute the linear expression $\delta = n_1 d + n_2 e_\pi$, with $|n_1|, |n_2| \leq \max\{d, e_\pi\}$. Let $h_{n+3} = h_k^{n_1} h_{n+2}^{n_2}$ and append this row to A_k, as row $n+3$. Note that $\mathrm{Coord}(h_{n+3}) = (0, \ldots, 0, \delta, \ldots)$, with δ in position π. Replace row k by $\mathrm{Coord}(h_k \cdot h_{n+3}^{-d/\delta})$ and row $n+2$ by $\mathrm{Coord}(h_{n+2} \cdot h_{n+3}^{-e_\pi/\delta})$, producing zeros in column π in these rows. Swap row k with row $n+3$. At this point, 2, 3, and 5 hold (for the first π_k columns) but 4 need not, since the pivot entry is now δ instead of d. For each $j = 1, \ldots, k-1$, replace row j by $\mathrm{Coord}(h_j \cdot h_k^{-\lfloor \alpha_{j\pi}/\delta \rfloor})$, ensuring 4.

Step 4. Identify the next pivot π_{k+1} (like in Step 1). If π_k is the last pivot, we set $\pi_{k+1} = m+1$. We now ensure condition 6 for $i < \pi_{k+1}$. Observe that Steps 1-3 preserve $\langle\, h_j \mid \pi_j \geq i \,\rangle$ for all $i < \pi_k$. Hence 6 holds in A_k for $i < \pi_k$ since it holds in A_{k-1} for the same range. Now consider i in the range $\pi_k \leq i < \pi_{k+1}$. It suffices to establish (vi.i) for all $j > k$ and (vi.ii) for π_k only. To obtain (vi.i), notice that $h_k^{-1} h_j h_k, h_k h_j h_k^{-1} \in \langle\, h_\ell \mid \ell > k \,\rangle$ if, and only if, $[h_j, h_k^{\pm 1}] \in \langle\, h_\ell \mid \ell > k \,\rangle$. Further, note that the subgroup generated by $S_j = \{1, h_j, [h_j, h_k], \ldots, [h_j, h_k, \ldots, h_k]\}$, where h_k appears $m - \pi_k$ times in the last commutator, is closed under commutation with h_k since if h_k appears more than $m - \pi_k$ times then the commutator is trivial. An inductive argument shows that the subgroup $\langle S_j \rangle$ coincides with $\langle h_k^{-\ell} h_j h_k^{\ell} \mid 0 \leq \ell \leq m - \pi_k \rangle$. Similar observations can be made for conjugation by h_k^{-1}. Therefore, appending via operation 5 rows $\mathrm{Coord}(h_k^{-\ell} h_j h_k^{\ell})$ for all $1 \leq |\ell| \leq m - \pi_k$ and all $k < j \leq n+3$ delivers (vi.i) for all $j > k$. Note that (vi.i) remains true for $i < \pi_k$.

To obtain (vi.ii), in the case $\pi_k \in \mathcal{T}$, we add row $\mathrm{Coord}(h_k^{e_k/\alpha_{k\pi_k}})$. Note that this element commutes with h_k and therefore (vi.i) is preserved.

Step 5. Using operation 3, eliminate all zero rows. The matrix A_k is now constructed.

We have to show that each step can be performed in TC^0 given that all Mal'cev coordinates

are encoded in unary (resp. in $\mathsf{TC}^0(\text{ExtGCD})$ if Mal'cev coordinates are encoded in binary). Since the total number of steps is constant (only depending on the nilpotency class and number of generators), this gives a TC^0 (resp. $\mathsf{TC}^0(\text{ExtGCD})$) circuit for computing the full form of a given subgroup.

Step 1. The next pivot can be found in TC^0 since it is simply the next column in the matrix with a non-zero entry, which can be found as a simple Boolean combination of test whether the entries are zero. In the unary case, by Proposition 7, $d = \gcd(\alpha_{k\pi}, \dots, \alpha_{n\pi})$ can computed in TC^0 together with $l_k, \dots, l_n$ encoded in unary such that $d = l_k\alpha_{k\pi} + \dots + l_n\alpha_{n\pi}$. Now, by Lemma 8, Step 1 can be done in TC^0.

In the binary case, d and $l_k, \dots, l_n$ can be computed using ExtGCD. Hence, by Lemma 8, Step 1 can be done in $\mathsf{TC}^0(\text{ExtGCD})$.

Step 2. The numbers $\lfloor \alpha_{i\pi}/d \rfloor$ (either in unary or binary) can be computed in TC^0 for all i in parallel by Theorem 1. After that one operation (2) is applied to each row of the matrix. By Lemma 8, this can be done in parallel for all rows in TC^0. Finally, swapping rows k and $n+1$ can be done in TC^0.

Step 3. As explained in Section 3, $\mathcal{T}$ and e_i for $i \in \mathcal{T}$ can be read directly from the quotient presentation. Thus, it can be decided in TC^0 whether Step 3 has to be executed. Appending a new row is in TC^0. Computing $\gcd(d, e_\pi) = d = n_1 d n_2 e_\pi$ is in TC^0 by Proposition 7 (in the unary case) and in $\mathsf{TC}^0(\text{ExtGCD})$ in the binary case. After that one operation (5) is followed by two operations (2), one operation (1), and, finally, $k-1$ times operation (2), which all can be done in TC^0 again by Lemma 8.

Step 4. The next pivot can be found in TC^0 as outlined in Step 1. After that, Step 4 consists of an application of a constant number (only depending on the nilpotency class and number of generators) of operations (5) and thus, by Lemma 8, is in TC^0.

Step 5. Clearly that is in TC^0.

Thus, we have completed the proof of our main result:

▶ **Theorem 9.** *Let $c, r \in \mathbb{N}$ be fixed. The following problem is in TC^0: given a unary encoded quotient presentation of $G \in \mathcal{N}_{c,r}$ and $h_1, \dots, h_n \in G$, compute the full form of the associated matrix of coordinates encoded in unary and hence the unique full-form sequence $(g_1, \dots, g_s)$ generating $\langle h_1, \dots, h_n \rangle$. Moreover, if the G and $h_1, \dots, h_n$ are given in binary, then the full-form sequence with binary coefficients can be computed in $\mathsf{TC}^0(\text{ExtGCD})$.*

Subgroup membership problem. As an easy application of the matrix reduction we can solve the subgroup membership problem in TC^0 – for a proof details see [21].

▶ **Corollary 10.** *Let $c, r \in \mathbb{N}$ be fixed. The following problem is in TC^0 (resp. $\mathsf{TC}^0(\text{ExtGCD})$ for binary inputs): given a quotient presentation of $G \in \mathcal{N}_{c,r}$, elements $h_1, \dots, h_n \in G$ and $h \in G$, decide whether or not h is an element of the subgroup $H = \langle h_1, \dots, h_n \rangle$.*

Moreover, if $h \in H$, the circuit computes the unique expression $h = g_1^{\gamma_1} \cdots g_s^{\gamma_s}$ where $(g_1, \dots, g_s)$ is the full-form sequence for H with the γ_i encoded in unary (resp. binary).

Alternatively, for unary inputs, the output can be given as word $h = h_{i_1}^{\epsilon_1} \cdots h_{i_t}^{\epsilon_t}$ where $i_j \in \{1, \dots, n\}$ and $\epsilon_j = \pm 1$.

Note that we do not know whether there is an analog of the second type of output for binary inputs. A possible way of expressing the output would be as a word with binary exponents over $h_1, \dots, h_n$. However, simply applying the same procedure as for unary inputs will not lead to a word with binary exponents.

Subgroup presentations. The full-form sequence associated to a subgroup H forms a Mal'cev basis for H. This allows us to compute a consistent nilpotent presentation for H. Note, however, that the resulting presentation is *not* a quotient presentation (although it can be transformed into one, see Proposition 14) – partly this is due to the fact that, in general, $H \notin \mathcal{N}_{c,r}$. The following is the TC^0 version of [16, Thm. 3.11]:

▶ **Corollary 11.** *Let $c, r \in \mathbb{N}$ be fixed. The following is in TC^0 for unary inputs and in $\mathsf{TC}^0(\text{ExtGCD})$ for binary inputs:*

Input: *a quotient presentation for $G \in \mathcal{N}_{c,r}$ and elements $h_1, \dots, h_n \in G$.*

Output: *a consistent nilpotent presentation for $H = \langle h_1, \dots, h_n \rangle$ given by a list of generators $(g_1, \dots, g_s)$ and numbers $\mu_{ij}, \alpha_{ijk}, \beta_{ijk} \in \mathbb{Z}$ encoded in unary (resp. binary) for $1 \le i < j < k \le s$ representing the relations (2)-(3).*

5 More algorithmic problems

The next two theorems are applications of Theorem 9. Their proofs (in [21]) follow essentially the proofs of their counterparts Theorems 4.1 and 4.6 in [16].

▶ **Theorem 12** (Kernels and preimages)**.** *Let $c, r \in \mathbb{N}$ be fixed. The following is in TC^0 for unary inputs and in $\mathsf{TC}^0(\text{ExtGCD})$ for binary inputs: On input of*

- *$G, H \in \mathcal{N}_{c,r}$ given as quotient presentations,*
- *a subgroup $K = \langle g_1, \dots, g_n \rangle \le G$,*
- *a list of elements $h_1, \dots, h_n$ defining a homomorphism $\varphi : K \to H$ via $\varphi(g_i) = h_i$, and*
- *optionally, an element $h \in H$ guaranteed to be in the image of φ,*

compute a generating set X for the kernel of φ, and an element $g \in G$ such that $\varphi(g) = h$.

In case of unary inputs, X and g will be returned as words, and for binary inputs, as words with binary exponents.

▶ **Theorem 13** (Conjugacy Problem)**.** *Let $c, r \in \mathbb{N}$ be fixed. The following is in TC^0 for unary inputs and in $\mathsf{TC}^0(\text{ExtGCD})$ for binary inputs: On input of some $G \in \mathcal{N}_{c,r}$ given as quotient presentation and elements $g, h \in G$, either produce some $u \in G$ such that $g = u^{-1}hu$, or determine that no such element u exists. In case of unary inputs, u will be returned as a word, for binary inputs, as a word with binary exponents.*

Computing quotient presentations. The results in the previous sections always required that the group is given as a quotient presentation. However, we can use Theorem 9 to transform an arbitrary presentation with at most r generators of a group in $\mathcal{N}_{c,r}$ into a quotient presentation. For a proof see [21].

▶ **Proposition 14.** *Let c and r be fixed integers. The following is in TC^0: given an arbitrary finite presentation with generators $a_1, \dots, a_r$ of a group $G \in \mathcal{N}_{c,r}$ (as a list of relators given as words over $\{a_1, \dots, a_r\}^{\pm 1}$), compute a quotient presentation of G (encoded in unary) and an explicit isomorphism. Moreover, if the relators are given as words with binary exponents, then the binary encoded quotient presentation can be computed in $\mathsf{TC}^0(\text{ExtGCD})$.*

▶ Remark. Because of Proposition 14, in all theorems above where the input is a quotient presentation, we can also take an arbitrary r-generated presentation of a group in $\mathcal{N}_{c,r}$ as input. However, be aware that for the word problem (Theorem 5 and Corollary 6) the complexity changes from TC^0 to $\mathsf{TC}^0(\text{ExtGCD})$ in the binary case.

Conclusion and Open Problems. We have seen that most problems which in [16] were shown to be in LOGSPACE indeed are in TC^0 even in the uniform setting where the number of generators and nilpotency class is fixed. Moreover, their binary versions are in $\mathsf{TC}^0(\text{ExtGCD})$ meaning that nilpotent groups are no more complicated than abelian groups in many algorithmic aspects. This contrasts with the slightly larger class of polycyclic groups: there the word problem is still in TC^0 [23, 12], but the conjugacy problem is not even known to be in NP. We conclude with some possible generalizations of our results:

- Does a uniform version of Theorem 5 hold (i. e., is the uniform word problem still in TC^0) for fixed nilpotency class but an arbitrary number of generators? What happens to the complexity if also the nilpotency class is part of the input? Note that in that case it is even not clear whether the word problem is still in polynomial time.
- Is there a way to solve the conjugacy problem for nilpotent groups with binary exponents in TC^0? Notice that we needed to compute gcds to solve the subgroup membership problem. However, the conjugacy problem might be solved using another method.
- What is the complexity of the uniform conjugacy problem with arbitrary nilpotency class?

References

1 Norman Blackburn. Conjugacy in nilpotent groups. *Proceedings of the American Mathematical Society*, 16(1):143–148, 1965.

2 William W. Boone. The Word Problem. *Ann. of Math.*, 70(2):207–265, 1959.

3 Max Dehn. Ueber unendliche diskontinuierliche Gruppen. *Math. Ann.*, 71:116–144, 1911.

4 Bettina Eick and Delaram Kahrobaei. Polycyclic groups: A new platform for cryptology? *ArXiv Mathematics e-prints*, 2004. `arXiv:math/0411077`.

5 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:128, 2011. URL: `http://eccc.hpi-web.de/report/2011/128`.

6 Alberd Garreta, Alexei Miasnikov, and Denis Ovchinnikov. Properties of random nilpotent groups. *ArXiv e-prints*, December 2016. `arXiv:1612.01242`.

7 Philip Hall. *The Edmonton notes on nilpotent groups*. Queen Mary College Mathematics Notes. Mathematics Department, Queen Mary College, London, 1969.

8 William Hesse. Division is in uniform TC^0. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 104–114. Springer, 2001. `doi:10.1007/3-540-48224-5_9`.

9 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.

10 Mikhail I. Kargapolov and Ju. I. Merzljakov. *Fundamentals of the theory of groups*, volume 62 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1979. Translated from the second Russian edition by Robert G. Burns.

11 Mikhail I. Kargapolov, Vladimir. N. Remeslennikov, N. S. Romanovskii, Vitaly A. Roman'kov, and V. A. Čurkin. Algorithmic questions for σ-powered groups. *Algebra i Logika*, 8:643–659, 1969.

12 Daniel König and Markus Lohrey. Evaluating matrix circuits. In *Computing and combinatorics*, volume 9198 of *Lecture Notes in Comput. Sci.*, pages 235–248. Springer, Cham, 2015. `doi:10.1007/978-3-319-21398-9_19`.

13 Klaus-Jörn Lange and Pierre McKenzie. On the complexity of free monoid morphisms. In Kyung-Yong Chwa and Oscar H. Ibarra, editors, *Algorithms and Computation, 9th International Symposium, ISAAC'98, Taejon, Korea, December 14-16, 1998, Proceedings*,

volume 1533 of *Lecture Notes in Computer Science*, pages 247–256. Springer, 1998. doi:10.1007/3-540-49381-6_27.

14 Charles. R. Leedham-Green and Leonard H. Soicher. Symbolic collection using Deep Thought. *LMS J. Comput. Math.*, 1:9–24 (electronic), 1998. doi:10.1112/S1461157000000127.

15 Richard J. Lipton and Yechezkel Zalcstein. Word problems solvable in logspace. *J. ACM*, 24(3):522–526, July 1977. doi:10.1145/322017.322031.

16 Jeremy MacDonald, Alexei G. Myasnikov, Andrey Nikolaev, and Svetla Vassileva. Logspace and compressed-word computations in nilpotent groups. *CoRR*, abs/1503.03888, 2015. URL: http://arxiv.org/abs/1503.03888.

17 Anatoly I. Mal'cev. On homomorphisms onto finite groups. *Transl., Ser. 2, Am. Math. Soc.*, 119:67–79, 1983. Translation from Uch. Zap. Ivanov. Gos. Pedagog Inst. 18, 49-60 (1958).

18 Andrzej Mostowski. Computational algorithms for deciding some problems for nilpotent groups. *Fundamenta Mathematicae*, 59(2):137–152, 1966. URL: http://eudml.org/doc/213887.

19 Alexei Myasnikov, Andrey Nikolaev, and Alexander Ushakov. The Post correspondence problem in groups. *J. Group Theory*, 17(6):991–1008, 2014. doi:10.1515/jgth-2014-0022.

20 Alexei Myasnikov, Andrey Nikolaev, and Alexander Ushakov. Non-commutative lattice problems. *J. Group Theory*, 19(3):455–475, 2016. doi:10.1515/jgth-2016-0506.

21 Alexei G. Myasnikov and Armin Weiß. TC^0 circuits for algorithmic problems in nilpotent groups. *CoRR*, abs/1702.06616, 2017. URL: http://arxiv.org/abs/1702.06616.

22 Pyotr S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *Trudy Mat. Inst. Steklov*, pages 1–143, 1955. In Russian.

23 David Robinson. *Parallel Algorithms for Group Word Problems*. PhD thesis, University of California, San Diego, 1993.

24 Charles C. Sims. *Computation with finitely presented groups*, volume 48 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1994. doi:10.1017/CBO9780511574702.

25 Heribert Vollmer. *Introduction to Circuit Complexity*. Springer, Berlin, 1999.

Better Complexity Bounds for Cost Register Automata*

Eric Allender[1], Andreas Krebs[2], and Pierre McKenzie[3]

1 **Department of Computer Science, Rutgers University, Piscataway, NJ, USA**
allender@cs.rutgers.edu
2 **WSI, Universität Tübingen, Germany**
mail@krebs-net.de
3 **DIRO, Université de Montréal, Québec, Canada**
mckenzie@iro.umontreal.ca

Abstract

Cost register automata (CRAs) are one-way finite automata whose transitions have the side effect that a register is set to the result of applying a state-dependent semiring operation to a pair of registers. Here it is shown that CRAs over the tropical semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$ can simulate polynomial time computation, proving along the way that a naturally defined width-k circuit value problem over the tropical semiring is P-complete. Then the copyless variant of the CRA, requiring that semiring operations be applied to distinct registers, is shown no more powerful than NC^1 when the semiring is $(\mathbb{Z}, +, \times)$ or $(\Gamma^* \cup \{\perp\}, \max, \text{concat})$. This relates questions left open in recent work on the complexity of CRA-computable functions to long-standing class separation conjectures in complexity theory, such as NC versus P and NC^1 versus GapNC^1.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases computational complexity, cost registers

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.24

1 Introduction

A weighted finite automaton on a given input computes the sum, over every computation path, of the product, over the transitions encountered along that path, of the semiring elements assigned to those transitions. Weighted automata have a long history and extensive theoretical support (see [17]) but their utility for the purpose of computer-aided verification is limited. This motivated Alur and his co-authors to introduce the streaming string transducer [3], soon followed by the cost register automaton (CRA) [5].

CRAs are deterministic and are yet strictly more expressive than weighted automata [5]. A CRA computes a so-called *regular function* from strings to a cost domain. (This should not be confused with Colcombet's regular cost functions, which are intended to capture asymptotic behavior [13].) A "copyless" variant (CCRA) of the CRA has the expressivity of single-valued weighted automata [5, Thm 4]. Another variant of CRAs restricts the multiplicative operation, by only allowing multiplication by constants; this model has the full expressivity of weighted automata [5, Thm 9]. A theory of CRAs, largely concerned with expressivity and decidability properties, was developed in a series of papers, including [5, 7, 6].

* Supported by NSF grant CCF-1555409 (Allender), by the DFG Emmy-Noether program KR 4042/2 (Krebs), and by the Natural Sciences and Engineering Research Council of Canada (McKenzie)

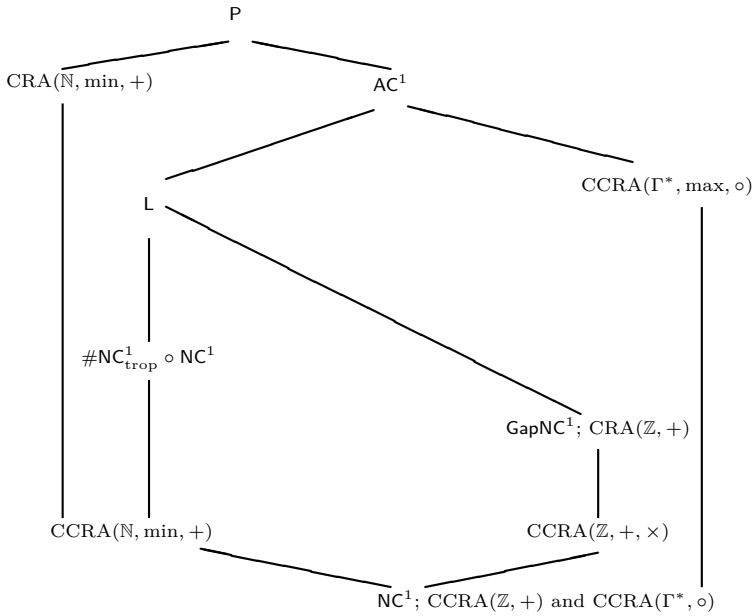


Figure 1 Prior state of knowledge, from [1]. When a class of CRA functions and a complexity class appear together, it means that containment of the CRA class in the complexity class is tight, since some of the CRA functions are complete for the complexity class. (Definitions of the complexity classes can be found in Section 2.)

None of the above work considered the computational complexity of the functions expressed by CRAs. Yet the CRA model is interesting from that viewpoint because it combines a parallelizable component (logarithmic depth boolean circuits indeed recognize regular languages) with a less structured component that builds and evaluates expressions over the cost domain. The three variants of the CRA discussed above (CRAs, CCRAs, and CRAs with restricted multiplication) in fact are reminiscent of three variants of algebraic circuits (general, tree-like, and skew). This raises the question of whether CRA variants over various domains capture interesting complexity classes, such as the (functional) class P and subclasses of NC. These considerations prompted Allender and Mertz to develop complexity bounds for the functions computable by CRAs and CCRAs [1]. This line of inquiry for various models was also pursued and extended in [22, 25, 24, 16, 15, 12].

The main results obtained by Allender and Mertz are depicted on Figure 1. Most results involve the (weaker) CCRA model with integer arithmetic, but also with "tropical" arithmetic, that is, over domains such as $(\mathbb{N} \cup \{infty\}, \min, +)$ and $(\Gamma^*, \max, \circ)$. We note that tropical semirings arise frequently in the study of weighted automata (see for instance [14, Sect. 1.1] and [5, Thm 9]). Computationally, min and max are "forgetful" operations that should intuitively lend themselves to simpler simulations.

Our contribution here is to improve some of the bounds from [1]. In particular, the closing section of [1] listed the following four open questions:

- Are there any CCRA functions over $(\mathbb{Z}, +, \times)$ that are complete for GapNC^1?
- Are there any CCRA functions over the tropical semiring that are hard for $\#\mathsf{NC}^1_{\text{trop}}$?
- The gap between the upper and lower bounds for CCRA functions over $(\Gamma^*, \max, \circ)$ is quite large (NC^1 versus $\mathsf{OptLogCFL} \subseteq \mathsf{AC}^1$). Can this be improved?
- Is there an NC upper bound for CRA functions (without the copyless restriction) over the tropical semiring?

We essentially answer all of these questions, modulo long-standing open questions in complexity theory. We show that CCRA functions over each of $(\mathbb{Z}, +, \times)$, $(\Gamma^*, \max, \circ)$, and the tropical semiring are all computable in NC^1. We thus give the improvement asked for in the third question, and we show that the answers to the first two questions are equivalent to $\mathsf{NC}^1 = \mathsf{GapNC}^1$ and $\mathsf{NC}^1 = \#\mathsf{NC}^1_{\text{trop}}$, respectively. We also provide a negative answer to the fourth question (assuming $\mathsf{NC} \neq \mathsf{P}$), by reducing a P-complete problem to the computation of a CRA function over the tropical semiring. It follows from the latter that for any k larger than a small constant, the width-k circuit value problem over structures such as $(\mathbb{N}, \max, +)$ and $(\mathbb{N}, \min, +)$ is P-complete under AC^0-Turing reductions. (See Section 2 for the precise definition of the problem and then Corollary 5.) Figure 2 summarizes our results.

2 Preliminaries

We assume familiarity with some common complexity classes and with basic notions of circuit complexity, such as can be found in any textbook on complexity theory.

Recall that a language $A \subseteq \{0,1\}^*$ is accepted by a Boolean circuit family $(C_n)_{\in\mathbb{N}}$ if for all x it holds that $x \in A$ iff $C_{|x|}(x) = 1$. Circuit families encountered in this paper will be *uniform*. Uniformity is a somewhat technical issue because of subtleties encountered at low complexity levels. We will not be concerned with such subtleties, and thus we refer the reader to a standard text (such as [29, Sect. 4.5]) for a precise definition of what it means for a circuit family $(C_n)_{n\geq 0}$ to be U_E-uniform. (Informally, this notion of uniformity means that there is a linear-time machine that takes inputs of the form (n, g, h, p) and determines if p encodes a path from gate h to gate g in C_n, and also determines what type of gate g and h are.) We will encounter the following circuit complexity classes.

- $\mathsf{NC}^i = \{A : A$ is accepted by a U_E-uniform family of circuits of bounded fan-in AND, OR and NOT gates, having size $n^{O(1)}$ and depth $O(\log^i n)\}$.
- $\mathsf{AC}^i = \{A : A$ is accepted by a U_E-uniform family of circuits of unbounded fan-in AND, OR and NOT gates, having size $n^{O(1)}$ and depth $O(\log^i n)\}$.
- $\mathsf{TC}^i = \{A : A$ is accepted by a U_E-uniform family of circuits of unbounded fan-in MAJORITY gates, having size $n^{O(1)}$ and depth $O(\log^i n)\}$.

We remark that, for constant-depth classes such as AC^0 and TC^0, U_E-uniformity coincides with U_D-uniformity, which is also frequently called DLOGTIME-uniformity. (Again, we refer the reader to [29] for more details on uniformity.)

Following the standard convention, we also use these same names to refer to the associated classes of *functions* computed by the corresponding classes of circuits. For instance, a function $f : \{0,1\}^* \to \{0,1\}^*$ is said to be in NC^1 if there is U_E-uniform family of circuits $\{C_n\}$ of bounded fan-in AND, OR and NOT gates, having size $n^{O(1)}$ and depth $O(\log n)$, where C_n has several output gates, and on input x of length n, C_n outputs an encoding of $f(x)$. (We say that an "encoding" of the output is produced, to allow the possibility that there are strings x and y of length n, such that $f(x)$ and $f(y)$ have different lengths.) It is easy to observe that, if the length of $f(x)$ is polynomial in $|x|$, then f is in NC^1 if and only if the language $\{(x, i, b) :$ the i-th symbol of $f(x)$ is $b\}$ is in NC^1. Similar observations hold for other classes.

A structure $(\mathcal{A}, +, \times)$ is a semiring if $(\mathcal{A}, +)$ is a commutative monoid with an additive identity element 0, and $(\mathcal{A}, \times)$ is a (not necessarily commutative) monoid with a multiplicative identity element 1, such that, for all a, b, c, we have $a \times (b + c) = (a \times b) + (a \times c)$, $(b + c) \times a = (ba \times ca)$, and $0 \times a = a \times 0 = 0$.

▶ **Definition 1.** An *arithmetic circuit* over a semiring $(R, +, \times)$ is a directed acyclic graph. Each vertex of the graph is called a "gate"; each gate is labeled with a "type" from the set $\{+, \times$, input, constant$\}$, where each input gate is labeled by one of the inputs $x_1, \ldots, x_n$, and each constant gate is labeled with an element of R. (Input and constant gates have indegree zero.) The is a unique sink called the "output gate". The *size* of a circuit is the number of gates, and the *depth* of the circuit is the length of the longest path in the circuit. We shall also need to refer to the *width* of a circuit, and here we use the notion of circuit width that was provided by Pippenger [27]: We will consider *layered* circuits, which means that the set of gates is partitioned into layers, where wires connect only gates in adjacent layers. The width of a circuit is the largest number of gates that occurs in any layer.

If an arithmetic circuit C_n over $(R, +, \times)$ has n input gates, then C_n computes a function $f : R^n \to R$ in the obvious way.

▶ **Definition 2.** A *straight-line program* over a semiring R with registers $\{r_1, \ldots, r_k\}$ consists of a sequence of statements of the form $r_i \leftarrow r_j \odot r_k$ where $\odot$ is one of the semiring operations, and each r_ℓ is a register, a value from R, or from the set of input variables. Straight-line programs have been studied at least as far back as [23], and they are frequently used as an alternative formulation of arithmetic circuits. Note that each line in a straight-line program can be viewed as a gate in an arithmetic circuit.

▶ **Definition 3.** The *width-k circuit value problem* over a semiring R is that of determining, given a width-k arithmetic circuit C over R (where C has no input gates, and hence all gates with indegree zero are labeled by a constant in R), and given a pair (i, b) whether the i-th bit of the binary representation of the output of C is b.

- $\#\mathsf{NC}^1{}_S$ is the class of functions $f : \bigcup_n R^n \to R$ for which there is a U_E-uniform family of arithmetic circuits $\{C_n\}$ of logarithmic depth, such that C_n computes f on R^n.
- By convention, when there is no subscript, $\#\mathsf{NC}^1$ denotes $\#\mathsf{NC}^1{}_{(\mathbb{N},+,\times)}$, with the additional restriction that the functions in $\#\mathsf{NC}^1$ are considered to have domain $\bigcup_n \{0,1\}^n$. That is, we restrict the inputs to the Boolean domain. (Boolean negation is also allowed at the input gates.)
- GapNC^1 is defined as $\#\mathsf{NC}^1 - \#\mathsf{NC}^1$; that is: the class of all functions that can be expressed as the difference of two $\#\mathsf{NC}^1$ functions. It is the same as $\#\mathsf{NC}^1{}_{\mathbb{Z}}$ restricted to the Boolean domain. See [29, 2] for more on $\#\mathsf{NC}^1$ and GapNC^1.

The following inclusions are known:

$$\mathsf{NC}^0 \subseteq \mathsf{AC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \#\mathsf{NC}^1 \subseteq \mathsf{GapNC}^1 \subseteq \mathsf{L} \subseteq \mathsf{AC}^1 \subseteq \mathsf{P}.$$

All inclusions are straightforward, except for $\mathsf{GapNC}^1 \subseteq \mathsf{L}$ [19].

2.1 Cost-register automata

A *cost-register automaton* (CRA) is a deterministic finite automaton (with a read-once input tape) augmented with a fixed finite set of *registers* that store elements of some algebraic domain $\mathcal{A}$. At each step in its computation, the machine

- consumes the next input symbol (call it a),
- moves to a new state (based on a and the current state (call it q)),
- based on q and a, updates each register r_i using updates of the form $r_i \leftarrow f(r_1, r_2, \ldots, r_k)$, where f is an expression built using the registers $r_1, \ldots, r_k$ using the operations of the algebra $\mathcal{A}$.

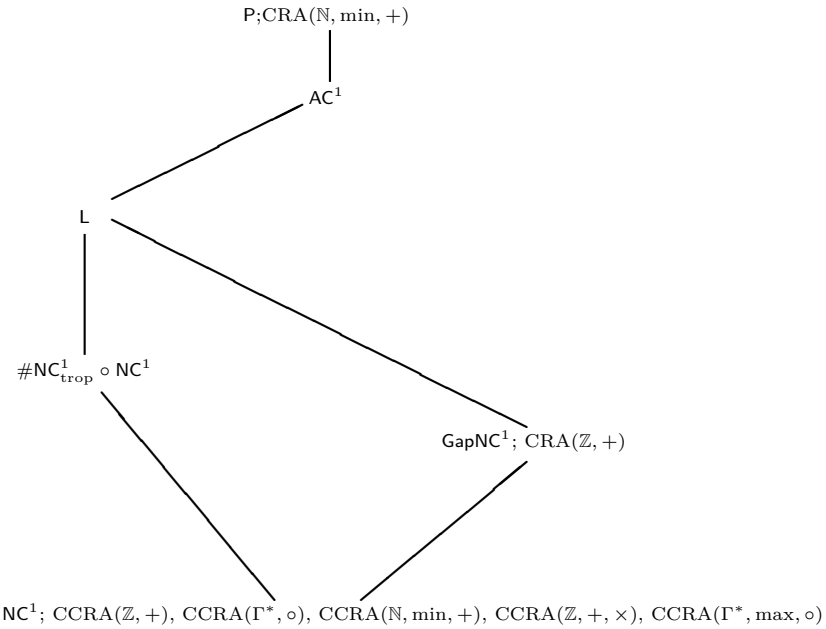


Figure 2 Update of the preceding figure, showing the improved state of our knowledge regarding CRA($\mathbb{N}, \min, +$) and the copyless CRA classes CCRA($\mathbb{N}, \min, +$), CCRA($\mathbb{Z}, +, \times$), and CCRA($\Gamma^*, \max, \circ$). All bounds listed are now tight.

There is also an "output" function μ defined on the set of states; μ is a partial function – it is possible for $\mu(q)$ to be undefined. Otherwise, if $\mu(q)$ is defined, then $\mu(q)$ is some expression of the form $f(r_1, r_2, \ldots, r_k)$, and the output of the CRA on input x is $\mu(q)$ if the computation ends with the machine in state q.

More formally, here is the definition as presented by Alur *et al.* [5].

A cost-register automaton M is a tuple $(\Sigma, Q, q_0, X, \delta, \rho, \mu)$, where

- Σ is a finite input alphabet.
- Q is a finite set of states.
- $q_0 \in Q$ is the initial state.
- X is a finite set of *registers*.
- $\delta : Q \times \Sigma \to Q$ is the state-transition function.
- $\rho : Q \times \Sigma \times X \to E$ is the register update function (where E is a set of algebraic expressions over the domain $\mathcal{A}$ and variable names for the registers in X).
- $\mu : Q \to E$ is a (partial) final cost function.

A *configuration* of a CRA is a pair (q, ν), where ν maps each element of X to an algebraic expression over $\mathcal{A}$. The *initial configuration* is (q_0, ν_0), where ν_0 assigns the value 0 to each register (or some other "default" element of the underlying algebra). Given a string $w = a_1 \ldots a_n$, the *run* of M on w is the sequence of configurations $(q_0, \nu_0), \ldots (q_n, \nu_n)$ such that, for each $i \in \{1, \ldots, n\}$ $\delta(q_{i-1}, a_i) = q_i$ and, for each $x \in X$, $\nu_i(x)$ is the result of composing the expression $\rho(q_{i-1}, a_i, x)$ to the expressions in ν_{i-1} (by substituting in the expression $\nu_{i-1}(y)$ for each occurrence of the variable $y \in X$ in $\rho(q_{i-1}, a_i, x)$). The output of M on w is undefined if $\mu(q_n)$ is undefined. Otherwise, it is the result of evaluating the expression $\mu(q_n)$ (by substituting in the expression $\nu_n(y)$ for each occurrence of the variable $y \in X$ in $\mu(q_n)$).

It is frequently useful to restrict the algebraic expressions that are allowed to appear in the transition function $\rho : Q \times \Sigma \times X \to E$. One restriction that is important in previous work [5] is the "copyless" restriction.

A CRA is *copyless* if, for every register $r \in X$, for each $q \in Q$ and each $a \in \Sigma$, the variable "r" appears at most once in the multiset $\{\rho(q, a, s) : s \in X\}$. In other words, for a given transition, no register can be used more than once in computing the new values for the registers. Following [6], we refer to copyless CRAs as CCRAs. Over many algebras, unless the copyless restriction is imposed, CRAs compute functions that can not be computed in polynomial time. For instance, CRAs that can concatenate string-valued registers and CRAs that can multiply integer-valued registers can perform "repeated squaring" and thereby obtain results that require exponentially-many symbols to write down.

3 CRAs over the Tropical Semiring

CRAs *without* the copyless restriction over the tropical semiring still yield only functions that are computable in polynomial time. The "repeated squaring" operation, when the "multiplicative" operation is $+$, yields only numbers whose binary representation remains linear in the length of the input. In this section, we show that some CRA functions over the tropical semiring are hard for P.

The name "tropical semiring" is used to refer to several related algebras. Most often it refers to $(\mathbb{R} \cup \{\infty\}, \min, +)$ (that is, the "additive" operation is min, and the "multiplicative" operation is $+$. However, frequently $(\mathbb{R} \cup \{-\infty\}, \max, +)$ is used instead. In discrete applications, $\mathbb{R}$ is frequently replaced with $\mathbb{Q}$, $\mathbb{Z}$, or even $\mathbb{N}$. For more details, we refer the reader to [26]. We will not need to make any use of ∞ or $-\infty$ in our hardness argument, and we will prove P-hardness over $\mathbb{N}$, which thus implies hardness for the other settings as well. Our arguments will be slightly different for both the max and the min versions, and thus we will consider both.

The standard reference for P-completeness, [18], credits Venkateswaran with the proof that the Min-plus Circuit Value Problem is P-complete. This shows that evaluating straight-line programs over $(\mathbb{N}, \min, +)$ is a P-complete problem, as long as they are allowed to have an unbounded number of registers.

Our focus will be more on straight-line programs with a *bounded* number of registers. Ben-Or and Cleve [11] showed that straight-line programs with $O(1)$ registers can simulate arithmetic formulae, and Koucky [21] has shown that these models are in fact *equivalent*, if the straight-line programs are restricted to compute only formal polynomials whose degree is bounded by a polynomial in the number of variables. It is observed in [1] that arithmetic formulae (that is, straight-line programs with $O(1)$ registers *and a polynomial degree restriction*) over the tropical semiring can be evaluated in logspace. Our P-completeness result demonstrates that, in the *absence* of any degree restriction, restricting straight-line programs over the tropical semiring to have only $O(1)$ registers yields a model that is as powerful as having an unlimited number of registers.

▶ **Theorem 4.** *There is a function f computable by a CRA operating over the tropical semiring (either $(\mathbb{N} \cup \{\infty\}, \min, +)$ or $(\mathbb{N} \cup \{-\infty\}, \max, +)$) such that computing f is hard for* P *under* AC^0*-Turing reductions.*

Proof. We will present a reduction from the P-complete problem Iterated Mod (problem A.8.5 in [18]), which was shown to be P-complete under logspace reductions by Karloff and Ruzzo [20]. The proof in [20] actually shows that the problem is complete under many-one reductions computable by dlogtime-uniform AC^0 circuits. (Incidentally, the proof sketch in [18] has a minor error, in that some indices are listed in the wrong order. The reader is advised to consult the original [20] proof.)

The input to the Iterated Mod problem is a list of natural numbers $v, m_1, m_2, \ldots, m_n$, and the question is to determine if $((\cdots((v \bmod m_1) \bmod m_2)\cdots) \bmod m_n) = 0$.

Let c be chosen so that 2^c is greater than any of the numbers $v, m_1, m_2, \ldots, m_n$. Then the naïve division algorithm that one would use to compute $v \bmod m$ can be seen to involve computing the following sequence:

- $v_0 = v$
- $v_i = v_{i-1} - max(0, v_{i-1} - m \cdot 2^{c-i})$

By induction, one can see that each $v_i \equiv v \pmod m$ and $v_i < m2^{c-i}$, and hence v_c is the remainder when one divides v by m.

Thus $v \bmod m$ can be seen to be computed by the following straight-line program over $\mathbb{Z}$ with operations $+, -, \max$:

```
1:  for i ≤ c do
2:     shift_of_m ← m
3:     for k ≤ c − i do
4:        shift_of_m ← shift_of_m + shift_of_m
5:     end for
6:     {At end of this loop, shift_of_m = m2^{c−i}}
7:     temp ← v−shift_of_m
8:     temp ← max(0, temp)
9:     v ← v − temp
10: end for
```

Of course, by definition, straight-line programs contain no loop statements, but the algorithm can be computed by a program described by an AC^0-computable sequence of symbols from the 5-letter alphabet $\{$*shift_of_m* $\leftarrow m$, *temp* $\leftarrow v-$ *shift_of_m*, $v \leftarrow v-$*temp*, *shift_of_m* $\leftarrow$ *shift_of_m* + *shift_of_m*, *temp* $\leftarrow \max(0,$*temp*$)\}$.

A number v, presented as a sequence of b binary digits v_i, can be loaded into a register r by initially setting r to 0, and then executing b instructions of the form $r \leftarrow r + r + v_i$. Thus, the naïve polynomial-time algorithm for computing Iterated Mod can be implemented via a polynomial-size straight-line program over $\mathbb{Z}$ with operations $+, -, \max$, by first inputting the numbers v and m_1, executing the algorithm above to compute $v \bmod m_1$, then inputting m_2, repeating the procedure to compute $((v \bmod m_1) \bmod m_2)$, etc.

We observe next that $\max(a, b) = (-1) \cdot \min(-a, -b)$. Thus there is a polynomial-size straight-line program over $\mathbb{Z}$ with operations $+, -, \min$ that outputs 0 if and only if $(v, m_1, \ldots, m_n)$ is a positive instance of Iterated Mod, where the process to input a number in binary has each instruction $r \leftarrow r + r + v_i$ replaced by $r \leftarrow r + r - v_i$. Similarly, in the code to compute $v \bmod m$, each occurrence of the instruction *temp* $\leftarrow \max(0,$*temp*$)$ is replaced by *temp* $\leftarrow \min(0,$*temp*$)$, and each occurrence of $r \leftarrow v - s$ for $\{r, s\} \subseteq \{v,$ *temp*, *shift_of_m*$\}$ is replaced by $r \leftarrow s - v$.

The next observation is that, given any straight-line program Q over $\mathbb{Z}$, it is easy to build a straight-line program Q' over $\mathbb{Z}$, such that each register of Q' always holds a nonnegative integer, and such that the value of each register r of Q at the end of the computation is equal to the value of the difference $r - r_0$ of Q' at the end, where r_0 is a new special register of Q'. We accomplish this by initially setting r_0 to 2^c (using repeated addition), where 2^c is larger than any value that is stored by any register of Q during its computation. (This is possible by taking c to be larger than the length of Q.) Then for every other register $r \neq r_0$, perform the operation $r \leftarrow r_0$. Now we will maintain the invariant that the value of register r of Q is obtained by subtracting r_0 from the value of register r of Q'. This is accomplished

as follows: Replace any assignment $r \leftarrow b$ where b is a constant, with $r \leftarrow r_0 + b$. Replace each operation $r \leftarrow s - u$ by $r \leftarrow s - u + r_0$, and replace each operation $r \leftarrow s + u$ by the operations: $r \leftarrow s + u; r' \leftarrow s + r_0$ (for every $r' \neq r$); $r_0 \leftarrow r_0 + r_0$.

The final step is to replace every straight-line program Q over $(\mathbb{Z}, \max, +, -)$ or ($\mathbb{Z}$,min,+, -) where every register holds only nonnegative values by a new program Q' over $(\mathbb{N}, \max, +)$ or $(\mathbb{N}, \min, +)$, where the value of every register r of Q at the end is equal to the value of the difference $r - r_{-1}$ of registers of Q', where r_{-1} is a new register of Q'. Initially, $r_{-1} \leftarrow 0$. Operations that involve min or max need no modification. If Q has the operation $r \leftarrow s + u$, then Q' has the operations $r \leftarrow s + u; r' \leftarrow s + r_{-1}$ (for every $r' \neq r$); $r_{-1} \leftarrow r_{-1} + r_{-1}$. (This is exactly the same replacement as was used in the preceding paragraph.) Finally, if Q has the operation $r \leftarrow s - u$, then Q' has the operations: $r \leftarrow s + r_{-1}; r_{-1} \leftarrow r_{-1} + u; r' \leftarrow r' + u$ for every $r' \notin \{r, u\}$ (including $r' = r_{-1}$); and then $u \leftarrow u + u$. A straightforward induction shows that the invariant is maintained, that each register r of Q has the value $r - r_{-1}$ of Q'.

Thus, given an instance y of Iterated Mod, an AC^0 reduction can produce a straight-line program Q over $(\mathbb{N}, \min, +)$ or $(\mathbb{N}, \max, +)$, such that $y \in$ Iterated Mod iff the output register of Q has a value equal to the value of r_0.

Note that there is a CRA that takes as input strings over an alphabet whose symbols encode straight-line program instructions with $O(1)$ registers, and simulates the operation of the straight-line program. The function f that is computed by this CRA is the function whose existence is asserted in the statement of the theorem. ◀

▶ **Corollary 5.** *Let R be the semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$ or $(\mathbb{N} \cup \{-\infty\}, \max, +)$. There is a constant c such that for every $k \geq c$, the width-k circuit value problem over R is* P*-complete under* AC^0*-Turing reductions.*

Proof. The P upper bounds are clear since each semiring operation is polynomial-time computable. Hardness follows by appealing to the straight-line programs with a bounded number of registers that are constructed in the proof of Theorem 4. A further AC^0-Turing reduction can transform a straight-line program that uses k registers into an arithmetic circuit of width $O(k)$. (Each layer in the arithmetic circuit contains a gate for each register, as well as gates for each constant that is used in the next time step. If a register r is not changed at time t, then the gate for register r in layer t is simply set to 0 + the value of register r at in layer $t - 1$.) ◀

Completeness under AC^0-many-one reductions (or even logspace many-one reductions) is still open.

4 CCRAs over Commutative Semirings

In this section, we study two classes of functions defined by CCRAs operating over commutative algebras with two operations satisfying the semiring axioms:

- CRAs operating over the commutative ring $(\mathbb{Z}, +, \times)$
- CRAs operating over the tropical semiring, that is, over the commutative semiring $(\mathbb{Z} \cup \{\infty\}, \min, +)$.

▶ **Theorem 6.** *Let $(\mathcal{A}, +, \times)$ be a commutative semiring such that the functions*

$$(x_1, x_2, \ldots, x_n) \mapsto \sum_i x_i \text{ and } (x_1, x_2, \ldots, x_n) \mapsto \prod_i x_i$$

can be computed in NC^1*. Then CCRA($\mathcal{A}$) $\subseteq \mathsf{NC}^1$.*

We remark that both the tropical semiring and the integers satisfy this hypothesis. We refer the reader to [29, 19] for more details about the inclusions:

- unbounded-fan-in min $\in \mathsf{AC}^0$.
- unbounded-fan-in $+ \in \mathsf{TC}^0$.
- unbounded-fan-in $\times \in \mathsf{TC}^0$.

Proof. Let $M = (Q, \Sigma, \delta, q_0, X, \rho, \mu)$ be a copyless CRA operating over $\mathcal{A}$. Let M have k registers $r_1, \ldots, r_k$.

As in the proof of [1, Theorem 1], it is straightforward to see that the following functions are computable in NC^1:

- $(x, i) \mapsto q$, such that M is in state q after reading the prefix of x of length i. Note that this also allows us to determine the state q that M is in while scanning the final symbol of x, and thus we can determine whether the output $\mu(q)$ is defined.
- $(x, i) \mapsto G_i$, where G_i is a labeled directed bipartite graph on $[2k] \times [k]$, with the property that there is an edge from j on the left-hand side to ℓ on the right hand side, if the register update operation that takes place when M consumes the i-th input symbol includes the update $r_\ell \leftarrow \alpha \otimes \beta$ where $r_j \in \{\alpha, \beta\}$ and $\otimes \in \{+, \times\}$. In addition, vertex ℓ is labeled with the operation $\otimes$. If one of $\{\alpha, \beta\}$ is a constant c (rather than being a register), then label vertex $k + \ell$ in the left-hand column with the constant c, and add an edge from vertex $k + \ell$ in the left-hand column to ℓ in the right-hand column. (To see that this is computable in NC^1, note that by the previous item, in NC^1 we can determine the state q that M is in as it consumes the i-th input symbol. Thus G_i is merely a graphical representation of the register update function corresponding to state q.) Note that the outdegree of each vertex in G_i is at most one, because M is copyless. (The indegree is at most two.) To simplify the subsequent discussion, define G_{n+1} to be the graph resulting from the "register update function" $r_\ell \leftarrow \mu(q)$ for $1 \le \ell \le k$, where q is the state that M is in after scanning the final symbol x_n.

Now consider the graph G that is obtained by concatenating the graphs G_i (by identifying the left-hand side of G_{i+1} with the first k vertices of the right-hand side of G_i for each i). This graph shows how the registers at time $i + 1$ depend on the registers at time i. G is a constant-width graph, and it is known that reachability in constant-width graphs is computable in NC^1 [8, 9].

The proof of the theorem proceeds by induction on the number of registers $k = |X|$. When $k = 1$, note that the graph G consists of a path of length $n + 1$, where each vertex v_i on the path is connected to two vertices on the preceding level, one of which is a leaf. (Here, we are ignoring degenerate cases, where the path back from the output node does not extend all the way back to the start, but instead stops at some vertex v_i where the corresponding register assignment function sets the register to a constant. An NC^1 computation can find where the path actually does start.) That is, when $k = 1$, the graph G has width two. We will thus really do our induction on the width of the graph G, starting with width two.

In $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$, we can partition the index set $I = \{0, \ldots, n + 1\}$ into consecutive subsequences $S_1, P_1, S_2, P_2, \ldots, S_m, P_m$, where $i \in S_j$ implies that vertex v_i on the path is labeled with $+$, and $i \in P_j$ implies that vertex v_i on the path is labeled with $\times$. (Assume for convenience that the first operation on the path is $+$ and the last one is $\times$; otherwise add dummy initial and final operations that add 0 and multiply by 1, respectively.) That is, $i \in S_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} + c_{i-1}$, and $i \in P_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} \times c_{i-1}$ for some sequence of constants $c_0, \ldots, c_n$.

In NC^1 we can compute the values $s_j = \sum_{i \in S_j} c_{i-1}$ and $p_j = \prod_{i \in P_j} c_{i-1}$. Thus the output computed by M on x is

$$(\ldots(((s_1 \times p_1) + s_2) \times p_2) \ldots \times p_m) = \sum_j s_j \prod_{\ell \geq j} p_\ell.$$

This expression can also be evaluated in NC^1. This completes the proof of the basis case, when $k = 1$.

Now assume that functions expressible in this way when the width of the graph G is at most k can be evaluated in NC^1. Consider the case when G has width $k+1$, and assume that vertex 1 in the final level is the vertex that evaluates to the value of the function. In NC^1 we can identify a path of longest length leading to the output. Let this path start in level i_0. Since there is no path from a vertex in any level $i < i_0$ to the output, we can ignore everything before level i_0 and just deal with the part of G starting at level i_0. Thus, for simplicity, assume that $i_0 = 0$. Let the vertices appearing on this path be $v_1, v_2, \ldots, v_{n+1}$, where each vertex v_i is labeled with the operation $v_i \leftarrow v_{i-1} \otimes_i w_i$ for some operation $\otimes_i$ and some vertex w_i. Let H_i be the subgraph consisting of all vertices that have a path to vertex w_i. Since the outdegree of each vertex in G is one, and since no w_i appears on the path, it follows that each H_i has width at most k, and thus the value computed by w_i (which we will also denote by w_i) can be computed in NC^1. (This is the only place where we use the restriction that M is a *copyless* CRA.)

Now, as before partition this path into subsequences $S_1, P_1, S_2, P_2, \ldots, S_m, P_m$, where $i \in S_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} + w_{i-1}$, and $i \in P_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} \times w_{i-1}$ for some NC^1-computable sequence of values $w_0, \ldots, w_n$.

Thus, as above, in NC^1 we can compute the values $s_j = \sum_{i \in S_j} w_{i-1}$ and $p_j = \prod_{i \in P_j} w_{i-1}$. Thus the output computed by M on x is

$$(\ldots(((s_1 \times p_1) + s_2) \times p_2) \ldots \times p_m) = \sum_j s_j \prod_{\ell \geq j} p_\ell.$$

This expression can also be evaluated in NC^1. ◀

5 CCRAs over Noncommutative Semirings

In this section, we show that the techniques of the preceding section can easily be adapted to work for noncommutative semirings.

The canonical example of such a semiring is $(\Gamma^* \cup \{\bot\}, \max, \circ)$. Here, the max operation takes two strings x, y in Γ^* as input, and produces as output the lexicographically-larger of the two. (Lexicographic order on Γ^* is defined as usual, where $x < y$ if $|x| < |y|$ or ($|x| = |y|$ and x precedes y, viewed as the representation of a number in $|\Gamma|$-ary notation). $\bot$ is the additive identity element. (One obtains a similar example of a noncommutative semiring, by using min in place of max.)

It is useful to describe how elements of Γ^* will be represented in an NC^1 circuit, in a way that allows efficient computation. For an input length n, let $m = n^{O(1)}$ be the maximum number of symbols in any string that will need to be manipulated while processing inputs of length n. Then a string y of length j will be represented as a sequence of $\log m + m \log |\Gamma|$ bits, where the first $\log m$ bits store the number j, followed by m blocks of length $\log |\Gamma|$, where the first j blocks store the symbols of y. Given a sequence of $l_1, r_1, l_2, r_2, \ldots, l_s, r_s$ represented in this way, we need to can compute the representation of the string $l_s l_{s-1} \ldots l_2 l_1 r_1 r_2 \ldots r_{s-1} r_s$.

It is easy to verify that this computation is in TC^0, since the i-th symbol of the concatenated string is equal to the j-th symbol of the ℓ-th string in this list, where j and ℓ are easy to compute by performing iterated addition on the lengths of the various strings, and comparing the result with i. In the max, $\circ$ semiring, where concatenation is the "multiplicative" operation, this corresponds to iterated product, and it is computable in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$.

▶ **Theorem 7.** *Let $(\mathcal{A}, +, \times)$ be a (possibly noncommutative) semiring such that the functions $(x_1, x_2, \ldots, x_n) \mapsto \sum_i x_i$ and $(x_1, x_2, \ldots, x_n) \mapsto \prod_i x_i$ can be computed in NC^1. Then CCRA($\mathcal{A}$) $\subseteq \mathsf{NC}^1$.*

Proof. The proof is a slight modification of the proof in the commutative case.

Given a CCRA M, we build the same graph G. Again, the proof proceeds by induction on the width of G (related to the number of registers in M).

Let us consider the basis case, where G has width two.

In $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$, we can partition the index set $I = \{0, \ldots, n+1\}$ into consecutive subsequences $S_1, P_1, S_2, P_2, \ldots, S_m, P_m$, where $i \in S_j$ implies that vertex v_i on the path is labeled with $+$, and $i \in P_j$ implies that vertex v_i on the path is labeled with $\times$. (Assume for convenience that the first operation on the path is $+$ and the last one is $\times$; otherwise add dummy initial and final operations that add 0 and multiply by 1, respectively.) That is, $i \in S_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} + c_{i-1}$, and $i \in P_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} \times c_{i-1}$ or $v_i \leftarrow c_{i-1} \times v_{i-1}$ for some sequence of constants $c_0, \ldots, c_n$.

In NC^1 we can compute the value $s_j = \sum_{i \in S_j} c_{i-1}$. The product segments P_j require just a bit more work. Let $l_{j,1}, l_{j,2}, \ldots, l_{j,m_{j_l}}$ be the list of indices, such that $l_{j,s}$ is the s-th element of $\{i \in P_j : \text{the multiplication operation at } v_i \text{ is of the form } v_i \leftarrow c_{i-1} \times v_{i-1}\}$, and similarly let $r_{j,1}, r_{j,2}, \ldots, r_{j,m_{j_r}}$ be the list of indices, such that $r_{j,s}$ is the s-th element of $\{i \in P_j : \text{the multiplication operation at } v_i \text{ is of the form } v_i \leftarrow v_{i-1} \times c_{i-1}\}$.

Let

$$l_j = c_{l_{j,m_{j_l}}-1} \times c_{l_{j,m_{j_l}-1}-1} \times \ldots c_{l_{j,2}-1} \times c_{l_{j,1}-1}$$

and let

$$r_j = c_{r_{j,1}-1} \times c_{r_{j,2}-1} \times c_{r_{j,m_{j_r}-1}-1} \times c_{r_{j,m_{j_r}}-1}.$$

Then if the value of the path when it enters segment P_j is y, it follows that the value computed when the path leaves segment P_j is $l_j y r_j$. Note that this value can be computed in NC^1.

Thus the output computed by M on x is

$$l_1 \times ((l_2 \times (\ldots (l_2 \times ((l_1 \times s_1 \times r_1) + s_2) \times r_2) \ldots) \times r_2) + s_1) \times r_1$$

which is equal to

$$\sum_j (\prod_{\ell \geq j} l_j) s_j (\prod_{\ell \geq j} r_\ell).$$

This expression can be evaluated in NC^1. This completes the proof of the basis case, when G has width two.

The proof for the inductive step is similar to the commutative case, combined with the algorithm for the basis case. ◀

6 Conclusion

We have obtained a polynomial time lower bound, conditional on $\mathsf{NC} \neq \mathsf{P}$, for some functions computed by CRAs over $(\mathbb{N}, \min, +)$ and other tropical semirings. This was done by proving that a straight-line program over such semirings using $O(1)$ registers can solve a P-complete problem. It followed that for some small k, the "width-k circuit value problem" over $(\mathbb{N}, \min, +)$ is P-complete. We have also shown that any function computed by a copyless CRA over such semirings belongs to (functional) NC^1.

An open question of interest would be to characterize the semirings $(R, +, \times)$ over which the width-k circuit value problem is P-complete. Given the P-completeness of the circuit value problem over the group A_5 [10], one possible approach would be to try to map R onto A_5 in such a way that iterating the evaluation of a fixed semiring expression over R would allow retrieving the result of a linear number of compositions of permutations from A_5.

A future direction in the study of copyless CRAs might be to refine our NC^1 analysis by restricting the algebraic properties of the underlying finite automaton, along the lines described in the context of ordinary finite automata (see Straubing [28] for a broader perspective). The way to proceed is not immediately clear however since merely restricting the finite automaton (say to an aperiodic automaton) would not reduce the strength of the model unless the interplay between the registers is also restricted.

Acknowledgments. Some of this research was performed at the 29th McGill Invitational Workshop on Computational Complexity, held at the Bellairs Research Institute of McGill University, in February, 2017.

References

1 E. Allender and I. Mertz. Complexity of regular functions. *Journal of Computer and System Sciences*, 2017. To appear; LATA 2015 Special Issue. Earlier version appeared in Proc. 9th International Conference on Language and Automata Theory and Applications (LATA'15), Springer Lecture Notes in Computer Science vol. 8977, pp. 449-460. doi:10.1016/j.jcss.2016.10.005.

2 Eric Allender. Arithmetic circuits and counting complexity classes. In J. Krajíček, editor, *Complexity of Computations and Proofs*, volume 13 of *Quaderni di Matematica*, pages 33–72. Seconda Università di Napoli, 2004.

3 Rajeev Alur and Pavol Cerný. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, pages 599–610, 2011. doi:10.1145/1926385.1926454.

4 Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions, cost register automata, and generalized min-cost problems. *CoRR*, abs/1111.0670, 2011. URL: https://arxiv.org/abs/1111.0670.

5 Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 13–22, 2013. See also the expanded version, [4]. doi:10.1109/LICS.2013.65.

6 Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science, (CSL-LICS)*, page 9. ACM, 2014. doi:10.1145/2603088.2603151.

7 Rajeev Alur and Mukund Raghothaman. Decision problems for additive regular functions. In *Proc. 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, number 7966 in Lecture Notes in Computer Science, pages 37–48. Springer, 2013. doi:10.1007/978-3-642-39212-2_7.

8 D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989. doi:10.1016/0022-0000(89)90037-8.

9 D. A. M. Barrington, C.-J. Lu, P. B. Miltersen, and S. Skyum. Searching constant width mazes captures the AC^0 hierarchy. In *Proc. 15th International Symposium on Theoretical Aspects of Computer Science (STACS)*, number 1373 in Lecture Notes in Computer Science, pages 73–83. Springer, 1998. doi:10.1007/BFb0028542.

10 M. Beaudry, P. McKenzie, P. Péladeau, and D. Thérien. Finite monoids: from word to circuit evaluation. *SIAM Journal on Computing*, 26:138–152, 1997. doi:10.1137/S0097539793249530.

11 Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21(1):54–58, 1992. doi:10.1137/0221006.

12 Michaël Cadilhac, Andreas Krebs, and Nutan Limaye. Value automata with filters. *CoRR*, abs/1510.02393, 2015. URL: http://arxiv.org/abs/1510.02393.

13 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Proceedings, Part II*, pages 139–150, 2009. doi:10.1007/978-3-642-02930-1_12.

14 Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013. doi:10.2168/LMCS-9(3:3)2013.

15 Thomas Colcombet, Denis Kuperberg, Amaldev Manuel, and Szymon Toruńczyk. Cost functions definable by min/max automata. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 29:1–29:13, 2016. doi:10.4230/LIPIcs.STACS.2016.29.

16 Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot. A generalised twinning property for minimisation of cost register automata. 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 857–866, 2016. doi:10.1145/2933575.2934549.

17 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer-Verlag New York Inc., 2009. doi:10.1007/978-3-642-01492-5.

18 Raymond Greenlaw, H James Hoover, and Walter L Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, 1995.

19 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002. doi:10.1016/S0022-0000(02)00025-9.

20 Howard J Karloff and Walter L Ruzzo. The iterated mod problem. *Information and Computation*, 80(3):193–204, 1989. doi:10.1016/0890-5401(89)90008-4.

21 Michal Koucký. Unpublished manuscript, 2003.

22 Andreas Krebs, Nutan Limaye, and Michael Ludwig. Cost register automata for nested words. In *Proc. 22nd International Computing and Combinatorics Conference - (COCOON)*, number 9797 in LNCS, pages 587–598. Springer, 2016. doi:10.1007/978-3-319-42634-1_47.

23 Nancy A Lynch. Straight-line program length as a parameter for complexity analysis. *Journal of Computer and System Sciences*, 21(3):251–280, 1980. doi:10.1016/0022-0000(80)90024-0.

24 Filip Mazowiecki and Cristian Riveros. Maximal partition logic: Towards a logical characterization of copyless cost register automata. In *24th EACSL Annual Conference on*

Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany, pages 144–159, 2015. doi:10.4230/LIPIcs.CSL.2015.144.

25 Filip Mazowiecki and Cristian Riveros. Copyless cost-register automata: Structure, expressiveness, and closure properties. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 53:1–53:13, 2016. doi:10.4230/LIPIcs.STACS.2016.53.

26 Jean-Eric Pin. *Tropical semirings*. Cambridge Univ. Press, Cambridge, 1998. doi:10.1017/CBO9780511662508.004.

27 Nicholas Pippenger. On simultaneous resource bounds. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 307–311, 1979. doi:10.1109/SFCS.1979.29.

28 H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, 1994. doi:10.1007/978-1-4612-0289-9.

29 H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York Inc., 1999. doi:10.1007/978-3-662-03927-4.

Kernelization of the Subset General Position Problem in Geometry

Jean-Daniel Boissonnat[1], Kunal Dutta[2], Arijit Ghosh[3], and Sudeshna Kolay[4]

1 **INRIA Sophia Antipolis - Méditerranée, France**
2 **INRIA Sophia Antipolis - Méditerranée, France**
3 **Indian Statistical Institute, Kolkata, India**
4 **Eindhoven University of Technology, Eindhoven, The Netherlands**

Abstract

In this paper, we consider variants of the Geometric Subset General Position problem. In defining this problem, a geometric subsystem is specified, like a subsystem of lines, hyperplanes or spheres. The input of the problem is a set of n points in $\mathbb{R}^d$ and a positive integer k. The objective is to find a subset of at least k input points such that this subset is in general position with respect to the specified subsystem. For example, a set of points is in general position with respect to a subsystem of hyperplanes in $\mathbb{R}^d$ if no $d+1$ points lie on the same hyperplane. In this paper, we study the Hyperplane Subset General Position problem under two parameterizations. When parameterized by k then we exhibit a polynomial kernelization for the problem. When parameterized by $h = n - k$, or the dual parameter, then we exhibit polynomial kernels which are also tight, under standard complexity theoretic assumptions. We can also exhibit similar kernelization results for d-Polynomial Subset General Position, where a vector space of polynomials of degree at most d are specified as the underlying subsystem such that the size of the basis for this vector space is b. The objective is to find a set of at least k input points, or in the dual delete at most $h = n-k$ points, such that no $b+1$ points lie on the same polynomial. Notice that this is a generalization of many well-studied geometric variants of the Set Cover problem, such as Circle Subset General Position. We also study general projective variants of these problems. These problems are also related to other geometric problems like Subset Delaunay Triangulation problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Incidence Geometry, Kernel Lower bounds, Hyperplanes, Bounded degree polynomials

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.25

1 Introduction

In the *geometric subset general position* problem, the input is a *family* of *algebraic objects*, e.g. lines, circles, hyperplanes, zero set of quadratic functions, and a point set P in $\mathbb{R}^d$. The objective is to extract a large subset S of P such that the subset S is in *general position* with respect to the geometric objects. The definition of general position is different for different families of geometric objects. For the case of hyperplanes in $\mathbb{R}^d$, a set S, assume $|S| > d$, will be in general position with respect to the family of hyperplanes in $\mathbb{R}^d$ if no more than d points of S lie on a hyperplane. For the case of spheres in $\mathbb{R}^d$, a set S with $|S| > d+1$, will be in general position with respect to the family of spheres in $\mathbb{R}^d$ if no more than $d+1$ points of S lie on a sphere. In this paper, we will assume that d is a constant.

In computational geometry it is generally assumed that the point set is in general position, such as no more than d points lie on a hyperplane in the case of convex hull computation or no more than $d+1$ points lie on a sphere for Delaunay triangulation computation (see [6]). Also, algebraic techniques like *simulation of simplicity* have been introduced to handle degenerate cases in practice [10].

The problem of determining whether a given point set in $\mathbb{R}^d$ is in general position with respect to family of spheres and family of hyperplanes has been extensively studied in computational geometry. Edelsbrunner, O'Rourke and Seidel [11] gave an $O(n^d)$ (and $O(n^{d+1})$) space and time complexity algorithm to determine if a point set is in general position with respect to hyperplanes (resp. spheres) in $\mathbb{R}^d$. Edelsbrunner and Guibas [8, 9] later improved the space bound to $O(n)$. Erickson and Seidel [12, 13] showed in the worst case $\Omega(n^d)$ (and $\Omega(n^{d+1})$) sided queries are required to determine whether a set of n points in $\mathbb{R}^d$ is in general position with respect to hyperplanes (resp. spheres). We have mentioned a small sample of the papers on this topic and for a more complete picture of this area and the more general problem of *arrangement of hyperplanes* please refer to the survey by Agarwal and Sharir [1].

More recently, the problem of finding a maximum-cardinality subset of points in general position has been studied in parameterized complexity, approximation algorithm, and combinatorial geometry [14, 18, 4]. Payne et al. [18] and Cardinal [4] gave a non-trivial lower bound on the size of a largest size subset in general position from a point set with bounded *coplanarity* in $\mathbb{R}^2$ and $\mathbb{R}^d$. A point set in $\mathbb{R}^2$ (and $\mathbb{R}^d$) has bounded coplanarity if the number of points from the set that lie on a given plane (or hyperplane) is bounded. Cao [3] and Froese et al. [14] studied the geometric subset general position problem in $\mathbb{R}^2$ with respect to lines in $\mathbb{R}^2$ through the lens of parameterized complexity. Cao [3] also gave an $\mathcal{O}(\sqrt{\mathsf{opt}})$-factor approximation algorithm for the general position subset selection problem with respect to lines in $\mathbb{R}^2$.

In this paper we generalize the results of [14] by studying the kernelization aspect of the following *primal* problem:

Hyperplane Subset General Position	**Parameter:** k
Input: An n point set P in $\mathbb{R}^d$, for a fixed constant d, and a positive integer k	
Question: Is there a subset $S \subseteq P$ of size at least k such that S is in general position with respect to hyperplanes in $\mathbb{R}^d$?	

We will also study a more general version of the above problem for *bounded degree polynomial families* (see Section 2 and 3):

d-Polynomial Sub. General Pos.	**Parameter:** k
Input: A set P of n points in $\mathbb{R}^d$, for a fixed constant d, a bounded degree polynomial family $\mathcal{F}$ in $\mathbb{R}^d$ and a positive integer k	
Question: Is there a subset $S \subseteq P$ of size at least k such that S is in general position with respect to $\mathcal{F}$ in $\mathbb{R}^d$?	

Therefore, the general position subset selection problems with respect to natural set families like the vector space of spheres, ellipses, etc. are special cases of the d-Polynomial Subset General Position problem. We also study the problems with respect to the dual parameter $h = n - k$. That is, the problem of Hyperplane Subset General Position or d-Polynomial Subset General Position still have the same input and aim for the same decision problem. However, the parameter for the problem becomes h.

Note that both these problems are NP-hard following from the results of [14] on Subset General Position in $\mathbb{R}^2$.

Our contribution

In this paper, we exhibit polynomial kernels for Hyperplane Subset General Position in $\mathbb{R}^d$. This is a generalization to higher dimensions of the results on kernelization obtained in [14], with more carefully designed reduction rules to take care of the higher dimension. We further generalize the result with the help of a variant of the Veronese mapping, to obtain polynomial kernels for d-Polynomial Subset General Position in $\mathbb{R}^t$, where the bounded degree polynomial family is a vector space of d-degree polynomials. Special cases of the d-Polynomial Subset General Position problem include variants where the polynomial family is that of spheres or quadratic surfaces. Also, Delaunay Subset Selection is a special case of this problem. We further study the general projective variants of these problems. These results are described in Section 3

We also give tight polynomial kernels for Hyperplane Subset General Position in $\mathbb{R}^d$ parameterized by h, the dual parameter, as described in Section 4. In Section 4, we obtain tight results for the number of elements in a polynomial kernel for Hyperplane Subset General Position in $\mathbb{R}^d$. These results are similar to those obtained in [15]. Finally, in Section 5, we are able to generalize this result for certain variants of d-Polynomial Subset General Position.

2 Preliminaries

Hypergraphs

A set of consecutive integers $\{1, 2, \ldots n\}$ will be written as $[n]$ in short. A hypergraph G is a set system where $V(G)$ denotes the universe and $E(G)$ denotes the family of sets. We refer to the objects in the universe $V(G)$ by either vertices or elements, and each subset of $E(G)$ as a *hyperedge*. For a hyperedge $e \in E(G)$ the set of vertices belonging to e is denoted as V_e. A *d-uniform hypergraph* is a hypergraph where each hyperedge has exactly d vertices. Similarly, a *d-hypergraph* is a hypergraph where each hyperedge has at most d vertices. An *independent set* in a hypergraph G is a subset $I \subseteq V(G)$ such that there is no $e \in E(G)$ where all vertices in e belong to I. The d-Hypergraph Independent Set problem takes as input a d-hypergraph and a positive integer k and determines whether the input hypergraph has an independent set of size at least k.

The d-Hitting Set problem takes as input a d-hypergraph G and a positive integer k and determines whether there is a set $S \subseteq V(G)$ of size at most k such that for each $e \in E(G)$, $V_e \cap S \neq \emptyset$. Such a set S is called a d-hitting set.

General position in Geometry

An i-flat in $\mathbb{R}^d$ is the affine hull of $i+1$ affinely independent points. The dimension of a (possibly infinite) set of points P, denoted as $dim(P)$, is the minimum i such that the entire set P is contained in an i-flat of $\mathbb{R}^d$ [16]. We use the term hyperplanes interchangeably for $(d-1)$-flats. A set P of points in $\mathbb{R}^d$ is said to be in general position with respect to hyperplanes, if for each i-flat, $i \leq d-1$, in $\mathbb{R}^d$ there are at most $i+1$ points from P lying on the i-flat.

As described earlier, the Geometric Subset General Position problem, defined on a subsystem of geometric objects, takes as input a set of points P and a positive integer k and determines whether there is a subset of at least k points that are in general position with respect to the specified subsystem of geometric objects. For example, Hyperplane Subset General Position in $\mathbb{R}^d$ takes in a set of points in $\mathbb{R}^d$ and a positive integer k

and determines whether there is a subset of at least k points that are in general position with respect to hyperplanes in $\mathbb{R}^d$.

Similarly, we can define the notion of general position with respect to multivariate polynomials. Given a set $\{X_1, X_2, \ldots, X_t\}$ of variables, a real multivariate polynomial on these variables is of the form $P(X_1, \ldots, X_t) = \sum_{i_1,i_2,\ldots,i_t} a_{i_1 i_2 \ldots i_t} \prod_{j\in[t]} X_j^{i_j}$ where $[t] = \{1, \ldots, t\}$ and $a_{i_1 i_2 \ldots i_t} \in \mathbb{R}$. The set of all real multivariate polynomials in the variables $\{X_1, \ldots, X_t\}$ will be denoted by $\mathbb{R}[X_1, X_2, \ldots, X_t]$. The degree of such a polynomial $P(X_1, \ldots, X_t)$ is defined as $\deg(P) := \max\{i_1 + i_2 + \ldots + i_t \mid a_{i_1 i_2 \ldots i_t} \neq 0\}$. A polynomial is said to be a degree d polynomial is its degree is d.

In this paper, we are interested in the set/subsets of polynomials whose degree is bounded by d, for some $d \in \mathbb{N}$. In this context we define $\text{Poly}_d[X_1, \ldots, X_t] := \{f(X_1, \ldots, X_t) \in \mathbb{R}[X_1, X_2, \ldots, X_t] \mid \deg(f) \leq d\}$. Observe that $\text{Poly}_d[X_1, \ldots, X_t]$ is a vector space over $\mathbb{R}$ with the *monomials* $\left\{X_1^{i_1} \ldots X_t^{i_t} \mid 0 \leq \sum_{j=1}^t i_j \leq d\right\}$ as the basis. Notice that $\left|\left\{X_1^{i_1} \ldots X_t^{i_t} \mid 0 \leq \sum_{j=1}^t i_j \leq d\right\}\right| = \binom{d+t}{d}$.

In d-Polynomial Subset General Position in $\mathbb{R}^t$, a subspace $\mathcal{F}$ of $\text{Poly}_d[X_1, \ldots, X_t]$ is given with a basis $\{f_1(X), f_2(X), \ldots, f_b(X), 1\}$, where $X = (X_1, \ldots, X_t)$ and $f_i(X) \in \text{Poly}_d[X_1, \ldots, X_t]$, and a set of n points from $\mathbb{R}^t$. The objective is to find a subset of points in general position with respect to the vector space of polynomials, i.e., no more than b points from the subset satisfy any equation of the form $f(X) := \sum_{i=1}^b \lambda_i f_i(X) + \lambda_{b+1} = 0$, where for each $j \in \{1, \ldots, b\}$, $\lambda_j \in \mathbb{R}$ and not all the λ_j's can be zero simultaneously. Here are some concrete examples of d-Polynomial Subset General Position.

▶ **Example 1.**

1. Hyperplanes in $\mathbb{R}^t$ are zero sets of linear combinations of polynomials $\{X_1, \ldots, X_t, 1\}$.
2. Union of spheres and hyperplanes in $\mathbb{R}^d$ are zero sets of linear combinations of polynomials $\left\{\sum_{i=1}^t X_i^2, X_1, \ldots, X_t, 1\right\}$.
3. Polynomial surfaces with degree bounded by d are zero sets of $\text{Poly}_d[X_1, \ldots, X_t]$.
4. Quadratic surfaces are zero set of polynomials in $\text{Poly}_2[X_1, \ldots, X_t]$.

Veronese mapping

In this paper, one of our strategies for generalizing our results is to convert d-Polynomial Subset General Position in $\mathbb{R}^t$ to Hyperplane Subset General Position in $\mathbb{R}^b$ by using a variant of Veronese mapping [17] from $\mathbb{R}^t \to \mathbb{R}^b$. The Veronese mapping of a vector space $\mathcal{F}$ of d-degree polynomials will be as the following: $\Phi_{\mathcal{F}} : \mathbb{R}^t \to \mathbb{R}^b$, where for a vector $X = (X_1, \ldots, X_t)$, $\Phi_{\mathcal{F}}(X) = (f_1(X), \ldots, f_b(X))$. Observe that if $p = (p_1, \ldots, p_t)$ satisfies the equation $f(X) := \sum_{i=1}^b \lambda_i f_i(X) + \lambda_{b+1} = 0$ then, for a vector of variables $Z = (Z_1, \ldots, Z_b)$, $\Phi_{\mathcal{F}}(p)$ will also satisfy the linear equation $\sum_{j=1}^b \lambda_j Z_j + \lambda_{b+1} = 0$. In other words, for any set of points P in $\mathbb{R}^t$ and the vector space $\mathcal{F}$, the incidences between P and $\mathcal{F}$ and incidences between $\Phi_{\mathcal{F}}(P)$ and $\text{Poly}_1[Z_1, \ldots, Z_b]$ (these are hyperplanes in $\mathbb{R}^b$) are preserved under the mapping $\Phi_{\mathcal{F}}$. Also, observe that there is a bijection between polynomials in $\mathcal{F}$ and hyperplanes in $\mathbb{R}^b$.

Parameterized Complexity

The instance of a parameterized problem/language is a pair containing the actual problem instance of size n and a positive integer called a parameter, usually represented as k. The problem is said to be in FPT if there exists an algorithm that solves the problem in $f(k)n^{\mathcal{O}(1)}$ time, where f is a computable function. The problem is said to *admit a $g(k)$-sized kernel*, if

there exists an polynomial time algorithm that converts the actual instance to a reduced instance of size $g(k)$, while preserving the answer. When g is a polynomial function, then the problem is said to admit a polynomial kernel. A *reduction rule* is a polynomial time procedure that changes a given instance I_1 of a problem Π to another instance I_2 of the same problem Π. We say that the reduction rule is *safe* when I_1 is a YES instance of Π if and only if I_2 is a YES instance. Readers are requested to refer [5] for more details on Parameterized Complexity.

Lower bounds in Parameterized Algorithms

There are several methods of showing lower bounds in parameterized complexity under standard assumptions in complexity theory. One such technique is *polynomial parameter transformation*. For two parameterized problems Π, Π', a polynomial time algorithm $\mathcal{A}$ is called a polynomial parameter transformation (or ppt) from Π to Π' if, given an instance (x, k) of Π, $\mathcal{A}$ outputs in polynomial time an instance (x', k') of Π' such that $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi'$ and $k' \leq k^{\mathcal{O}(1)}$. By a result of [2], if Π, Π' are two parameterized problems such that Π is NP-hard, $\Pi \in$ NP and there exists a polynomial parameter transformation from Π to Π', then, if Π does not admit a polynomial kernel neither does Π'.

We also require a lower bound technique given in [7]. This technique links kernelization to oracle protocols.

▶ **Definition 2.** [7] Given a language L, an oracle communication protocol for L is a two-player communication protocol. The first player gets an input x and can only execute computations taking time polynomial in $|x|$. The second player is computationally unbounded, but does not know x. At the end of the protocol, the first player has to decide correctly whether $x \in L$. The cost of the protocol is the number of bits of communication from the first player to the second player.

▶ **Proposition 3.** *[7] Let $d \geq 2$ be an integer, and ϵ be a positive real number. If* co-NP $\not\subseteq$ NP*/poly, then there is no protocol of cost $\mathcal{O}(n^{d-\epsilon})$ to decide whether a d-uniform hypergraph on n vertices has a d-hitting set of at most k vertices, even when the first player is co-nondeterministic.*

As noted in [7], this implies that for any $d \geq 2$ and any positive real number ϵ, if co-NP $\not\subseteq$ NP/poly, then there is no kernel of size $k^{d-\epsilon}$ for d-HITTING SET. In general, a lower bound for oracle communication protocols for a parameterized language L gives a lower bound for kernelization for L.

Kernels: size vs. number of elements

In literature, a lower bound on the kernel means the lower bound on the size of the kernel, but not necessarily on the number of input elements in the kernel. Kratsch et al. [15] were one of the first to study lower bounds in terms of the number of input elements in the kernel. They used the results of Dell and Melkebeek [7] along with results in two dimensional geometry to build a new technique to show lower bounds for the number of input elements in a kernel for a problem. In this paper, we have adhered to the general convention by saying that a kernel has a lower bound on its size if it has a lower bound on its representation in bits, while explicitly mentioning the cases where the kernel has a lower bound on the number of input elements.

3 Kernel Upper Bounds for primal parameter

In this section, we consider the Hyperplane Subset General Position problem in $\mathbb{R}^d$ parameterized by the primal parameter k. We describe a polynomial kernelization for this problem. This method is similar to that described in [14]. However, there is an error in the analysis of kernel size in [14]. Our proof, when restricted to the case of Line Subset General Position problem gives the correct bound on the kernel. We will point out the place where there is an error in [14], while describing our proof. Moreover, using the well-known Veronese mapping, we can generalize this result to give polynomial kernels for d-Polynomial Subset General Position in $\mathbb{R}^d$ parameterized by k.

3.1 Hyperplane case

First, we consider an easy variant of the Hyperplane Subset General Position problem in $\mathbb{R}^d$, where the input point set P is such that for every subset S of P of size less than d, $\mathsf{dim}(S) = |S| - 1$. In this case, the i-flats are said to be *non-degenerate*. In this case, parameterization by k gives us a polynomial kernel by a generalization of the results obtained in [14]. For the sake of completeness, we describe the kernelization. We apply a reduction rule that bounds the coplanarity of hyperplanes in $\mathbb{R}^d$.

▶ **Reduction Rule 4.** *Given an instance (P, k) of* Hyperplane Subset General Position *in $\mathbb{R}^d$, if there is a hyperplane H with at least $(d-1)\binom{k-d}{d} + d$ points then we delete all the points in $H \cap P$ and set $k = k - d$.*

▶ **Lemma 5.** *Reduction Rule 4 is safe.*

We apply this Reduction Rule exhaustively. In the end, we know that each hyperplane can contain $\mathcal{O}(k^d)$ input points. Together with the bound on coplanarity, we will also use the following result by Cardinal et al. [4, Theorem 4.1] to get a kernel.

▶ **Theorem 6** (Cardinal et al. [4])**.** *Let P be a set of n points in $\mathbb{R}^d$ with at most ℓ cohyperplanar points, where $\ell = O(\sqrt{n})$. Then P contains a subset of size $\Omega\left((n/\log l)^{1/d}\right)$ of points in general position.*

▶ **Theorem 7.** Hyperplane Subset General Position *in $\mathbb{R}^d$, parameterized by k and with an input point set where all lower dimensional flats are non-degenerate, has a $\mathcal{O}(k^{2d})$ kernel.*

Proof. We know from Theorem 6 that for a point set of size n and *cohyperplanarity* ℓ, i.e., at most ℓ points from the point set can lie on a given hyperplane, such that $\ell \leq \sqrt{n}$, there is a point set in general position of size at least $C(\frac{n}{\log \ell})^{1/d}$ where $C = C(d)$ is a constant. Thus, when $\ell \leq \sqrt{n}$, if $C(\frac{n}{\log \ell})^{1/d} > k$, we correctly say Yes. Substituting ℓ by its upper bound of $\mathcal{O}(k^d)$, this equation is true when $n \geq \Omega(k^{d+1})$. When $n = \mathcal{O}(k^{d+1})$, then anyway we obtain a kernel of size $\mathcal{O}(k^{d+1})$. The remaining case is when $\ell > \sqrt{n}$. Then, substituting ℓ by its upper bound of $\mathcal{O}(k^d)$, we know that $n = \mathcal{O}(k^{2d})$. Thus, we obtain the required polynomial kernel. ◀

Now we consider the general problem. To design a kernel for the general problem, point subsets lying in lower dimensional flats also have to be kept in mind. The approach is to first reduce the number of points that can lie in a lower dimensional flat before we can employ a strategy similar to the kernelization of Theorem 7. First, we describe a reduction rule such that in the reduced instance, the coplanarity of each i-flat with $i \leq d$ is bounded by a

function of k. This reduction rule is similar to Reduction Rule 4, except that it has to take care of point subsets lying in lower dimensional flats before it considers point subsets lying in hyperplanes of $\mathbb{R}^d$.

▶ **Reduction Rule 8.** *Let (P, k) be an instance of* Hyperplane Subset General Position *parameterized by k. Let i be the smallest integer between 2 and d, auch that there is an $(i-1)$-flat that contains at least $c(d) \cdot k^{id} + 1$ points. Then we delete all but $c(d)k^{id}$ points. Here $c(d) = 15(d-1)$ is a constant.*

▶ **Lemma 9.** *Reduction Rule 8 is safe.*

Proof sketch. We prove the correctness of the reduction rule by induction on i. In the base case, suppose $i = 2$. Suppose there is a line L containing at least $c(d)k^4 + 1$. Then by the reduction rule, we construct an instance (P', k) such that P is modified to P' by deleting all but $c(d)k^4$ points. We show that P has a k-sized set in general position if and only if P' has a k-sized set in general position. Since, $P' \subset P$, if P' has a k-sized set in general position, so does P.

In the forward direction, suppose P has a k-sized set S in general position. Let P_L be the set of points in $L \cap P$ and P'_L be the set of points in $L \cap P'$. By definition of general position, there are $\Sigma_{j \leq d}\binom{k}{j}$ flats of dimension at most d that can be formed by the points in S. Consider the intersection of these flats with the line L. Each intersection is of dimension at most 1. That is, if the intersection is not the line L itself, then it is a point in L. Thus there are at most $\Sigma_{j \leq d}\binom{k}{j}$ points of intersection. Since, the set S is in general position, at most two points from S lie on L. If there are no points or if all the points in $S \cap L$ belong to P' then S is also a k-sized set in general position for the instance (P', k). Otherwise, suppose there are at most $t \leq 2$ points from $P_L \setminus P'_L$. The number of intersection points is at most $\Sigma_{j \leq d}\binom{k}{j} < c(d)k^{2d}$. Thus there is a set $\hat{S} = \{p_t | t \leq 2\}$ on L that are not intersection points. Let $S' = S \setminus (P \cap L) \cup \hat{S}$. We show that S' is also a set in general position. Consider a flat defined by points from S'. If they do not contain points from $\hat{S}$, then they remain non-degenerate flats. Suppose a flat contains points from $\hat{S}$. Also, for the sake of contradiction, suppose the flat is degenerate. If the flat contains all the points in $\hat{S}$ then it contains the line L and therefore the points from $L \cap P$. Thus, in P the set S was not in general position, which is a contradiction. Now, suppose the flat F contains a single point, say p_1, from $\hat{S}$. Then the points from $F \cap S$ were in general position and therefore the flat was either the line L or $L \cap F$ was an intersection point. This contradicts the fact that p_1 belongs to F, as p_1 is chosen to be a point that is not an intersection point. Thus, $S' \subseteq P'$ is in general position and of size k. Note that this means that after exhaustive application of this rule all lines contain at most $\lambda_2 = c(d)k^{2d}$ points.

The arguments of the other values of i are similar but with more case analysis. The full proof can be found in the full version of this paper. ◀

This Reduction Rule is exhaustively applied and in the end the reduced instance is such that for any hyperplane in $\mathbb{R}^d$, the coplanarity is $\mathcal{O}(k^{d^2})$. Now, we can exhibit a polynomial kernel for Hyperplane Subset General Position in $\mathbb{R}^d$ parameterized by k. The proof is same as that of Theorem 7, while taking into account that the collinearity bound in this case is $\mathcal{O}(k^{d^2})$.

▶ **Theorem 10.** Hyperplane Subset General Position *in $\mathbb{R}^d$ parameterized by k has a kernel of size $\mathcal{O}(k^{2d^2})$.*

Using the techniques in the proof of Theorem 10 we can also solve the *projective version* of the general position problem. For a given point set P in $\mathbb{R}^d$, $S \subseteq P \setminus \{0\}$ is said to be in

projective general position if no more than $d-1$ points from S lie on a hyperplane in $\mathcal{R}^d$ that passes through the origin. The parameterized version of the problem is the following:

Projective Hyperplane Subset General Position	**Parameter:** k
Input: An n point set P in $\mathbb{R}^d$, for a fixed constant d, and a positive integer k	
Question: Is there a subset $S \subseteq P$ of size at least k such that S is in projective general position?	

Notice that this problem in $\mathbb{R}^2$ is polynomial time solvable, as the problem is equivalent to asking whether the projection of the points on a unit sphere, centered at the origin, equals to at least k points where no two lie on the same line through the origin.

We will apply the following reduction rule to reduce the coplanarity of the hyperplanes passing through the origin.

▶ **Reduction Rule 11.** *Let (P,k) be an instance of* Projective Hyperplane Subset General Position *parameterized by k. Let i be the smallest integer between 2 and $d-1$, such that there is an $(i-1)$-flat passing through the origin that contains at least $c'(d)\cdot k^{id}+1$ points. Then we delete all but $c'(d)k^{i(d-1)}$ points. Here, as in the case with Reduction Rule 8, $c'(d)$ is a large constant depending linearly on d.*

The correctness of this Reduction Rule is same as the inductive proof of Reduction Rule 8. Applying the above reduction rule exhaustively, as was the case with Hyperplane Subset General Position problem, we will get that any hyperplane passing through the origin has $O(k^{(d-1)^2})$ input points on them. To get a polynomial kernel for the Projective Hyperplane Subset General Position problem we will need the following analogue to Theorem 6 with bounded coplanarity.

▶ **Lemma 12** (Projective version of Theorem 6). *Let P be a set of n points in $\mathbb{R}^d$ such that for any hyperplane H passing through the origin, we have $|H\cap P|\leq \ell$, where $\ell = O(n^{1/3})$. Then P contains a subset of size $\Omega\left(\left(\frac{n^{2/3}}{\log \ell}\right)^{1/(d-1)}\right)$ of points in projective general position.*

Proof. The proof of this lemma will use Theorem 6. Without loss of generality we will assume that the hyperplane $X_1 = 1$, intersects all the lines passing through the origin and one point of P. Let L denotes the set of lines passing through the origin and one point of P. Observe that since for any hyperplane H passing through the origin, $|H\cap P|\leq \ell$, therefore $|L|\geq \frac{n}{\ell}$. Let P' be the set points we get by intersecting lines in L with the hyperplane $X_1 = 1$. Again observe that $|P'|\geq \frac{n}{\ell}$, and for any $(d-2)$-hyperplane H' contained in the hyperplane $X_1 = 1$, we have $|H'\cap P'|\leq \ell$, otherwise we can get a hyperplane passing through origin containing more than ℓ points from P. Using the fact that $\ell = O\left(\sqrt{n/\ell}\right)$ and Theorem 6, we get that P' contains a subset P'_1 of size $\Omega\left(\left(\frac{n^{2/3}}{\log \ell}\right)^{1/(d-1)}\right)$ with no more than ℓ points from P'_1 lying on any $(d-2)$-dimensional subflat of the hyperplane $X_1 = 1$. For each point $p'\in P'_1$, include in the set P_1 a point p from the set P such that p and p' are on the same line passing through the origin. By construction the set P_1 is in projective general position and $|P_1| = \Omega\left(\left(\frac{n^{2/3}}{\log \ell}\right)^{1/(d-1)}\right)$. ◀

Now, using the fact that any hyperplane passing through the origin has $O(k^{(d-1)^2})$ input points on them after application of Reduction Rule 11 and Lemma 12, we can prove the following result in the same way we proved Theorem 10.

▶ **Theorem 13.** Projective Hyperplane Subset General Position *in* $\mathbb{R}^d$ *parameterized by* k *has a kernel of size* $\mathcal{O}(k^{3(d-1)^2})$.

3.2 Bounded degree polynomials

The following lemma will show the direct connection between the d-Polynomial Subset General Position problem and Hyperplane Subset General Position problem:

▶ **Lemma 14.** *Let* P *be a set of points in* $\mathbb{R}^t$*, and* $\mathcal{F}$ *be a subspace of* $\text{Poly}_d[X_1, \ldots, X_t]$ *with a basis* $\{f_1(X), \ldots, f_b(X), 1\}$*, where* $X = (X_1, \ldots, X_t)$.

1. *If* P *is a set of* ℓ *points in general position with respect to the polynomial family* $\mathcal{F}$ *(defined earlier in the section) then* $\Phi_{\mathcal{F}}(P)$ *(*$\Phi_{\mathcal{F}}$ *is defined earlier in Section 2) is a set of* ℓ *points in general position with respect to hyperplanes in* $\mathbb{R}^b$.
2. *Let* $S = \{q_1, \ldots, q_\ell\} \subseteq \Phi_{\mathcal{F}}(P)$ *be a set of* ℓ *points in general position with respect to hyperplanes in* $\mathbb{R}^b$*. Then the set* $S' = \{p_1, \ldots, p_\ell\}$*, where* $p_i \in \Phi_{\mathcal{F}}^{-1}(q_i) \cap P$*, will be a set of* ℓ *points in general position with respect to* $\mathcal{F}$.

Proof.

1. First, observe that it is enough to show that the map $\Phi_{\mathcal{F}}$ is injective on P. In general, the map $\Phi_{\mathcal{F}}$ need not be an injective mapping on an arbitrary set of n points in $\mathbb{R}^t$. However, we show that Φ is injective when restricted to P if P is in general position with respect to $\mathcal{F}$. To reach a contradiction, let $\Phi_{\mathcal{F}}(p_1) = \Phi_{\mathcal{F}}(p_2)$ where $p_1, p_2 (\neq p_1) \in P$. Let $S \subseteq P$ be of size $b+1$ and $p_1, p_2 \in S$. Observe that the set $\Phi_{\mathcal{F}}(S)$ will have less than $b+1$ points and this will imply that there exists a hyperplane $\sum_{i=1}^{b} \lambda_i Z_i + \lambda_{b+1} = 0$ on which the set $\Phi_{\mathcal{F}}(S)$ will lie. But this implies that the polynomial $\sum_{i=1}^{b} \lambda_i f_i(X) + \lambda_{b+1} = 0$ will be satisfied by all the points in S. This is a contradiction to the fact that the point set P is in general position.
2. This result follows directly from the construction of the mapping $\Phi_{\mathcal{F}}$. ◀

With the above result and Theorem 10 we get the following theorem.

▶ **Theorem 15.** d-Polynomial Subset General Position *in* $\mathbb{R}^t$ *parameterized by* k*, has a polynomial kernel of size* $\mathcal{O}(k^{2b^2})$*. Here* b *is the size of the basis generating the underlying vector space of polynomials.*

As in the case with Theorem 10 we can also get a projective version of Theorem 15. Let $\mathcal{F}$ be a subspace of $\text{Poly}_d[X_1, \ldots, X_t]$ with basis $\{f_1(X), \ldots, f_b(X)\}$ where $X = (X_1, \ldots, X_t)$ and none of the polynomial functions $f_i(X)$ are constants. Then we can define projective analog of the general position problem for polynomial families like $\mathcal{F}$. For a given point set P in $\mathbb{R}^t$, a subset S of P will be in general position with respect to $\mathcal{F}$ if no more than $b-1$ points from S lie on any $f(X) \in \mathcal{F}$.

Using the same techniques as in the proof of Theorem 10 we will get the following result:

▶ **Corollary 16.** Projective Polynomial Subset General Position *in* $\mathbb{R}^t$ *parameterized by* k *has a kernel of size* $\mathcal{O}(k^{3(b-1)^2})$.

Upper bounds for non-vector space families

Observe that we may be interested in getting general position point set with respect to families of polynomials that are not vector spaces of $\text{Poly}_d[X_1, \ldots, X_t]$. For example, consider the case of hyperplanes in R^t of the following type $H := \sum_{i=1}^{t-1} \lambda_i X_i + X_t + \beta$, where λ_i's and β

are in $\mathbb{R}^+$. One might be interested in getting a general position set in $\mathbb{R}^t$ with respect to these hyperplanes.

Note that our upper bound on the kernel size in the primal parameter extends to these families as well.

▶ **Corollary 17.** *Let $\mathcal{F}$ be a subfamily of $\text{Poly}_d[X_1, \ldots, X_t]$ such that there exists polynomial functions $f_1(X), \ldots, f_b(X)$ in $\text{Poly}_d[X_1, \ldots, X_t]$ and for any $f(X) \in \mathcal{F}$, $f(X) = \sum_{i=1}^{b} \lambda_i f_i(X)$ where the λ_i's are in $\mathbb{R}$. Subset general position problem with respect to $\mathcal{F}$ parameterized by primal parameter k has a kernel of size $\mathcal{O}(k^{3(b-1)^2})$.*

4 Tight kernels for hyperplanes in dual parameter

In this Section, we show that Hyperplane Subset General Position in $\mathbb{R}^d$, parameterized by the dual parameter h, cannot have a kernel of size $h^{d-\epsilon}$ if co-NP $\not\subseteq$ NP/poly. We show this result by the standard technique of polynomial parameter transformation. For a fixed d, we reduce the d-Hitting Set problem on d-uniform graphs to the problem of Hyperplane Subset General Position in $\mathbb{R}^d$. By Proposition 3, this gives us a lower bound for Hyperplane Subset General Position in $\mathbb{R}^d$.

For the main result, we construct for each positive integer n and each d, a set of n points in $\mathbb{R}^d$ with some special properties.

▶ **Lemma 18.** *For every d-uniform hypergraph G in n vertices and m hyperedges, there is a transformation to a set $P \uplus B = \{p_1, p_2, \ldots, p_n\} \uplus \{b_1, b_2, \ldots, b_m\}$ of $n + m$ points in $\mathbb{R}^d$ that have the following properties:*

1. *The points $\{p_1, p_2, \ldots, p_n\}$ are in general position.*
2. *Each vertex $v_i \in V(G)$ is mapped to the point $p_i \in P$.*
3. *Each hyperedge $e_j \in E(G)$ is mapped to the point $b_j \in B$.*
4. *For a hyperedge $e_j \in E(G)$, if there are d points $\{p_{i_1}, p_{i_2}, \ldots, p_{i_d}\} \in P$ such that $b_j, p_{i_1}, p_{i_2}, \ldots, p_{i_d}$ are cohyperplanar, then it must be the case that $e_j = \{v_{i_1}, v_{i_2}, \ldots, v_{i_d}\}$.*
5. *For any set of $i \leq d$ points of B and $d + 1 - i$ points of P cannot be cohyperplanar.*
6. *The points in B are in general position. In other words, no $d + 1$ points in B are cohyperplanar.*

Proof Idea. The main idea behind the proof is that the sets P and B satisfying the conditions of Lemma 18 can be generated in a greedy manner from considering large grids. We will first construct the point set P of n points in a greedy manner such that P comes from a large grid and is in general position. After P is constructed, the set B is again greedily constructed, this time using a lower dimensional grid lying in a particular hyperplane. ◀

This transformation from a graph to a point set leads to the following observation.

▶ **Observation 19.** *Let G be a d-uniform hypergraph and $P \uplus B$ be the set of points in $\mathbb{R}^d$ corresponding to G. For any maximal set S of points in general position, there is a set of size at least $|S|$ that contains all the points in B.*

This helps us to design a reduction from d-Hypergraph Independent Set to Hyperplane Subset General Position in $\mathbb{R}^d$.

▶ **Lemma 20.** *There is a many-one reduction from d-Hypergraph Independent Set on d-uniform hypergraphs to Hyperplane Subset General Position in $\mathbb{R}^d$.*

Finally, we are ready to prove the main result.

▶ **Theorem 21.** Hyperplane Subset General Position *in* $\mathbb{R}^d$ *parameterized by the dual parameter cannot have a kernel of size* $\mathcal{O}(h^{d-\epsilon})$ *if* co-NP $\not\subseteq$ NP*/poly.*

Proof. We give a reduction from d-Hitting Set on d-uniform hypergraphs. Given an instance (G, h) of d-Hitting Set on d-uniform hypergraphs, we consider the equivalent d-Hypergraph Independent Set instance $(G, |V(G)| - h)$. By Lemma 20, we construct an instance $(P \uplus B, |V(G)| + |E(G)| - h)$. Note that the transformation is such that $|P| = |V(G)|$ and $|B| = |E(G)|$. Thus, G has a d-hitting set of size k if and only if G has an independent set of size $|V(G)| - h$, which by Lemma 20 happens if and only if $P \uplus B$ has a point subset of size $k' = |P \uplus B| - h$ that is in general position with respect to hyperplanes in $\mathbb{R}^d$. This means that the dual parameter $|P \uplus B| - k'$ is equal to h, which is the d-hitting set size in G. This implies the lower bound on the kernel size of Hyperplane Subset General Position in $\mathbb{R}^d$ parameterized by the dual parameter. ◀

We obtain tight polynomial kernels from the following Proposition, derived from a folklore result.

▶ **Proposition 22.** Hyperplane Subset General Position *in* $\mathbb{R}^d$ *parameterized by the dual parameter* h *has a kernel of size* h^d*.*

Proof. We state the folklore reduction from Hyperplane Subset General Position in $\mathbb{R}^d$ to $(d+1)$-Hitting Set. Given an instance (P, h) of Hyperplane Subset General Position in $\mathbb{R}^d$, we construct the following instance (G, h) of $(d+1)$-Hitting Set. Corresponding to each point in P we create a vertex in $V(G)$. For any $d+1$ point in P that are coplanar in $\mathbb{R}^d$, we create a hyperedge with the corresponding vertices. Consider the vertices in a hyperedge of G. At least one of the corresponding points has to be deleted in order to construct a subset of P that is in general position with respect to hyperplanes in $\mathbb{R}^d$. Thus, the set of points deleted correspond to a hitting set of G. Therefore, the reduction is correct.

$(d+1)$-Hitting Set has a kernel where the universe size if $\mathcal{O}(h^d)$ [19]. This gives us an $\mathcal{O}(h^d)$ kernel for Hyperplane Subset General Position in $\mathbb{R}^d$ parameterized by the dual parameter. ◀

Lower bounds on elements in a kernel for hyperplanes in dual parameter

In this section, we show that by the method suggested by Dell and Melkebeek [7], we can show a lower bound on the number of points in a polynomial kernel for Hyperplane Subset General Position in $\mathbb{R}^d$, for each fixed positive integer d. This result is a direct extension of the results obtained in [15] and [14].

▶ **Theorem 23.** Hyperplane Subset General Position *in* $\mathbb{R}^d$*, parameterized by* h*, cannot have a kernel with* $\mathcal{O}(h^{d-\epsilon})$ *points if* co-NP $\not\subseteq$ NP*/poly.*

5 Bounded degree polynomials and the dual parameter

In this section we discuss about the generalization of Theorems 21 and 23. Note that, for any given point set P with n points, both theorems can be proved for finding a point set of size $n - h$, h being the dual parameter, if the construction of Lemma 18 can be replicated for a particular family $\mathcal{F}$. In particular, consider a family $\mathcal{F}$ of polynomials of degree at most d with basis $\{f_1(X), \ldots, f_b(X), 1\}$, where $X = \{X_1, \ldots, X_t\}$. Suppose for each b-uniform hypergraph G with n vertices and m edges it is possible to make a transformation as follows:

1. The points $\{p_1, p_2, \ldots, p_n\}$ are in general position with respect to $\mathcal{F}$ and have bounded representation.
2. Each vertex $v_i \in V(G)$ is mapped to the point $p_i \in P$.
3. Each hyperedge $e_j \in E(G)$ is mapped to the point $b_j \in B$.
4. For a hyperedge $e_j \in E(G)$, if there are d points $\{p_{i_1}, p_{i_2}, \ldots, p_{i_b}\} \in P$ such that $b_j, p_{i_1}, p_{i_2}, \ldots, p_{i_b}$ lie on a polynomial from $\mathcal{F}$, then it must be the case that $e_j = \{p_{i_1}, p_{i_2}, \ldots, p_{i_b}\}$.
5. For any set of $i \leq b$ points of B and $b+1-i$ points of P cannot be on any polynomial of $\mathcal{F}$.
6. The points in B are in general position. In other words, no $b+1$ points in B can be on any polynomial of $\mathcal{F}$.

Let us call such a transformation a *good* transformation. Then with respect to such a family $\mathcal{F}$, equivalent tight kernelizations for the dual parameter can be given. When $\mathcal{F}$ is the family of spheres, then such a construction is possible.

▶ **Corollary 24.** *Given the family of spheres in $\mathbb{R}^d$, for each $(d+1)$-uniform hypergraph with n vertices and m points there is a good transformation to a set $P \uplus B$ of $n+m$ points.*

Proof. The construction is similar as that of Lemma 18, as we again can construct the sets greedily. The point set P can be extracted from a large enough grid as in the construction given in Lemma 18. The construction of the points in B is also done inductively. Suppose the points of a subset $B' \subset B$ have already been placed on rational points such that all the necessary conditions are satisfied. Let $b_e \in B \setminus B'$. Consider the sphere S_e defined by the d-sized point set P_e corresponding to the vertices of $e \in E(G)$. Consider the family $\mathcal{F}$ of spheres formed by (i) a set of any d points in P other than the set P_e, (ii) any d points from B', and (iii) a set S of d points with at least one point from P and at least one point from B'. The intersection of this family of spheres with S_e is a family $\mathcal{F}'$ of lower dimensional spaces. Since the points in P have bounded representation, so do the intersection spaces. It is possible to determine in polynomial time the arrangement of the lower dimensional intersections on S_e [1]. From this arrangement, we select a point with bounded representation that does not belong to any of the lower dimensional flats in $\mathcal{F}'$ and set the point to b_e. The set $B' \cup b_e$ again satisfies all the necessary conditions. We continue till all the points in B have been determined. Thus, we construct the required set $P \uplus B$. ◀

6 Open Problems

One of the major open questions in this area is regarding techniques for showing kernel lower bounds with respect to the primal parameter. Currently, no non-trivial lower bound is known for even Hyperplane Subset General Position in $\mathbb{R}^2$.

In Section 5, we gave tight lower bounds with respect to the dual parameter under some restricted vector spaces of d-degree polynomials. It will be interesting to understand the problem better for general vector spaces of d-degree polynomials. In fact, it might be useful for both algorithmic as well as combinatorial studies to understand the Geometric Subset General Position in both the primal and dual parameter for families of d-degree polynomials that are not vector spaces or a subset of a vector space. We are interested in studying families that fall outside these frameworks.

General position with respect to spheres is also connected to Delaunay Subset Selection problem where we are given a point set $P \subset \mathbb{R}^d$ as input and the problem is to extract a maximum size subset S of P such that the Delaunay complex $\mathrm{Del}(S)$ is a triangulation

of the convex hull $\text{conv}(S)$ of S. Although the upper bounds for kernelization given in this paper hold for this problem, lower bound questions remain open for both primal and dual parameter.

References

1 Pankaj K Agarwal and Micha Sharir. Arrangements and their applications. *Handbook of Computational Geometry*, pages 49–119, 2000.

2 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011.

3 C. Cao. Study on Two Optimization Problems: Line Cover and Maximum Genus Embedding. Master's thesis, Texas A&M University, May 2012.

4 Jean Cardinal, Csaba D. Tóth, and David R. Wood. General position subsets and independent hyperplanes in d-space. *Journal of Geometry*, pages 1–11, 2016. doi:10.1007/s00022-016-0323-5.

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

6 Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.

7 Holger Dell and Dieter Van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 251–260. ACM, 2010.

8 Herbert Edelsbrunner and Leonidas J. Guibas. Topologically Sweeping an Arrangement. *J. Comput. Syst. Sci.*, 38(1):165–194, 1989.

9 Herbert Edelsbrunner and Leonidas J. Guibas. Corrigendum: Topologically Sweeping an Arrangement. *J. Comput. Syst. Sci.*, 42(2):249–251, 1991.

10 Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

11 Herbert Edelsbrunner, Joseph O'Rourke, and Raimund Seidel. Constructing Arrangements of Lines and Hyperplanes with Applications. *SIAM J. Comput.*, 15(2):341–363, 1986.

12 Jeff Erickson and Raimund Seidel. Better Lower Bounds on Detecting Affine and Spherical Degeneracies. *Discrete & Computational Geometry*, 13:41–57, 1995.

13 Jeff Erickson and Raimund Seidel. Erratum to Better Lower Bounds on Detecting Affine and Spherical Degeneracies. *Discrete & Computational Geometry*, 18(2):239–240, 1997.

14 Vincent Froese, Iyad Kanj, André Nichterlein, and Rolf Niedermeier. Finding Points in General Position. *CCCG*, pages 7–14, 2016.

15 Stefan Kratsch, Geevarghese Philip, and Saurabh Ray. Point Line Cover: The Easy Kernel is Essentially Tight. *ACM Trans. Algorithms*, 12(3):40, 2016.

16 Stefan Langerman and Pat Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.

17 Jiří Matoušek. *Lectures on Discrete Geometry*, volume 108. Springer New York, 2002.

18 Michael S Payne and David R Wood. On the general position subset selection problem. *SIAM Journal on Discrete Mathematics*, 27(4):1727–1733, 2013.

19 René van Bevern. Towards optimal and expressive kernelization for d-hitting set. *Algorithmica*, 70(1):129–147, 2014.

Satisfiable Tseitin Formulas Are Hard for Nondeterministic Read-Once Branching Programs*

Ludmila Glinskih[1] and Dmitry Itsykson[2]

1 **St. Petersburg Department of V.A. Steklov Institute of Mathematics of the Russian Academy of Sciences and St. Petersburg Academic University, Russia**
lglinskih@gmail.com

2 **St. Petersburg Department of V.A. Steklov Institute of Mathematics of the Russian Academy of Sciences, Russia**
dmitrits@pdmi.ras.ru

Abstract

We consider satisfiable Tseitin formulas $\mathrm{TS}_{G,c}$ based on d-regular expanders G with the absolute value of the second largest eigenvalue less than $\frac{d}{3}$. We prove that any nondeterministic read-once branching program (1-NBP) representing $\mathrm{TS}_{G,c}$ has size $2^{\Omega(n)}$, where n is the number of vertices in G. It extends the recent result by Itsykson at el. [9] from OBDD to 1-NBP.

On the other hand it is easy to see that $\mathrm{TS}_{G,c}$ can be represented as a read-2 branching program (2-BP) of size $O(n)$, as the negation of a nondeterministic read-once branching program (1-coNBP) of size $O(n)$ and as a CNF formula of size $O(n)$. Thus $\mathrm{TS}_{G,c}$ gives the best possible separations (up to a constant in the exponent) between 1-NBP and 2-BP, 1-NBP and 1-coNBP and between 1-NBP and CNF.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes - Relations among complexity measures, F.2.2 Nonnumerical Algorithms and Problems - Complexity of proof procedures

Keywords and phrases Tseitin formula, read-once branching program, expander

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.26

1 Introduction

1.1 Satisfiable and unsatisfiable Tseitin formulas

A Tseitin formula $\mathrm{TS}_{G,c}$ is defined for every undirected graph $G(V, E)$ and labelling function $c : V \to \{0, 1\}$. We introduce a propositional variable for every edge of G. The Tseitin formula $\mathrm{TS}_{G,c}$ represents a linear system over the field GF(2) that for every vertex $v \in V$ states that the sum of all edges adjacent to v equals $c(v)$.

A Tseitin formula is satisfiable if and only if the sum of values of the labeling function for all vertices in every connected component is even [17]. The study of Tseitin formulas is motivated by Proof Complexity. Proof Complexity basically deal with unsatisfiable Tseitin formulas that roughly speaking encode that it is impossible that a graph has an odd number of vertices with odd degree. It is important for Proof Complexity that propositional formulas have small CNF representations; thus it is usually assumed that G has constant degree;

* The research was supported by Russian Science Foundation (Project 16-11-10123).

for such graphs Tseitin formulas have CNF representations of size $O(n)$ and it contains $O(n)$ variables where n is the number of vertices in G. Tseitin formulas were invented by Tseitin in 1968 for the graph of $n \times n$ cellular square and were used for the proving of the first superpolynomial lower bound for regular resolution. In 1987 Urquhart extended this result and proved exponential lower bound on the complexity of resolution refutations of Tseitin formulas based on expanders. Unsatisfiable Tseitin formulas are one of the basic examples of hard formulas for many proof systems; in particular, Tseitin formulas are hard for bounded depth Frege [3], [15], Polynomial Calculus over the field $\mathbb{F}$ with $\mathrm{char}(\mathbb{F}) \neq 2$, tree-like Lovasz-Schrijver proof system [8], etc.

Satisfiable Tseitin formulas have been studied less intensively. In the recent paper by Itsykson at el. [9] satisfiable Tseitin formulas appeared in the proof of an exponential lower bound on the size of the derivation of unsatisfiable Tseitin formulas in the proof system OBDD(join, reordering). An OBDD is a partial case of a read-once deterministic branching program, where in every path from the source to a sink all variables appear in the same order. The key step in the proof of the mentioned lower bound is the proof of an exponential lower bound on the OBDD representation of satisfiable Tseitin formulas based on constant degree expanders. The latter lower bound motivated us for the current research. There are known examples of Boolean functions that are easy for read-once branching program but hard for OBDD (see for example Theorem 6.1.2 in [18]). Is it possible to extend the mentioned lower bound from OBDD to read-once branching program?

It is well known that the size of the shortest regular resolution proof of any unsatisfiable CNF formula ϕ equals the size of the minimal read-once branching program for the following search problem $Search_\phi$: given an assignment of variables of ϕ, find a clause that is refuted by this assignment [12]. Thus lower bounds for the size of resolution proofs of ϕ implies lower bounds on the size of read-once branching program for $Search_\phi$. However it is unclear whether the sizes of read-once branching programs for $Search_\phi$ for unsatisfiable Tseitin formula ϕ and for the evaluation of a satisfiable Tseitin formula are connected. The difference is the following:

1. the first case is about unsatisfiable Tseitin formulas while the second case is about satisfiable Tseitin formulas;
2. to find a clause that is refuted may be harder than just to say that the value of a function is 0.

1.2 Results

In this paper we prove that every nondeterministic read-once branching program (1-NBP) representing a satisfiable Tseitin formula $\mathrm{TS}_{G,c}$ based on d-regular expander with the absolute value of the second largest eigenvalue less than $\frac{d}{3}$ has size $2^{\Omega(n)}$, where n is the number of vertices in G and d is a constant. As a corollary we get a lower bound $2^{\Omega(n)}$ on the size of nondeterministic read-once branching programs for Tseitin formulas based on complete graph K_n. All mentioned lower bounds are tight up to a constant in the exponent since every satisfiable Tseitin formula based on graph with n vertices and m edges may be represented as OBDD of size $O(m2^n)$ (see Proposition 3 below).

1.3 Comparison with other works

If we consider a Tseitin formula as a system of linear equations then every variable will have exactly two occurrences. Therefore by straightforward transformation every satisfiable Tseitin formula $\mathrm{TS}_{G,c}$ may be represented as read-2 deterministic branching program

(2-BP) of size $O(m)$, where m is the number of edges in G. Thus satisfiable Tseitin formulas based on constant-degree expanders strongly exponentially separate 1-NBP and 2-BP. And this separation is optimal up to a constant in the exponent. Consider a function $\text{CLIQUE\_ONLY}_n : \{0,1\}^{n(n-1)/2} \to \{0,1\}$ that detect whether a undirected graph on n vertices is exactly a clique on $\lfloor n/2 \rfloor$ vertices. Borodin, Razborov and Smolensky [4] proved that any nondeterministic read-once branching program representing CLIQUE_ONLY_n has size $2^{\Omega(n)}$ (note that CLIQUE_ONLY_n depends on $\Theta(n^2)$ variables) while there is a deterministic read-twice branching program of size $\text{poly}(n)$. Thathachar [16] gave, for every natural k, an explicit function that can be evaluated by deterministic read-$(k+1)$ branching program of linear size but every nondeterministic read-k branching program for this function has size at least $2^{\Omega\left(n^{1/(k+1)}\right)}$. As far as we know, the best previously known gap between sizes of 1-NBP and 2-BP was $2^{\Omega(\sqrt{n})}$ and we improved it to $2^{\Omega(n)}$.

First explicit Boolean function with strongly exponential lower bound on the size of 1-NBP was constructed in [4], however this function was rather artificial. Duros at el. [7] proved strongly exponential lower bounds on the size of 1-NBP for the function $\oplus cl_{3,n}$ that computes the parity of the number of triangles in the graph (and this extends the result of Babai at el.[2] from 1-BP to 1-NBP) and for the function $\Delta_{3,n}$ that is true iff the input graph does not contain triangles. So satisfiable Tseitin formulas based on constant-degree expanders is one more natural example of functions that require strongly exponential 1-NBP.

A satisfiable Tseitin formula based on a d-regular graph on n vertices is a characteristic function of an affine subspace of $\{0,1\}^{dn/2}$. Characteristic functions of affine (linear) subspaces were already studied in the context of complexity of deterministic and nondeterministic reak-k branching programs, namely characteristic functions of linear error-correcting codes were studied by Okolnishnikova [14] and Jukna [10]. Jukna [10] proved lower bound $2^{\Omega(\sqrt{n})}$ on the size of nondeterministic read-k branching program for characteristic functions of error-correcting codes $\mathcal{C} \subseteq \{0,1\}^n$. Duris at el. [7] presented a probabilistic construction of a linear code such that its characteristic function require 1-NBP of the size at least $2^{\Omega(n)}$. Jukna [10] noted that the negation of a characteristic function of an affine subspace may be represented by linear size (in the size of linear system that defines this subspace) nondeterministic read-once branching program. Indeed we just need to guess an equation that is not satisfied and then check this equation. Duris at el. [7] also showed that the characteristic function of a linear subspace of $\{0,1\}^n$ (and hence it is also true for affine subspaces) may be represented by a randomized read-once branching program with one-sided error 2^{-r} of size $O(n^r)$ for all natural r. As far as we know, satisfiable Tseitin formulas based on explicit constant-degree expanders are the only example of *explicit* functions (randomized construction was presented in [7]) that strongly exponentially separate nondeterministic and co-nondeterministic read-once branching program. And also our separation between nondeterministic and randomized read-once branching program seems to be the best known for the explicit functions.

We finally note that Tseitin formulas based on constant degree graphs may be represented as CNF formulas of size $O(n)$, so we get a strongly exponential separation between sizes of 1-NBP and CNF.

2 Preliminaries

2.1 Branching programs

A deterministic branching program (BP) is a form of representation of Boolean functions. A Boolean function $\{0,1\}^n \to \{0,1\}$ is represented by a directed acyclic graph with exactly one source and two sinks. All nodes except sinks are labeled with a variable; every internal

node has exactly two outgoing edges: one is labeled with 1 and the other is labeled with 0. One of the sinks is labeled with 1 and the other is labeled with 0. The value of the function for a given values of variables is evaluated as follows: we start a path from the source such that for every node on its path we go along the edge that is labeled with the value of the corresponding variable. This path will end in a sink. The label of this sink is the value of the function.

A nondeterministic branching program (NBP) differs from a deterministic in the way that we also allow guessing nodes that are unlabeled and have two outgoing unlabeled edges. So nondeterministic branching program may have three type of nodes: guessing nodes, nodes labeled with a variable (we call them just labeled nodes) and two sinks; the source may be either a guessing node or labeled node. The result of a function represented by a nondeterministic branching program equals 1, if there exists at least one path from the source to the sink labeled with 1 such that for every node labeled with a variable on its path we go along an edge that is labeled with the value of the corresponding variable, while for guessing nodes we are allowed to choose any of two outgoing edges.

Note that deterministic branching programs constitute a special case of nondeterministic branching programs.

A deterministic or nondeterministic branching program is (syntactic) read-k (k-BP or k-NBP) if every path from the source to a sink contains at most k occurrences of every variable.

An ordered binary decision diagram (OBDD) is a partial case of 1-BP, where on every path from the source to a sink all variables appear in the same order.

2.2 Tseitin formulas

Let $G(V, E)$ be an undirected graph without loops but possibly with multiple edges, $c : V \to \{0, 1\}$ be a labeling function that matches every vertex with a boolean value. Let us match every edge $e \in E$ with a propositional variable x_e. Tseitin formula $\mathrm{TS}_{G,c}$ based on a graph G and a labeling function c is the conjunction of the following conditions: for every vertex v the sum of variables x_e for all edges e that are incident to v equals $c(v)$ modulo 2. More formally: $\bigwedge\limits_{v \in V} \left(\sum\limits_{e \text{ is incident to } v} x_e = c(v) \bmod 2 \right)$.

If the maximal degree of a graph G is bounded by a constant d, then a sum modulo 2 can be written as a d-CNF formula with size at most $O(2^d d)$. Hence the size of CNF representation of $\mathrm{TS}_{G,c}$ does not exceed $O(2^d dn)$.

We will use the following criterion of the satisfiability of Tseitin formulas:

▶ **Proposition 1** ([17]). A Tseitin formula $\mathrm{TS}_{G,c}$ is satisfiable if and only if for every connected component U the following holds: $\sum\limits_{v \in U} c(v) = 0 \bmod 2$.

▶ Remark. Note that a substitution of a value to a variable $x_e := \alpha$ transforms Tseitin formula $\mathrm{TS}_{G,c}$ to a Tseitin formula $\mathrm{TS}_{G',c'}$, where graph G' is obtained from the graph G by deleting the edge e, c' equals c in every vertex except two vertices that are incident to edge e. On these two vertices the values of c and c' differ by α. In particular it follows that if $\mathrm{TS}_{G,c}$ is satisfiable and an edge e is not a bridge in the graph G, then the formula $\mathrm{TS}_{G',c'}$ is also satisfiable by Proposition 1 since the parity of sum of labels in G' in every connected component is the same as in G.

▶ **Lemma 2.** *Let $G(V, E)$ be a graph with k connected components. If the Tseitin formula $\mathrm{TS}_{G,c}$ is satisfiable, then the number of its satisfying assignments equals $2^{|E|-|V|+k}$.*

Proof. Let us fix some spanning forest F of the graph G; F contains exactly $|V| - k$ edges. Consider some partial substitution ρ to the edges of G that are not in F. By the Remark we know that after the application of the partial substitution ρ to $\mathrm{TS}_{G,c}$ we will get a satisfiable Tseitin formula based on the graph F. Since F is a forest the resulting Tseitin formula has exactly one satisfying assignment. Indeed a forest always has a vertex with degree 1 which helps us unambiguously determine the value of the incident edge. After that we can delete this edge from the forest and we will get a forest again; and so on. Hence the number of satisfying assignment of $\mathrm{TS}_{G,c}$ equals the number of different partial substitutions to the edges that are not in F; so the number of satisfying assignment equals $2^{|E|-|V|+k}$. ◀

▶ **Proposition 3.** Any satisfiable Tseitin formula based on a graph with n vertices and m edges can be represented as OBDD of size $O(m2^n)$.

Proof. Let us fix some order on the edges of the graph. The described OBDD will have m levels. Nodes on the i-th level are labeled with i-th edge of the graph.

Assume that we already ask for the value of the first $i-1$ edges. For every vertex of the graph we compute the sum modulo 2 of values of edges from these $i-1$ that are incident to the vertex. So we will have a vector of n parities. The i-th level of the OBDD contains 2^n nodes corresponding to the all possible values of vector of parities that we get after the reading of the first $(i-1)$ edges. Every node on the i-th level has two outgoing edges to nodes on the $(i+1)$-th level corresponding to the way how values on the edges change the parity of vertices. The node on the first level corresponding to all zero values of parities is the source of the OBDD (all nodes that are not reachable from the source should be removed). Outgoing edges for every node on the last level will go to a sink corresponding to the fact, whether the labeling function of the Tseitin formula is consistent with the resulting values of parities. ◀

▶ **Proposition 4.** 1) Every two satisfying assignments of a satisfiable Tseitin formula $\mathrm{TS}_{G,c}$ differ in at least two positions. 2) Every path from the source to the sink labeled with 1 in 1-NBP representing a satisfiable Tseitin formula $\mathrm{TS}_{G,c}$ contains variables for all edges of G.

Proof. 1) If we change a value of any edge in a satisfying assignment of $\mathrm{TS}_{G,c}$, the parity condition will be violated on two ends of this edge. 2) Assume that some acceptance path does not contains a variable x. Then there are two satisfying assignments of $\mathrm{TS}_{G,c}$ that differ only in the value of the variable x; this contradicts item 1. ◀

2.3 Expanders

Let $G(V,E)$ be an undirected graph without loops but possibly with multiple edges. G is an algebraic (n,d,α)-expander if G is d-regular, $|V| = n$ and the absolute value of the second largest eigenvalue of the adjacency matrix of G is not greater than αd.

It is well known that for all $1 > \alpha > 0$ and all large enough constants d there exist natural number n_0 and a family $\{G_n\}_{n=n_0}^{\infty}$ of (n,d,α)-algebraic expanders. There are explicit constructions such that G_n can be constructed in $\mathrm{poly}(n)$ time [13]. Also, it is known that a random d-regular graph is an expander with high probability.

Let us denote by $E(A,B)$ a multiset of edges that have one end in A and another end in B. Note that in the case where both ends of an edge are simultaneously in A and in B, we count this edge twice.

▶ **Lemma 5** (Cheeger inequality [5]). *Let $G(V,E)$ be an (n,d,α)-expander. Then for all $A \subseteq V$ such that $|A| \le \frac{n}{2}$ the following inequality holds: $|E(A, V \setminus A)| \ge \frac{1-\alpha}{2} d|A|$.*

▶ **Corollary 6.** *Every (n, d, α)-expander with $0 < \alpha < 1$ is connected.*

Proof. If G is not connected, then we will get a contradiction with Lemma 5 if we choose A to be a smallest connected component. ◀

▶ **Lemma 7** (Expander mixing lemma [1])**.** *Let $G(V, E)$ be (n, d, α)-expander, $A, B \subseteq V$. Then $\left||E(A, B)| - \frac{d|A||B|}{n}\right| \leq \alpha d\sqrt{|A||B|}$.*

Using Lemma 7 we can improve the estimation of the number of edges that go from A to the complement of A for small sets A.

▶ **Proposition 8.** For every (n, d, α)-expander for every $A \subseteq V$ the following inequality holds: $|E(A, V \setminus A)| \geq d|A|(1 - \frac{|A|}{n} - \alpha)$.

Proof. $|E(A, V \setminus A)| = |E(A, V)| - |E(A, A)| = d \cdot |A| - |E(A, A)| \geq d \cdot |A|(1 - \frac{|A|}{n} - \alpha)$. The last inequality follows from Lemma 7. ◀

3 Lower bound

Our main goal is to prove the following theorem:

▶ **Theorem 9.** *Let $G(V, E)$ be an algebraic (n, d, α)-expander, where $\alpha < \frac{1}{3}$. Let $\mathrm{TS}_{G,c}$ be a satisfiable Tseitin formula. Then the size of every* 1-NBP *that represents $\mathrm{TS}_{G,c}$ is $2^{\Omega(n)}$.*

Let us describe the plan of the proof. Consider a minimal 1-NBP that evaluates $\mathrm{TS}_{G,c}$. For every node of this branching program, except the sink labeled with 0 there exists a path to the sink labeled with 1. In the opposite case this node could be merged with a sink labeled with 0 and it would decrease the size of the 1-NBP.

For nondeterministic branching program, by *the length of a path* we will mean the number of labeled edges in it (i.e. we do not count outgoing edges from guessing nodes). For every labeled node v in a branching program we define its level as the minimal length of paths from the source to v. We choose a level $l = \Omega(n)$ and prove that the minimal 1-NBP contains many label nodes on the level l. The proof consists of two parts:

1. We show that every minimal 1-NBP that evaluates $\mathrm{TS}_{G,c}$ contains at least $2^{C_1 n}$ paths of length l from the source to a labeled node that correspond to different partial substitutions, where C_1 is a constant.
2. We show that in every minimal 1-NBP that evaluates $\mathrm{TS}_{G,c}$ for every labeled node v on the level l there are at most $2^{C_2 n}$ different partial substitutions that correspond to different paths from the source to the vertex v, where $C_2 < C_1$ is a constant.

These two propositions imply that the number of label nodes on the level l is at least $2^{(C_1 - C_2)n}$.

3.1 Lower bound on the number of paths

In this section we perform the first part of the plan and estimate the number of paths of length l from the source of the minimal 1-NBP that end in a labeled node and correspond to different partial substitutions.

▶ **Lemma 10.** *Let $G(V, E)$ be a connected graph. Let k be the maximum number of connected components that can be obtained after deleting of l edges from G. Then every minimal* 1-NBP *evaluating a satisfiable Tseitin formula $\mathrm{TS}_{G,c}$ contains at least $2^{l-(k-1)}$ paths of length l from the source that end in a labeled node and correspond to different partial substitutions.*

Proof. By Lemma 2 the number of satisfying assignments of the formula $\mathrm{TS}_{G,c}$ equals $2^{|E|-|V|+1}$.

For every satisfying assignment of $\mathrm{TS}_{G,c}$ there exists a path in the minimal 1-NBP from the source to the sink labeled with 1 of length $|E|$ that is consistent with the assignment. By Proposition 4 it is impossible that there are paths from the source to the sink labeled with 1 that are shorter than $|E|$. Let P be the set of paths from the source to the sink labeled with 1 such that for every satisfying assignment of $\mathrm{TS}_{G,c}$ there are exactly one path in P that represents this assignment.

We estimate the number of paths in P that define the same partial substitution ρ that corresponds to the first l labeled edges of the path.

If we apply ρ to $\mathrm{TS}_{G,c}$ we will get a Tseitin formula $\mathrm{TS}_{G',c'}$, where G' is obtained from G by deleting l edges corresponding to the path p (the labeling function also changes after the application of ρ, see Remark 2.2 for details). All paths from P that are consistent with ρ satisfy the formula $\mathrm{TS}_{G',c'}$. Recall that all paths from P correspond to different satisfying assignments, hence the number of paths that are consistent with ρ is not greater than the number of satisfying assignments of the formula $\mathrm{TS}_{G',c'}$. By Lemma 2 the number of satisfying assignments of the formula $\mathrm{TS}_{G',c'}$ equals $2^{|E|-l-|V|+m}$, where m is the number of connected components in the graph G'. By the statement of the lemma $m \leq k$, therefore the number of satisfying assignments of $\mathrm{TS}_{G',c'}$ is not greater than $2^{|E|-l-|V|+k}$. So we get that every partial substitution of l variables may be a prefix of length l (we assume that prefixes end in labeled nodes) of at most $2^{|E|-l-|V|+k}$ paths from P. Hence there are at least $\frac{2^{|E|-|V|+1}}{2^{|E|-l-|V|+k}} = 2^{l-(k-1)}$ different partial substitutions that correspond to prefixes of length l of paths from P, and these prefixes we will consider as the paths which number we estimate in the lemma. ◀

▶ **Lemma 11.** *Every graph that can be obtained by deleting $l \leq \frac{n}{4}$ edges from an algebraic (n, d, α)-expander G contains at most $\frac{2l}{d(1-\alpha)} + 1$ connected components.*

Proof.

▶ **Claim 12.** Let graph $H(V, E)$ have n vertices and k connected components, where $1 < k \leq \frac{n}{4} + 1$. Then there exists $M \subseteq V$ such that M consists of the union of all vertices of several connected components and $k - 1 \leq |M| \leq \frac{n}{2}$.

Proof of Claim 12. We construct M iteratively. Assume that initially M is empty. Let us sort all connected components in the increasing order of their sizes: $s_1, s_2, \ldots, s_k$. We add connected components to M starting from the smallest one while the sum of the sizes of these components is less than $k - 1$. Let i be the number of connected components that we added to M. If $|M| \leq \frac{n}{2}$ then we are already done. Assume that $|M| > \frac{n}{2}$. Note that $|M \setminus s_i| < k - 1$ by the construction of M. Hence $|s_i| > \frac{n}{2} - (k - 1) \geq k - 1$ since $k - 1 \leq \frac{n}{4}$. If $|s_i| \leq \frac{n}{2}$ then the we can take $M = s_i$. So we may assume that $|s_i| > n/2$, therefore s_i is the biggest connected component and $i = k$. Since every connected component contains at least one vertex the number of vertices in $M \setminus s_i$ should be at least $k - 1$ that contradicts the construction of M. ◀

Consider some subgraph H that may be obtained from G by deleting of at most l edges. By Corollary 6 G is connected, hence H contains at most $\frac{n}{4} + 1$ connected components. Consider the set M from Claim 12. Let us estimate the number of edges that we need to delete from G in order to separate M from other vertices of the graph. By Lemma 5 $l \geq \frac{|M| \cdot d \cdot (1-\alpha)}{2} \geq \frac{(k-1) \cdot d \cdot (1-\alpha)}{2}$. Hence $k - 1 \leq \frac{2l}{d \cdot (1-\alpha)}$. ◀

Altogether Lemma 10 and Lemma 11 imply the following lemma:

▶ **Lemma 13.** *In every minimal* 1-NBP *that represents a satisfiable Tseitin formula based on an* (n, d, α)*-expander for every* $l \leq \frac{n}{4}$ *there are at least* $2^{l\left(1-\frac{2}{d(1-\alpha)}\right)}$ *paths of length* l *from the source to a labeled node that correspond to different partial substitutions.*

3.2 Upper bound on the number of paths that end at the same vertex

In this section we estimate the maximum number of paths with length l that ends in a fixed labeled node v and correspond to different partial substitutions. In particular we prove the following lemma:

▶ **Lemma 14.** *For every minimal* 1-NBP *that evaluates a satisfiable Tseitin formula* $\mathrm{TS}_{G,c}$ *based on an* (n, d, α)*-expander* G *for every* $\beta \in (0; 1)$ *for every labeled node* v *of the* 1-NBP *there are at most* $2^{l\left(1-\frac{1}{d(\alpha+\beta)}\right)}$ *different partial substitutions that correspond to paths of length* l *from the source to* v*, where* $l \leq \beta n - 1$*.*

Proof.

▶ **Claim 15.** Consider some labeled node v of the 1-NBP. Let p_1 and p_2 be two different paths from the source to the node v. Then

1. the sets of variables that correspond to labeled nodes on the paths p_1 and p_2 are equal;
2. if we apply to $\mathrm{TS}_{G,c}$ a partial substitution corresponding to p_1, we get the same Tseitin formula as if we apply to $\mathrm{TS}_{G,c}$ a partial substitution corresponding to p_2.

Proof of Claim 15. 1. Since v is a labeled node and the 1-NBP is minimal there is a path s from v to the sink labeled with 1. Both paths $p_1 s$ and $p_2 s$ go from the source to the sink labeled with 1. Every variable appears in both of these paths at most once. Let x be a variable that appears in p_1 but doesn't appear in p_2 then the substitution corresponding to the path $p_2 s$ satisfy $\mathrm{TS}_{G,c}$. By Proposition 4 p_2 should contain the variable x.

2. By Remark 2.2 if we apply a partial substitution to a Tseitin formula we also get a Tseitin formula. The sets of satisfying assignments of two different Tseitin formulas do not intersect, because every satisfying assignment of variables unambiguously determines the labeling function of Tseitin formula. Paths $p_1 s$ and $p_2 s$ satisfy the initial formula hence the path s should satisfy both Tseitin formulas, the one corresponding to the path p_1 and the one corresponding to the path p_2. Hence these two Tseitin formulas should be equal. ◀

Let v be some labeled node that has level l. By Claim 15 every path from the source to the node v contains the same set of variables and if we apply to $\mathrm{TS}_{G,c}$ any of the substitutions corresponding to these paths we get the same Tseitin formula $\mathrm{TS}_{H,c'}$. Consider some path from the source to v of length l and denote the set of labels (i.e. variables) of the first l labeled nodes on this path by I. By Claim 15 I does not depend on the choice of the path. Let F be a set of edges that correspond to variables from I. Then $I = \{x_e \mid e \in F\}$ and H is obtained from G by deleting of all edges from F.

We define a system of linear equations depending on variables from I. This system states that the substitution to variables from I change labeling function from c to c' as follows:

$$\bigwedge_{u \in V} \left(\sum_{\substack{e \in F: \\ e \text{ is incident to } u}} x_e = c(u) + c'(u) \bmod 2 \right) \tag{1}$$

For every path from the source to v a partial substitution corresponding to this path is a solution of the system (1). The opposite is not always true since that it is not necessary that a path corresponding to the solution of the system (1) exists in the branching program.

▶ **Claim 16.** The number of solutions of the system (1) is equal to 2^{l-t}, where t is the number of the edges in the spanning forest of a graph $H(V, F)$ that is obtained from G by the deletion of all edges that are not in F.

Proof. Notice that the system (1) is precisely the Tseitin formula $\mathrm{TS}_{H,c+c'}$ based on the graph H and labelling function $c + c'$. We know that the system (1) has solutions, hence the number of its solutions by Lemma 2 equals $2^{|F|-|V|+k}$, where k is the number of connected components in H. The claim is proved since $|F| = l$ and $t = |V| - k$. ◀

▶ **Corollary 17.** *The number of different partial substitutions that correspond to paths going from the source to v is at most 2^{l-t}.*

▶ **Claim 18.** Let G be an algebraic (n, d, α)-expander. Assume that we deleted all edges from the graph except l edges, where $l = \beta n - 1$ and $0 < \beta < 1$. Then the number of edges in the spanning forest of the resulting graph H is at most $\frac{l}{d\cdot(\alpha+\beta)}$.

Proof. Consider any connected component $C \subseteq V$ in the resulting graph H. Let m be the number of edges and t be the number of vertices in C. We estimate the maximal number of edges that connect two vertices from C in the original graph G.

Since G is an algebraic (n, d, α)-expander by Proposition 8 there are at least $dt(1 - \frac{t}{n} - \alpha)$ edges connecting vertices from C with vertices from $V \setminus C$ in the graph G. Hence there are at most $\frac{dt - dt + \frac{dt^2}{n} + \alpha\cdot dt}{2} = \frac{\frac{dt^2}{n} + \alpha\cdot dt}{2}$ edges in G that connect two vertices from C.

Let us note that $t \le m + 1 \le l + 1 \le \beta n$, hence $m \le \frac{dt\cdot(\alpha+\beta)}{2}$. The latter implies that

$$t \ge \frac{2m}{d \cdot (\alpha + \beta)}. \tag{2}$$

Let t_i and m_i be the numbers of vertices and edges in the i-th connected component respectively. Note that the size of the spanning forest in H equals $\sum\limits_i (t_i - 1) = \sum\limits_{i:t_i\ge 2} (t_i - 1) \ge \sum\limits_{i:t_i\ge 2} \frac{t_i}{2}$. Note that all edges of H are in the components of size at least two.

By the inequality (2) we get $\sum\limits_{i:t_i\ge 2} t_i \ge \sum\limits_{i:t_i\ge 2} \frac{2m_i}{d\cdot(\alpha+\beta)} = \frac{2l}{d\cdot(\alpha+\beta)}$. Hence the resulting size of the spanning forest is at least $\frac{l}{d\cdot(\alpha+\beta)}$. ◀

Lemma 14 follows from Corollary 17 and Claim 18. ◀

3.3 Proof of Theorem 9

Proof of Theorem 9. Let $\beta = \min\{\frac{1}{4}, \frac{1-3\alpha}{3}\}$ and $l = \beta n - 1$. Consider the minimal 1-NBP for the Tseitin formula $\mathrm{TS}_{G,c}$.

By Lemma 13 there exist at least $2^{l\left(1-\frac{2}{d(1-\alpha)}\right)}$ paths of length l from the source that end in a labeled node that correspond to different partial substitutions. By Lemma 14 for every labeled node v on the level l there are at most $2^{l\left(1-\frac{1}{d(\alpha+\beta)}\right)}$ different partial substitutions that correspond to paths from the source to v.

Hence there are at least $2^{\frac{l}{d}\left(\frac{1}{\alpha+\beta}-\frac{2}{1-\alpha}\right)}$ labeled nodes on the distance l from the source. The latter is $2^{\Omega(n)}$ since $\beta < \frac{1-3\alpha}{2}$. ◀

3.4 Tseitin formula for complete graph

▶ **Corollary 19.** *Let* $\mathrm{TS}_{K_n,c}$ *be a satisfiable Tseitin formula, where* K_n *is a complete graph on* n *vertices. Then the size of every* 1-NBP *for* $\mathrm{TS}_{K_n,c}$ *is* $2^{\Omega(n)}$.

Proof. Consider a 1-NBP D that evaluates the formula $\mathrm{TS}_{K_n,c}$. Consider a graph G on n vertices that is an algebraic (n,d,α)-expander with $\alpha < \frac{1}{3}$ (note that G may have multiple edges). Consider a partial substitution ρ that assigns 0 for every edge that is in K_n but is not in G. Let D' be a 1-NBP that represents the result of the application of ρ to D. It is straightforward that the size of D' is at most the size of D. D' evaluates satisfiable Tseitin formula $\mathrm{TS}_{G',c}$, where G' is a graph that differs from G only by the fact that G may contain multiple edges (G' does not contain multiple edges). I.e., between every two vertices in the graph G' there is an edge if and only if there is at least one edge between these two vertices in G. Now we show how to obtain the diagram for $\mathrm{TS}_{G,c}$ from the diagram D'. Let graph G contain k edges between vertices u and v: $e_1, e_2, \ldots, e_k$. Note that $k \leq d$. It is well known that there exists a read-once deterministic branching program that evaluates $x_{e_1} + x_{e_2} + \cdots + x_{e_k}$ of size $k+2$. Let us denote this branching program by R. We put the source of R in the nodes labeled with variable $x_{u,v}$; the sink labeled with 0 in R should be identified with the end of the edge that correspond to the decision $x_{u,v} = 0$. And similarly we do with the sink labeled with 1. We do such substitutions for every pair of vertices that has multiple edges. The resulting program will be read-once because the original diagram was read-once. The size of the resulting program is at most d times greater than the size of the original branching program. By Theorem 9 the size of the resulting program is $2^{\Omega(n)}$ hence the size of D is $2^{\Omega(n)}$. ◀

3.5 Lower bound for arbitrary graphs

Let for connected graph $G(V,E)$ the value $k_G(l)$ denote the maximal number of connected components that can be obtained from G by deleting of l edges.

Lemma 10 and Corollary 17 imply:

▶ **Corollary 20.** *For all connected graphs* $G(V,E)$ *and arbitrary* $1 \leq l \leq |E|$ *the size of any* 1-NBP *evaluating a satisfiable Tseitin formula* $\mathrm{TS}_{G,c}$ *is at least* $2^{|V|-k_G(l)-k_G(|E|-l)+1}$.

Proof. Consider the minimal 1-NBP for the Tseitin formula $\mathrm{TS}_{G,c}$.

By Lemma 10 there exist at least $2^{l-k_G(l)+1}$ paths of length l from the source that end in a labeled node that correspond to different partial substitutions. By Corollary 17 for every labeled node v on the level l there are at most $2^{l-|V|+k_G(|E|-l)}$ different partial substitutions that correspond to paths from the source to v.

Hence there are at least $2^{|V|-k_G(l)-k_G(|E|-l)+1}$ labeled nodes on the distance l from the source. ◀

In the proof of Theorem 9 we actually show that for (n,d,α)-expander with $\alpha < \frac{1}{3}$ $k_G(l) - k_G(|E|-l) < (1-\epsilon)n$ for some l and some constant $\epsilon > 0$. It implies that Theorem 9 also holds for graphs that differ from (n,d,α) expander by at most $\epsilon n/4$ edges since modification of $\epsilon n/4$ edges changes $k_G(l) + k_G(|E|-l)$ by at most $\epsilon n/2$.

It was proved in the paper [9] that for all connected graphs $G(V,E)$ and arbitrary $1 \leq l \leq |E|$ the size of OBDD evaluating a satisfiable Tseitin formula $\mathrm{TS}_{G,c}$ is at least $2^{|V|-k'_G(l)}$, where $k'_G(l)$ is the maximum over all sets $E' \subseteq E$ of size l of the total number of connected components in graphs G' and G'', where G' is a graph with vertices V and edges E', G'' is a graph with vertices V and edges $E \setminus E'$. It is straightforward that

$k_G(l) + k_G(|E| - l) \geq k'_G(l)$. Thus theoretically the lower bound on the size of OBDD from [9] may be slightly stronger then the lower bound from Corollary 20 for some specific graphs.

4 Futher research

Jukna [10] defined the notion of semantic nondeterministic read-k branching programs that have weaker requirement about occurrences of variables. Namely on every consistent path from the source to a sink labeled with 1 every variable should be tested in at most k times. Jukna showed that semantic nondeterministic read-once branching programs are strictly stronger than syntactic ones and formulated an open question to prove superpolynomial lower bound on the size of semantic 1-NBP. Currently such lower bounds are known only for explicit functions from $D^n \to \{0, 1\}$ with non-binary domains D of size at least 3 [6, 11]. Perhaps a satisfiable Tseitin formula is a good candidate for the binary case.

Acknowledgements. The authors are grateful to Mikhail Slabodkin and reviewers for useful comments and to Alexander Knop and Dmitry Sokolov for fruitful discussions.

References

1 Noga Alon and Fan R. K. Chung. Explicit construction of linear sized tolerant networks. *Discrete Mathematics*, 306(10-11):1068–1071, 2006. doi:10.1016/j.disc.2006.03.025.

2 L. Babai, P. Hajnal, E. Szemeredi, and G. Turan. A lower bound for read-once-only branching programs. *Journal of Computer and System Sciences*, 35:153–162, 1987.

3 Eli Ben-Sasson. Hard examples for bounded depth Frege. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 563–572, 2002. doi:10.1145/509907.509988.

4 Allan Borodin, Alexander A. Razborov, and Roman Smolensky. On lower bounds for read-k-times branching programs. *Computational Complexity*, 3:1–18, 1993. doi:10.1007/BF01200404.

5 J Cheeger. A lower bound for the smallest eigenvalue of the laplacian. *Problems Anal.*, page 195, 1970.

6 Stephen A. Cook, Jeff Edmonds, Venkatesh Medabalimi, and Toniann Pitassi. Lower bounds for nondeterministic semantic read-once branching programs. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 36:1–36:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.36.

7 Pavol Duris, Juraj Hromkovic, Stasys Jukna, Martin Sauerhoff, and Georg Schnitger. On multi-partition communication complexity. *Inf. Comput.*, 194(1):49–75, 2004. doi:10.1016/j.ic.2004.05.002.

8 D.M. Itsykson and A.A. Kojevnikov. Lower bounds of static Lovasz-Schrijver calculus proofs for Tseitin tautologies. *Zapiski Nauchnykh Seminarov POMI*, 340:10–32, 2006.

9 Dmitry Itsykson, Alexander Knop, Andrei Romashchenko, and Dmitry Sokolov. On obdd-based algorithms and proof systems that dynamically change order of variables. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 43:1–43:14, 2017. doi:10.4230/LIPIcs.STACS.2017.43.

10 Stasys Jukna. A note on read-k times branching programs. *ITA*, 29(1):75–83, 1995.

11 Stasys Jukna. A nondeterministic space-time tradeoff for linear codes. *Inf. Process. Lett.*, 109(5):286–289, 2009. doi:10.1016/j.ipl.2008.11.001.

12 László Lovász, Moni Naor, Ilan Newman, and Avi Wigderson. Search Problems in the Decision Tree Model. *SIAM J. Discrete Math.*, 8(1):119–132, 1995. doi:10.1137/S0895480192233867.

13 A Lubotzky, R Phillips, and P Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

14 EA Okolnishnikova. On lower bounds for branching programs. *Siberian Advances in Mathematics*, 3(1):152–166, 1993.

15 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, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 644–657, 2016. doi:10.1145/2897518.2897637.

16 Jayram S. Thathachar. On separating the read-k-times branching program hierarchy. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 653–662, 1998. doi:10.1145/276698.276881.

17 A. Urquhart. Hard examples for resolution. *JACM*, 34(1):209–219, 1987.

18 Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.

The Complexity of Quantified Constraints Using the Algebraic Formulation

Catarina Carvalho[1], Barnaby Martin[2], and Dmitriy Zhuk[3]

1 **School of Physics, Astronomy and Mathematics, University of Hertfordshire, Hatfield, UK**
2 **School of Engineering and Computing Sciences, Durham University, UK**
3 **Moscow State University, Moscow, Russia**

Abstract

Let $\mathbb{A}$ be an idempotent algebra on a finite domain. We combine results of Chen [7], Zhuk [20] and Carvalho et al. [5] to argue that if $\mathbb{A}$ satisfies the polynomially generated powers property (PGP), then QCSP(Inv($\mathbb{A}$)) is in NP. We then use the result of Zhuk to prove a converse, that if Inv($\mathbb{A}$) satisfies the exponentially generated powers property (EGP), then QCSP(Inv($\mathbb{A}$)) is co-NP-hard. Since Zhuk proved that only PGP and EGP are possible, we derive a full dichotomy for the QCSP, justifying the moral correctness of what we term the Chen Conjecture (see [8]).

We examine in closer detail the situation for domains of size three. Over any finite domain, the only type of PGP that can occur is switchability. Switchability was introduced by Chen in [7] as a generalisation of the already-known Collapsibility [6]. For three-element domain algebras $\mathbb{A}$ that are Switchable, we prove that for every finite subset Δ of Inv($\mathbb{A}$), Pol(Δ) is Collapsible. The significance of this is that, for QCSP on finite structures (over three-element domain), all QCSP tractability explained by Switchability is already explained by Collapsibility.

Finally, we present a three-element domain complexity classification vignette, using known as well as derived results.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Quantified Constraints, Computational Complexity, Universal Algebra, Constraint Satisfaction

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.27

1 Introduction

A large body of work exists from the past twenty years on applications of universal algebra to the computational complexity of *constraint satisfaction problems* (CSPs) and a number of celebrated results have been obtained through this method. One considers the problem CSP($\mathcal{B}$) in which it is asked whether an input sentence φ holds on $\mathcal{B}$, where φ is *primitive positive*, that is using only $\exists$, $\wedge$ and $=$. The CSP is one of a wide class of model-checking problems obtained from restrictions of first-order logic. For almost every one of these classes, we can give a complexity classification [14]: the two outstanding classes are CSPs and its popular extension *quantified CSPs* (QCSPs) for positive Horn sentences – where $\forall$ is also present – which is used in Artificial Intelligence to model non-monotone reasoning or uncertainty [11].

The outstanding conjecture in the area is that all finite-domain CSPs are either in P or are NP-complete, something surprising given these CSPs appear to form a large microcosm of NP, and NP itself is unlikely to have this dichotomy property. This Feder-Vardi conjecture [12], given more concretely in the algebraic language in [4], remains unsettled, but is now

known for large classes of structures. It is well-known that the complexity classification for QCSPs embeds the classification for CSPs: if $\mathcal{B}+1$ is $\mathcal{B}$ with the addition of a new isolated element not appearing in any relations, then $\mathrm{CSP}(\mathcal{B})$ and $\mathrm{QCSP}(\mathcal{B}+1)$ are polynomially equivalent. Thus the classification for QCSPs may be considered a project at least as hard as that for CSPs. The following is the merger of Conjectures 6 and 7 in [8] which we call the *Chen Conjecture*.

▶ **Conjecture 1** (Chen Conjecture). *Let $\mathcal{B}$ be a finite relational structure expanded with all constants. If Pol($\mathcal{B}$) has PGP, then QCSP($\mathcal{B}$) is in NP; otherwise QCSP($\mathcal{B}$) is Pspace-complete.*

In [8], Conjecture 6 gives the NP membership and Conjecture 7 the Pspace-completeness. We now know from [20] and [5] that the NP membership of Conjecture 6 is indeed true. The most interesting result of this paper is Theorem 2 below, but note that we permit infinite signatures (languages) although our domains remain finite. This aspect of our work will be discussed in detail later.

▶ **Theorem 2** (Revised Chen Conjecture). *Let $\mathbb{A}$ be an idempotent algebra on a finite domain A. If $\mathbb{A}$ satisfies PGP, then QCSP($\mathrm{Inv}(\mathbb{A})$) is in NP. Otherwise, QCSP($\mathrm{Inv}(\mathbb{A})$) is co-NP-hard.*

Zhuk has previously proved [20] that only the cases PGP and EGP may occur, even in the non-idempotent case. With infinite languages, the NP-membership for Theorem 2 is no longer immediate from [5], but requires a little extra work. We are also able to refute the following form.

▶ **Conjecture 3** (Alternative Chen Conjecture). *Let $\mathbb{A}$ be an idempotent algebra on a finite domain A. If $\mathbb{A}$ satisfies PGP, then for every finite subset $\Delta \subseteq \mathrm{Inv}(\mathbb{A})$, QCSP($\Delta$) is in NP. Otherwise, there exists a finite subset $\Delta \subseteq \mathrm{Inv}(\mathbb{A})$ so that QCSP(Δ) is co-NP-hard.*

In proving Theorem 2 we are saying that the complexity of QCSPs, with all constants included, is classified modulo the complexity of CSPs.

▶ **Corollary 4.** *Let $\mathbb{A}$ be an idempotent algebra on a finite domain A. Either QCSP($\mathrm{Inv}(\mathbb{A})$) is co-NP-hard or QCSP($\mathrm{Inv}(\mathbb{A})$) has the same complexity as CSP($\mathrm{Inv}(\mathbb{A})$).*

In this manner, our result follows in the footsteps of the similar result for the Valued CSP, which has also had its complexity classified modulo the CSP, as culminated in the paper [13].

For a finite-domain algebra $\mathbb{A}$ we associate a function $f_{\mathbb{A}} : \mathbb{N} \to \mathbb{N}$, giving the cardinality of the minimal generating sets of the sequence $\mathbb{A}, \mathbb{A}^2, \mathbb{A}^3, \ldots$ as $f_{\mathbb{A}}(1), f_{\mathbb{A}}(2), f_{\mathbb{A}}(3), \ldots$, respectively. A subset Λ of A^m is a generating set for $\mathbb{A}^m$ exactly if, for every $(a_1, \ldots, a_m) \in A^m$, there exists a k-ary term operation f of $\mathbb{A}$ and $(b_1^1, \ldots, b_m^1), \ldots, (b_1^k, \ldots, b_m^k) \in \Lambda$ so that $f(b_1^1, \ldots, b_1^k) = a_1, \ldots, f(b_m^1, \ldots, b_m^k) = a_m$. We may say $\mathbb{A}$ has the g-GP if $f_{\mathbb{A}}(m) \leq g(m)$ for all m. The question then arises as to the growth rate of $f_{\mathbb{A}}$ and specifically regarding the behaviours constant, logarithmic, linear, polynomial and exponential. Wiegold proved in [19] that if $\mathbb{A}$ is a finite semigroup then $f_{\mathbb{A}}$ is either linear or exponential, with the former prevailing precisely when $\mathbb{A}$ is a monoid. This dichotomy classification may be seen as a gap theorem because no growth rates intermediate between linear and exponential may occur. We say $\mathbb{A}$ enjoys the *polynomially generated powers* property (PGP) if there exists a polynomial p so that $f_{\mathbb{A}} = O(p)$ and the *exponentially generated powers* property (EGP) if there exists a constant b so that $f_{\mathbb{A}} = \Omega(g)$ where $g(i) = b^i$.

In Hubie Chen's [7], a new link between algebra and QCSP was discovered. Chen's previous work in QCSP tractability largely involved the special notion of *Collapsibility*

[6], but in [7] this was extended to a *computationally effective* version of the PGP. For a finite-domain, idempotent algebra $\mathbb{A}$, *k-collapsibility* may be seen as that special form of the PGP in which the generating set for $\mathbb{A}^m$ is constituted of all tuples $(x_1, \ldots, x_m)$ in which at least $m - k$ of these elements are equal. *k-switchability* may be seen as another special form of the PGP in which the generating set for $\mathbb{A}^m$ is constituted of all tuples $(x_1, \ldots, x_m)$ in which there exists $a_i < \ldots < a_{k'}$, for $k' \leq k$, so that

$$(x_1, \ldots, x_m) = (x_1, \ldots, x_{a_1}, x_{a_1+1}, \ldots, x_{a_2}, x_{a_2+1}, \ldots, \ldots, x_{a'_k}, x_{a'_k+1}, \ldots, x_m),$$

where $x_1 = \ldots = x_{a_1-1}$, $x_{a_1} = \ldots = x_{a_2-1}$, $\ldots$, $x_{a_{k'}} = \ldots = x_{a_m}$. Thus, $a_1, a_2, \ldots, a_{k'}$ are the indices where the tuple switches value. Note that these are not the original definitions, which we will see shortly, but they are proved equivalent to the original definitions (at least for finite signatures) in [5]. Moreover, these are the definitions that we will use. We say that $\mathbb{A}$ is collapsible (switchable) if there exists k such that it is k-collapsible (k-switchable). We note that Zhuk uses this definition of switchability in [20] in which he proved that the only kind of PGP for finite-domain algebras is switchability.

Let us capitalise Collapsibility and Switchability to indicate Chen's original definitions from [7] are used, following an example for *arithmetic* versus *Arithmetic* by Raymond Smullyan in [18]. There is the potential for confusion at the start of the sentence but, as was the case with Smullyan, the two will transpire to be interchangeable throughout our discourse. It is straightforward to see that k-Switchability implies k-switchability and k-Collapsibility implies k-collapsibility. The converses, for finite signatures, also hold, but this requires rather more work [5]. For any finite algebra, k-Collapsibility implies k-Switchability, and for any 2-element algebra, k-Switchability implies k-Collapsibility. Chen originally introduced Switchability because he found a 3-element algebra that enjoyed the PGP but was not Collapsible [7]. He went on to prove that Switchability of $\mathbb{A}$ implies that the corresponding QCSP is in P, what one might informally state as QCSP(Inv($\mathbb{A}$)) in P, where Inv($\mathbb{A}$) can be seen as the structure over the same domain as $\mathbb{A}$ whose relations are precisely those that are preserved by (invariant under) all the operations of $\mathbb{A}$. However, the QCSP was traditionally defined only on finite sets of relations (else the question arises as to encoding), thus a more formal definition might be that, for any finite subset Δ of Inv($\mathbb{A}$), QCSP(Δ) is in P. What we prove in this paper is that, as far as the QCSP is concerned, Switchability on a three-element algebra $\mathbb{A}$ is something of a mirage. What we mean by this is that when $\mathbb{A}$ is Switchable, for all finite subsets Δ of Inv($\mathbb{A}$), already Pol(Δ) is Collapsible. Thus, for QCSP complexity for three-element structures, we do not need the additional notion of Switchability to explain tractability, as Collapsibility will already suffice. Since these notions were originally introduced in connection with the QCSP this is particularly surprising. Note that the parameter k of Collapsibility is unbounded over these increasing finite subsets Δ while the parameter of Switchability clearly remains bounded. In some way we are suggesting that Switchability itself might be seen as a limit phenomenon of Collapsibility.

1.1 Infinite languages

Our use of infinite languages (i.e. signatures, since we work on a finite domain) is the only controversial part of our discourse and merits special discussion. We wish to argue that a necessary corollary of the algebraic approach to (Q)CSP is a reconciliation with infinite languages. The traditional approach to consider arbitrary finite subsets of Inv($\mathbb{A}$) is unsatisfactory in the sense that choosing this way to escape the – naturally infinite – set Inv($\mathbb{A}$) is as arbitrary as the choice of encoding required for infinite languages. However, the difficulty in that choice is of course the reason why this route is often eschewed. The

first possibility that comes to mind for encoding a relation in Inv($\mathbb{A}$) is probably to list its tuples, while the second is likely to be to describe the relation in some kind of "simple" logic. Both these possibilities are discussed in [10], for the Boolean domain, where the "simple" logic is the propositional calculus. For larger domains, this would be equivalent to quantifier-free propositions over equality with constants. Both Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF) representations are considered in [10] and a similar discussion in [2] exposes the advantages of the DNF encoding. The point here is that testing non-emptiness of a relation encoded in CNF may already be NP-hard, while for DNF this will be tractable. Since DNF has some benign properties, we might consider it a "nice, simple" logic while for "simple" logic we encompass all quantifier-free sentences, that include DNF and CNF as special cases. The reason we describe this as "simple" logic is to compare against something stronger, say all first-order sentences over equality with constants. Here recognising non-emptiness becomes Pspace-hard and since QCSPs already sit in Pspace, this complexity is unreasonable.

For the QCSP over infinite languages Inv($\mathbb{A}$), Chen and Mayr [9] have declared for our first, tuple-listing, encoding. In this paper we will choose the "simple" logic encoding, occasionally giving more refined results for its "nice, simple" restriction to DNF. Our choice of the "simple" logic encoding over the tuple-listing encoding will ultimately be justified by the (Revised) Chen Conjecture holding for "simple" logic yet failing for tuple-listings. Note that our demonstration of the (Revised) Chen Conjecture for infinite languages with the "simple" logic encoding does not resolve the original Chen Conjecture for finite languages $\mathcal{B}$ with constants because QCSP(Inv(Pol($\mathcal{B}$))) could conceivably have higher complexity than QCSP($\mathcal{B}$) due to a succinct representation of relations in Inv(Pol($\mathcal{B}$)). Indeed, this belies one justification for the preferential study of finite subsets of Inv(Pol($\mathcal{B}$)), since for finite signature $\mathcal{B}$ we can then say QCSP($\mathcal{B}$) and QCSP(Inv(Pol$\mathcal{B}$)) must have the same complexity. Note that for finite relational bases $\mathcal{B}', \mathcal{B}''$ of Inv(Pol($\mathcal{B}$)), QCSP($\mathcal{B}'$) and QCSP($\mathcal{B}''$) must have the same complexity. Further, we do not know of any concrete finite $\mathcal{B}$ with constants, so that QCSP(Inv(Pol($\mathcal{B}$))) and QCSP($\mathcal{B}$) have different complexity.

Let us consider examples of our encodings. For the domain $\{1, 2, 3\}$, we may give a binary relation either by the tuples $\{(1, 2), (2, 1), (2, 3), (3, 2), (1, 3), (3, 1), (1, 1)\}$ or by the "simple" logic formula $(x \neq y \vee x = 1)$. For the domain $\{0, 1\}$, we may give the ternary (not-all-equal) relation by the tuples $\{(1, 0, 0), (0, 1, 0), (0, 0, 1), (1, 1, 0), (1, 0, 1), (1, 1, 0)\}$ or by the "simple" logic formula $(x \neq y \vee y \neq z)$. In both of these examples, the simple formula is also in DNF.

Nota Bene. The results of this paper apply for the "simple" logic encoding as well as the "nice, simple" encoding in DNF except where specifically stated otherwise. These exceptions are Proposition 13 and Corollary 14 (which uses the "nice, simple" DNF) and Proposition 16 (which uses the tuple encoding).

Related work. This paper is the merger of [16, 15], neither of which was submitted for publication, considerably extended.

2 Preliminaries

Let $[k] := \{1, \ldots, k\}$. A *k-ary polymorphism* of a relational structure $\mathcal{B}$ is a homomorphism f from $\mathcal{B}^k$ to $\mathcal{B}$. Let Pol($\mathcal{B}$) be the set of polymorphisms of $\mathcal{B}$ and let Inv($\mathbb{A}$) be the set of relations on A which are invariant under (each of) the operations of some finite algebra $\mathbb{A}$. Pol($\mathcal{B}$) is an object known in Universal Algebra as a *clone*, which is a set of operations

containing all projections and closed under composition (superposition). A *term operation* of an algebra $\mathbb{A}$ is an operation which is a member of the clone generated by $\mathbb{A}$.

We will conflate sets of operations over the same domain and algebras just as we do sets of relations over the same domain and constraint languages (relational structures). Indeed, the only technical difference between such objects is the movement away from an ordered signature, which is not something we will ever need. A *reduct* of a relational structure $\mathcal{B}$ is a relational structure $\mathcal{B}'$ over the same domain obtained by forgetting some of the relations. If Δ is some finite subset of $\mathrm{Inv}(\mathbb{A})$, then we may view Δ a being a finite reduct of the structure (associated with) $\mathrm{Inv}(\mathbb{A})$.

A k-ary operation f over A is a *projection* if $f(x_1, \ldots, x_k) = x_i$, for some $i \in [k]$. When α, β are strict subsets of A so that $\alpha \cup \beta = A$, then a k-ary operation f on A is said to be *$\alpha\beta$-projective* if there exists $i \in [k]$ so that if $x_i \in \alpha$ (respectively, $x_i \in \beta$), then $f(x_1, \ldots, x_k) \in \alpha$ (respectively, $f(x_1, \ldots, x_k) \in \beta$).

We recall $\mathrm{QCSP}(\mathcal{B})$, where $\mathcal{B}$ is some structure on a finite-domain, is a decision problem with input ϕ, a pH-sentence (i.e. using just $\forall$, $\exists$, $\wedge$ and $=$) involving (a finite set of) relations of $\mathcal{B}$, encoded in propositional logic with equality and constants. The yes-instances are those ϕ for which $\mathcal{B} \models \phi$. If the input sentence is restricted to have alternation Π_k then the corresponding problem is designated Π_k-$\mathrm{CSP}(\mathcal{B})$.

2.1 Games, adversaries and reactive composition

We now recall some terminology due to Chen [6, 7], for his natural adaptation of the model checking game to the context of pH-sentences. We shall not need to explicitly play these games but only to handle strategies for the existential player. This will enable us to give the original definitions for Collapsibility and Switchability. An *adversary* $\mathcal{B}$ of length $m \geq 1$ is an m-ary relation over A. When $\mathcal{B}$ is precisely the set $B_1 \times B_2 \times \ldots \times B_m$ for some non-empty subsets $B_1, B_2, \ldots, B_m$ of A, we speak of a *rectangular adversary* (we will sometimes specify this as a tuple rather than a product). Let ϕ be a pH-sentence with universal variables $x_1, \ldots, x_m$ and quantifier-free part ψ. We write $\mathcal{A} \models \phi_{\restriction \mathcal{B}}$ and say that *the existential player has a winning strategy in the $(\mathcal{A}, \phi)$-game against adversary $\mathcal{B}$* iff there exists a set of Skolem functions $\{\sigma_x : \text{`}\exists x\text{'} \in \phi\}$ such that for any assignment π of the universally quantified variables of ϕ to A, where $\big(\pi(x_1), \ldots, \pi(x_m)\big) \in \mathcal{B}$, the map h_π is a homomorphism from $\mathcal{D}_\psi$ (the canonical database) to $\mathcal{A}$, where

$$h_\pi(x) := \begin{cases} \pi(x) & \text{, if } x \text{ is a universal variable; and,} \\ \sigma_x(\pi|_{Y_x}) & \text{, otherwise.} \end{cases}$$

(Here, Y_x denotes the set of universal variables preceding x and $\pi|_{Y_x}$ the restriction of π to Y_x.) Clearly, $\mathcal{A} \models \phi$ iff the existential player has a winning strategy in the $(\mathcal{A}, \phi)$-game against the so-called *full (rectangular) adversary* $A \times A \times \ldots \times A$ (which we will denote hereafter by A^m). We say that an adversary $\mathcal{B}$ of length m *dominates* an adversary $\mathcal{B}'$ of length m when $\mathcal{B}' \subseteq \mathcal{B}$. Note that $\mathcal{B}' \subseteq \mathcal{B}$ and $\mathcal{A} \models \phi_{\restriction \mathcal{B}}$ implies $\mathcal{A} \models \phi_{\restriction \mathcal{B}'}$. We will also consider sets of adversaries of the same length, denoted by uppercase Greek letters as in Ω_m (here the length is m); and, sequences thereof, which we denote with bold uppercase Greek letters as in $\mathbf{\Omega} = \big(\Omega_m\big)_{m \in \mathbb{N}}$. We will write $\mathcal{A} \models \phi_{\restriction \Omega_m}$ to denote that $\mathcal{A} \models \phi_{\restriction \mathcal{B}}$ holds for every adversary $\mathcal{B}$ in Ω_m.

We now introduce reactive composition as a means to obtain larger adversaries from a number of smaller adversaries. Let f be a k-ary operation of $\mathcal{A}$ and $\mathcal{A}, \mathcal{B}_1, \ldots, \mathcal{B}_k$ be adversaries of length m. We say that $\mathcal{A}$ is *reactively composable* from the adversaries

$\mathcal{B}_1, \ldots, \mathcal{B}_k$ via f, and we write $\mathcal{A} \trianglelefteq f(\mathcal{B}_1, \ldots, \mathcal{B}_k)$ iff there exist partial functions $g_i^j : A^i \to A$ for every i in $[m]$ and every j in $[k]$ such that, for every tuple $(a_1, \ldots, a_m)$ in adversary $\mathcal{A}$ the following holds.

- for every j in $[k]$, the values $g_1^j(a_1), g_2^j(a_1, a_2), \ldots, g_m^j(a_1, a_2, \ldots, a_m)$ are defined and the tuple $\big(g_1^j(a_1), g_2^j(a_1, a_2), \ldots, g_m^j(a_1, a_2, \ldots, a_m)\big)$ is in adversary $\mathcal{B}_j$; and,
- for every i in $[m]$, $a_i = f\big(g_i^1(a_1, a_2, \ldots, a_i), g_i^2(a_1, a_2, \ldots, a_i), \ldots, g_i^k(a_1, a_2, \ldots, a_i)\big)$.

We write $\mathcal{A} \trianglelefteq \{\mathcal{B}_1, \ldots, \mathcal{B}_k\}$ if there exists a k-ary operation f such that $\mathcal{A} \trianglelefteq f(\mathcal{B}_1, \ldots, \mathcal{B}_k)$ Reactive composition allows to interpolate complete Skolem functions from partial ones.

▶ **Theorem 5** ([7, Theorem 7.6]). *Let ϕ be a pH-sentence with m universal variables. Let $\mathcal{A}$ be an adversary and Ω_m a set of adversaries, both of length m.*

If $\mathcal{A} \models \phi_{\restriction \Omega_m}$ and $\mathcal{A} \trianglelefteq \Omega_m$ then $\mathcal{A} \models \phi_{\restriction \mathcal{A}}$.

As a concrete example of an interesting sequence of adversaries, consider the adversaries for the notion of *p-Collapsibility*. Let $p \geq 0$ be some fixed integer. For x in A, let $\Upsilon_{m,p,x}$ be the set of all rectangular adversaries of length m with p co-ordinates that are the set A and all the others that are the fixed singleton $\{x\}$. For $B \subseteq A$, let $\Upsilon_{m,p,B}$ be the union of $\Upsilon_{m,p,x}$ for all x in B. Let $\boldsymbol{\Upsilon}_{p,B}$ be the sequence of adversaries $(\Upsilon_{m,p,B})_{m\in\mathbb{N}}$. We will define a structure $\mathcal{A}$ to be *p-Collapsible from source B* iff for every m and for all pH-sentence ϕ with m universal variables, $\mathcal{A} \models \phi_{\restriction \Upsilon_{m,p,B}}$ implies $\mathcal{A} \models \phi$.

For p-Switchability, the set of adversaries will be of the form $\Xi_{m,p}$, where each adversary is built from the set of tuples that have some $k' < p$ switches at specific points $0 < a_1 < \ldots < a_{k'} \leq m$.

For rectangular adversaries, such as $\Upsilon_{m,p,x}$, reactive composition is rather simpler than in the definition above, becoming just (ordinary) composition, as follows. $\mathcal{A}$ is *composable* from the adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_k$ via f if $f(B_1^i, \ldots, B_i^k) \supseteq A^i$, where $\mathcal{A} = (A^1, \ldots, A^m)$ and each $\mathcal{B}_j = (B_j^1, \ldots, B_j^m)$. Reactive composition plays a key role in the proof of our main theorem but its use appears only in other papers that we will cite. Ordinary composition is the only type of reactive composition that will be used in this paper.

3 The Chen Conjecture

3.1 NP-membership

We need to revisit the main result of [5] to show that it holds not just for finite signatures but for infinite signatures also. In its original the following theorem discussed "projective sequences of adversaries, none of which are degenerate". This includes Switching adversaries and we give it in this latter form. We furthermore remove some parts of the theorem that are not currently relevant to us.

▶ **Theorem 6** (In abstracto [5]). *Let $\boldsymbol{\Omega} = (\Omega_m)_{m\in\mathbb{N}}$ be the sequence of the set of all (k-)Switching m-ary adversaries over the domain of $\mathcal{A}$, a finite structure. The following are equivalent.*

(i) *For every $m \geq 1$, for every pH-sentence ψ with m universal variables, $\mathcal{A} \models \psi_{\restriction \Omega_m}$ implies $\mathcal{A} \models \psi$.*

(vi) *For every $m \geq 1$, Ω_m generates $\mathrm{Pol}(\mathcal{A})^m$.*

▶ **Corollary 7** (In abstracto levavi). *Let $\boldsymbol{\Omega} = (\Omega_m)_{m\in\mathbb{N}}$ be the sequence of the set of all (k-)Switching m-ary adversaries over the domain of $\mathcal{A}$, a finite-domain structure with an infinite signature. The following are equivalent.*

(i) *For every $m \geq 1$, for every pH-sentence ψ with m universal variables, $\mathcal{A} \models \psi_{\restriction \Omega_m}$ implies $\mathcal{A} \models \psi$.*

(vi) *For every $m \geq 1$, Ω_m generates* $\mathrm{Pol}(\mathcal{A})^m$.

Proof. We know from Theorem 6 that the following are equivalent:

(i') For every finite-signature reduct $\mathcal{A}'$ of $\mathcal{A}$ and $m \geq 1$, for every pH-sentence ψ with m universal variables, $\mathcal{A}' \models \psi_{\restriction \Omega_m}$ implies $\mathcal{A}' \models \psi$.

(vi') For every finite-signature reduct $\mathcal{A}'$ of $\mathcal{A}$ and every $m \geq 1$, Ω_m generates $\mathrm{Pol}(\mathcal{A}')^m$.

Since it is clear that both $(i) \Rightarrow (i')$ and $(vi) \Rightarrow (vi')$, it remains to argue that $(i') \Rightarrow (i)$ and $(vi') \Rightarrow (vi)$.

$[(i') \Rightarrow (i).]$ By contraposition, if (i) fails then it fails on some specific pH-sentence ψ which only mentions a finite number of relations of $\mathcal{A}'$. Thus (i') also fails on some finite reduct of $\mathcal{A}'$ mentioning these relations.

$[(vi') \Rightarrow (vi).]$ Let m be given. Consider some chain of finite reducts $\mathcal{A}_1, \ldots, \mathcal{A}_i, \ldots$ of $\mathcal{A}$ so that each $\mathcal{A}_i$ is a reduct of $\mathcal{A}_j$ for $i < j$ and every relation of $\mathcal{A}$ appears in some $\mathcal{A}_i$. We can assume from $(vi)'$ that Ω_m generates $\mathrm{Pol}(\mathcal{A}_i)^m$, for each i. However, since the number of tuples $(a_1, \ldots, a_m)$ and operations mapping Ω_m pointwise to $(a_1, \ldots, a_m)$, witnessing generation in $\mathrm{Pol}(\mathcal{A}')^m$, is finite, the sequence of operations $(f^i_1, \ldots, f^i_{|A|^m})$ (where f^i_j witnesses generation of the jth tuple in A^m) witnessing these must have an infinitely recurring element as i tends to infinity. One such recurring element we call $(f_1, \ldots, f_{|A|^m})$ and this witnesses generation in $\mathrm{Pol}(\mathcal{A})^m$. ◀

Note that in $(vi') \Rightarrow (vi)$ above we did not need to argue uniformly across the different $(a_1, \ldots, a_m)$ and it is enough to find an infinitely recurring operation for each of these individually.

The following result is essentially a corollary of the works of Chen and Zhuk [7, 20] via [5].

▶ **Theorem 8.** *Let $\mathbb{A}$ be an idempotent algebra on a finite domain A. If $\mathbb{A}$ satisfies PGP, then QCSP*$(\mathrm{Inv}(\mathbb{A}))$ *reduces to a polynomial number of instances of CSP*$(\mathrm{Inv}(\mathbb{A}))$ *and is in NP.*

Proof. We know from Theorem 7 in [20] that $\mathbb{A}$ is Switchable, whereupon we apply Corollary 7, $(vi) \Rightarrow (i)$. By considering instances whose universal variables involve only the polynomial number of tuples from the Switching Adversary, one can see that $\mathrm{QCSP}(\mathrm{Inv}(\mathbb{A}))$ reduces to a polynomial number of instances of $\mathrm{CSP}(\mathrm{Inv}(\mathbb{A}))$ and is therefore in NP. Further details of the NP algorithm are given in Corollary 38 of [5] but the argument here follows exactly Section 7 from [7], in which it was originally proved that Switchability yields the corresponding QCSP in NP. ◀

Note that Chen's original definition of Switchability, based on adversaries and reactive composability, plays a key role in the NP membership algorithm in Theorem 8. It is the result from [5] that is required to reconcile the two definitions of switchability as equivalent, and indeed Corollary 7 is needed in this process for infinite signatures. If we were to use just our definition of switchability then it is only possible to prove, à la Proposition 3.3 in [7], that the bounded alternation Π_n-$\mathrm{CSP}(\mathrm{Inv}(\mathbb{A}))$ is in NP. Thus, using just the methods from [7] and [20], we cannot prove the Revised Chen Conjecture, but rather some bounded alternation (re)revision.

3.2 co-NP-hardness

Suppose there exist α, β strict subsets of A so that $\alpha \cup \beta = A$, define the relation $\tau_k(x_1, y_1, z_1 \ldots, x_k, y_k, z_k)$ by

$$\tau_k(x_1, y_1, z_1 \ldots, x_k, y_k, z_k) := \rho'(x_1, y_1, z_1) \vee \ldots \vee \rho'(x_k, y_k, z_k),$$

where $\rho'(x, y, z) = (\alpha \times \alpha \times \alpha) \cup (\beta \times \beta \times \beta)$. Strictly speaking, the α and β are parameters of τ_k but we dispense with adding them to the notation since they will be fixed at any point in which we invoke the τ_k. The purpose of the relations τ_k is to encode co-NP-hardness through the complement of the problem (monotone) 3-*not-all-equal-satisfiability* (3NAESAT). Let us introduce also the important relations $\sigma_k(x_1, y_1, \ldots, x_k, y_k)$ defined by

$$\sigma_k(x_1, y_1, \ldots, x_k, y_k) := \rho(x_1, y_1) \vee \ldots \vee \rho(x_k, y_k),$$

where $\rho(x, y) = (\alpha \times \alpha) \cup (\beta \times \beta)$.

▶ **Lemma 9.** *The relation τ_k is pp-definable in σ_k.*

Proof. We will argue that τ_k is definable by the conjunction Φ of 3^k instances of σ_k that each consider the ways in which two variables may be chosen from each of the (x_i, y_i, z_i), i.e. $x_i \sim y_i$ or $y_i \sim z_i$ or $x_i \sim z_i$ (where $\sim$ is infix for ρ). We need to show that this conjunction Φ entails τ_k (the converse is trivial). We will assume for contradiction that Φ is satisfiable but τ_k not. In the first instance of σ_k of Φ some atom must be true, and it will be of the form $x_i \sim y_i$ or $y_i \sim z_i$ or $x_i \sim z_i$. Once we have settled on one of these three, $p_i \sim q_i$, then we immediately satisfy 3^{k-1} of the conjunctions of Φ, leaving $2 \cdot 3^{k-1}$ unsatisfied. Now we can evaluate to true no more than one other among $\{x_i \sim y_i, y_i \sim z_i, x_i \sim z_i\} \setminus \{p_i \sim q_i\}$, without contradicting our assumptions. If we do evaluate this to true also, then we leave 3^{k-1} conjunctions unsatisfied. Thus we are now down to looking at variables with subscript other than i and in this fashion we have made the space one smaller, in total $k - 1$. Now, we will need to evaluate in Φ some other atom of the form $x_j \sim y_j$ or $y_j \sim z_j$ or $x_j \sim z_j$, for $j \neq i$. Once we have settled on at most two of these three then we immediately satisfy 3^{k-2} of the conjunctions remaining of Φ, leaving 3^{k-2} still unsatisfied. Iterating this thinking, we arrive at a situation in which 1 clause is unsatisfied after we have gone through all k subscripts, which is a contradiction. ◀

▶ **Theorem 10.** *Let $\mathbb{A}$ be an idempotent algebra on a finite domain A. If $\mathbb{A}$ satisfies EGP, then QCSP*(Inv($\mathbb{A}$)) *is co-NP-hard.*

Proof. We know from Lemma 11 in [20] that there exist α, β strict subsets of A so that $\alpha \cup \beta = A$ and the relation σ_k is in Inv($\mathbb{A}$), for each $k \in \mathbb{N}$. From Lemma 9, we know also that τ_k is in Inv($\mathbb{A}$), for each $k \in \mathbb{N}$.

We will next argue that τ_k enjoys a relatively small specification in DNF (at least, polynomial in k). We first give such a specification for $\rho'(x, y, z)$.

$$\rho'(x, y, z) := \bigvee_{a, a', a'' \in \alpha} x = a \wedge y = a' \wedge z = a'' \vee \bigvee_{b, b', b'' \in \beta} x = b \wedge y = b' \wedge z = b''$$

which is constant in size when A is fixed. Now it is clear from the definition that the size of τ_n is polynomial in n.

We will now give a very simple reduction from the complement of 3NAESAT to QCSP(Inv($\mathbb{A}$)). 3NAESAT is well-known to be NP-complete [17] and our result will follow.

Take an instance ϕ of 3NAESAT which is the existential quantification of a conjunction of k atoms $\mathrm{NAE}(x,y,z)$. Thus $\neg\phi$ is the universal quantification of a disjunction of k atoms $x=y=z$. We build our instance ψ of $\mathrm{QCSP}(\mathrm{Inv}(\mathbb{A}))$ from $\neg\phi$ by transforming the quantifier-free part $x_1=y_1=z_1\vee\ldots\vee x_k=y_k=z_k$ to $\tau_k=\rho'(x_1,y_1,z_1)\vee\ldots\vee\rho'(x_k,y_k,z_k)$.

($\neg\phi\in$ co-3NAESAT implies $\psi\in\mathrm{QCSP}(\mathrm{Inv}(\mathbb{A}))$.) From an assignment to the universal variables $v_1,\ldots,v_m$ of ψ to elements $x_1,\ldots,x_m$ of A, consider elements $x'_1,\ldots,x'_m\in\{0,1\}$ according to

- $x_i\in\alpha\setminus\beta$ implies $x'_i=0$,
- $x_i\in\beta\setminus\alpha$ implies $x'_i=1$, and
- $x_i\in\alpha\cap\beta$ implies we don't care, so w.l.o.g. say $x'_i=0$.

The disjunct that is satisfied in the quantifier-free part of $\neg\phi$ now gives the corresponding disjunct that will be satisfied in τ_k.

($\psi\in\mathrm{QCSP}(\mathrm{Inv}(\mathbb{A}))$ implies $\neg\phi\in$ co-3NAESAT.) From an assignment to the universal variables $v_1,\ldots,v_m$ of $\neg\phi$ to elements $x_1,\ldots,x_m$ of $\{0,1\}$, consider elements $x'_1,\ldots,x'_m\in A$ according to

- $x_i=0$ implies x'_i is some arbitrarily chosen element in $\alpha\setminus\beta$, and
- $x_i=1$ implies x'_i is some arbitrarily chosen element in $\beta\setminus\alpha$.

The disjunct that is satisfied in τ_k now gives the corresponding disjunct that will be satisfied in the quantifier-free part of $\neg\phi$. ◀

The demonstration of co-NP-hardness in the previous theorem was inspired by a similar proof in [1]. Note that an alternative proof that τ_k is in $\mathrm{Inv}(\mathbb{A})$ is furnished by the observation that it is preserved by all $\alpha\beta$-projections (see [20]). We note surprisingly that co-NP-hardness in Theorem 10 is optimal, in the sense that some (but not all!) of the cases just proved co-NP-hard are also in co-NP.

▶ **Proposition 11.** *Let α,β strict subsets of $A:=\{a_1,\ldots,a_n\}$ so that $\alpha\cup\beta=A$ and $\alpha\cap\beta\neq\emptyset$. Then QCSP$(A;\{\tau_k:k\in\mathbb{N}\},a_1,\ldots,a_n)$ is in co-NP.*

Proof. Assume $|A|>1$, i.e. $n>1$ (note that the proof is trivial otherwise). Let ϕ be an input to $\mathrm{QCSP}(A;\{\tau_k:k\in\mathbb{N}\},a_1,\ldots,a_n)$. We will now seek to eliminate atoms $v=a$ ($a\in\{a_1,\ldots,a_n\}$) from ϕ. Suppose ϕ has an atom $v=a$. If v is universally quantified, then ϕ is false (since $|A|>1$). Otherwise, either the atom $v=a$ may be eliminated with the variable v since v does not appear in a non-equality relation; or ϕ is false because there is another atom $v=a'$ for $a\neq a'$; or $v=a$ may be removed by substitution of a into all non-equality instances of relations involving v. This preprocessing procedure is polynomial and we will assume w.l.o.g. that ϕ contains no atoms $v=a$. We now argue that ϕ is a yes-instance iff ϕ' is a yes-instance, where ϕ' is built from ϕ by instantiating all existentially quantified variables as any $a\in\alpha\cap\beta$. The universal ϕ' can be evaluated in co-NP (one may prefer to imagine the complement as an existential $\neg\phi'$ to be evaluated in NP) and the result follows. ◀

In fact, this being an algebraic paper, we can even do better. Let $\mathcal{B}$ signify a set of relations on a finite domain but not necessarily itself finite. For convenience, we will assume the set of relations of $\mathcal{B}$ is closed under all co-ordinate projections and instantiations of constants. Call $\mathcal{B}$ *existentially trivial* if there exists an element $c\in B$ (which we call a *canon*) such that for each k-ary relation R of $\mathcal{B}$ and each $i\in[k]$, and for every $x_1,\ldots,x_k\in B$, whenever $(x_1,\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_k)\in R^{\mathcal{B}}$ then also $(x_1,\ldots,x_{i-1},c,x_{i+1},\ldots,x_k)\in R^{\mathcal{B}}$. We want to expand this class to *almost existentially trivial* by permitting conjunctions of the form $v=a_i$ or $v=v'$ with relations that are existentially trivial.

▶ **Lemma 12.** *Let α, β be strict subsets of $A := \{a_1, \ldots, a_n\}$ so that $\alpha \cup \beta = A$ and $\alpha \cap \beta \neq \emptyset$. The set of relations pp-definable in $(A; \{\tau_k : k \in \mathbb{N}\}, a_1, \ldots, a_n)$ is almost existentially trivial.*

Proof. Consider a formula with a pp-definition in $(A; \{\tau_k : k \in \mathbb{N}\}, a_1, \ldots, a_n)$. We assume that only free variables appear in equalities since otherwise we can remove these equalities by substitution. Now existential quantifiers can be removed and their variables instantiated as the canon c. Indeed, their atoms τ_n may now be removed since they will always be satisfied. Thus we are left with a conjunction of equalities and atoms τ_n, and the result follows. ◀

▶ **Proposition 13.** *If $\mathcal{B}$ is comprised exclusively of relations that are almost existentially trivial, then QCSP($\mathcal{B}$) is in co-NP under the **DNF encoding**.*

Proof. The argument here is quite similar to that of Proposition 11 except that there is some additional preprocessing to find out variables that are forced in some relation to being a single constant or pairs of variables within a relation that are forced to be equal. In the first instance that some variable is forced to be constant in a k-ary relation, we should replace with the $(k-1)$-ary relation with the requisite forcing. In the second instance that a pair of variables are forced equal then we replace again the k-ary relation with a $(k-1)$-ary relation as well as an equality. Note that projecting a relation to a single or two co-ordinates can be done in polynomial time because the relations are encoded in DNF. After following these rules to their conclusion one obtains a conjunction of equalities together with relations that are existentially trivial. Now is the time to propagate variables to remove equalities (or find that there is no solution). Finally, when only existentially trivial relations are left, all remaining existential variables may be evaluated to the canon c. ◀

▶ **Corollary 14.** *Let α, β be strict subsets of $A := \{a_1, \ldots, a_n\}$ so that $\alpha \cup \beta = A$ and $\alpha \cap \beta \neq \emptyset$. Then QCSP($\mathrm{Inv}(\mathrm{Pol}(A; \{\tau_k : k \in \mathbb{N}\}, a, \ldots, a_n))$) is in co-NP under the **DNF encoding**.*

This last result, together with its supporting proposition, is the only time we seem to require the "nice, simple" DNF encoding, rather than arbitrary propositional logic. We do not require DNF for Proposition 11 as we have just a single relation in the signature for each arity and this is easy to keep track of. We note that the set of relations $\{\tau_k : k \in \mathbb{N}\}$ is not maximal with the property that with the constants it forms a co-clone of existentially trivial relations. One may add, for example, $\alpha \times \beta \cup \beta \times \alpha$.

The following, together with our previous results, gives the refutation of the Alternative Chen Conjecture.

▶ **Proposition 15.** *Let α, β strict subsets of $A := \{a_1, \ldots, a_n\}$ so that $\alpha \cup \beta = A$ and $\alpha \cap \beta \neq \emptyset$. Then, for each finite signature reduct $\mathcal{B}$ of $(A; \{\tau_k : k \in \mathbb{N}\}, a_1, \ldots, a_n)$, QCSP($\mathcal{B}$) is in NL.*

Proof. We will assume $\mathcal{B}$ contains all constants (since we prove this case gives a QCSP in NL, it naturally follows that the same holds without constants). Take m so that, for each $\tau_i \in \mathcal{B}$, $i \leq m$. Recall from Lemma 9 that τ_i is pp-definable in σ_i. We will prove that the structure $\mathcal{B}'$ given by $(A; \{\sigma_k : k \leq m\}, a_1, \ldots, a_n)$ admits a $(3m+1)$-ary near-unanimity operation f as a polymorphism, whereupon it follows that $\mathcal{B}$ admits the same near-unanimity polymorphism. We choose f so that all tuples whose map is not automatically defined by the near-unanimity criterion map to some arbitrary $a \in \alpha \cap \beta$. To see this, imagine that this f were not a polymorphism. Then some $(3m+1)$ m-tuples in σ_i would be mapped to some tuple not in σ_i which must be a tuple $\bar{t}$ of elements from $\alpha \setminus \beta \cup \beta \setminus \alpha$. Note that column-wise

this map may only come from $(3m+1)$-tuples that have $3m$ instances of the same element. By the pigeonhole principle, the tuple $\bar{t}$ must appear as one of the $(3m+1)$ m-tuples in σ_i and this is clearly a contradiction.

It follows from [6] that QCSP($\mathcal{B}$) reduces to a polynomially bounded ensemble of $\binom{n}{3m} \cdot n \cdot n^{3m}$ instances CSP($\mathcal{B}$), and the result follows. ◀

3.3 The question of the tuple encoding

▶ **Proposition 16.** *Let* $\alpha := \{0,1\}$ *and* $\beta := \{0,2\}$. *Then, QCSP*$(\{0,1,2\}; \{\tau_k : k \in \mathbb{N}\}, 0, 1, 2)$ *is in P under the* ***tuple encoding****.*

Proof. Consider an instance ϕ of this QCSP of size n involving relation τ_m but no relation τ_k for $k > m$. The number of tuples in τ_m is $> 3^m$. Following Proposition 11 together with its proof, we may assume that the instance is strictly universally quantified over a conjunction of atoms (involving also constants). Now, a universally quantified conjunction is true iff the conjunction of its universally quantified atoms is true. We can further say that there are at most n atoms each of which involves at most $3m$ variables. Therefore there is an exhaustive algorithm that takes at most $O(n \cdot 3^{3m})$ steps with is $O(n^4)$. ◀

The proof of Proposition 16 suggests an alternative proof of Proposition 15, but placing the corresponding QCSP in P instead of NL. Proposition 16 shows that Chen's Conjecture fails for the tuple encoding in the sense that it provides a language $\mathcal{B}$, expanded with constants, so that Pol($\mathcal{B}$) has EGP, yet QCSP($\mathcal{B}$) is in P under the tuple encoding. However, it does not imply that the algebraic approach to QCSP violates Chen's Conjecture under the tuple encoding. This is because $(\{0,1,2\}; \{\tau_k : k \in \mathbb{N}\}, 0, 1, 2)$ is not of the form Inv($\mathbb{A}$) for some idempotent algebra $\mathbb{A}$. For this stronger result, we would need to prove QCSP(Inv(Pol($\{0,1,2\}; \{\tau_k : k \in \mathbb{N}\}, 0, 1, 2$))) is in P under the tuple encoding.

4 Switchability, Collapsability and the three-element case

An algebra $\mathbb{A}$ is a *G-set* if its domain is not one-element and every of its operations f is of the form $f(x_1, \ldots, x_k) = \pi(x_i)$ where $i \in [k]$ and π is a permutation on A. An algebra $\mathbb{A}$ contains a G-set as a *factor* if some homomorphic image of a subalgebra of $\mathbb{A}$ is a G-set. A *Gap Algebra* [6] is a three-element idempotent algebra that omits a G-set as a factor and is not Collapsible.

Our first task is the deduction of the following theorem, whose lengthy proof is omitted. For each of the following two theorems, α and β are chosen such that α, β are strict subsets of $\{0,1,2\}$, $\alpha \cup \beta = \{0,1,2\}$ and $\alpha \cap \beta \neq \emptyset$.

▶ **Theorem 17.** *Suppose* $\mathbb{A}$ *is a Gap Algebra that is not* $\alpha\beta$*-projective. Then, for every finite subset of* Δ *of Inv*($\mathbb{A}$)*, Pol*(Δ) *is Collapsible.*

Our second task is the deduction of the following theorem, whose lengthy proof is omitted.

▶ **Theorem 18.** *Suppose* $\mathbb{A}$ *is a* 3*-element idempotent algebra that is not* $\alpha\beta$*-projective, containing a* 2*-element G-set as a subalgebra. Then,* $\mathbb{A}$ *is Collapsible.*

▶ **Corollary 19.** *Suppose* $\mathbb{A}$ *is a* 3*-element idempotent algebra that is not EGP, i.e. is Switchable. Then, for every finite subset of* Δ *of Inv*($\mathbb{A}$)*, Pol*(Δ) *is Collapsible.*

Proof. Recall Lemma 11 in [20] that $\mathbb{A}$ has EGP iff there exists α and β such that α, β are strict subsets of D, $\alpha \cup \beta = D$, and all operations of $\mathbb{A}$ are $\alpha\beta$-projective.

If $\mathbb{A}$ does not contain a G-set as a factor, then $\mathbb{A}$ is a Gap Algebra and the result follows from Theorem 17. Otherwise, $\mathbb{A}$ contains a G-set as a factor. If $\mathbb{A}$ contains a G-set as a homomorphic image then $\mathbb{A}$ has EGP from [7]. Else, since $\mathbb{A}$ is 3-element, $\mathbb{A}$ contains a 2-element G-set as a subalgebra and we are in the situation of Theorem 18. ◀

5 A three-element vignette

We would love to be able to improve Theorem 2 to describe the boundary between those cases that are co-NP-complete and those that are Pspace-complete, if indeed such a result is true. However, even in the three-element case this appears challenging, but we are able to provide a variant vignette, whose proof is omitted.

▶ **Theorem 20.** *Let $\mathbb{A}$ be an idempotent algebra on a 3-element domain. Either*

- Π_k-*CSP*(Inv($\mathbb{A}$)) *is in NP, for all k; or*
- Π_k-*CSP*(Inv($\mathbb{A}$)) *is co-NP-complete, for all k; or*
- Π_k-*CSP*(Inv($\mathbb{A}$)) *is Π_2^{P}-hard, for some k.*

Note that the trichotomy of Theorem 20 does not hold for QCSP along the same boundary for, respectively, NP, co-NP-complete and Pspace-complete. For the semilattice-without-unit s it is known that Π_k-CSP(Inv(s)) is co-NP-complete, for all k, while QCSP(Inv(s)) is Pspace-complete [3].

6 Discussion

The major contribution of this paper is its discussion of the Chen Conjecture with two infinite-signature variants one of which is proved to hold (with encoding in "simple logic") and one of which fails (with the tuple listing).

In addition to this, the contribution is largely mathematical, examining the relationship between Switchability and Collapsibility in the three-element case. However, this mathematical study uncovers something of importance to the computer scientist who is not reconciled to infinite signatures! Since here it demonstrates that all three-element domain NP-memberships that may be shown by Switchability, may already be shown by Collapsibility.

The work associated with Theorem 17 is distinctly non-trivial and involves a new method, whereas the work associated with Theorem 18 uses known methods and involves mostly turning the handle with these. Similarly, the work involved with the three element vignette uses known methods on top of our earlier new results.

The Chen Conjecture in its original form remains open. As does the general question (for arbitrary finite domains) as to whether, if $\mathbb{A}$ is Switchable, all finite subsets $\mathcal{B}$ of Inv($\mathbb{A}$) are so that Pol($\mathcal{B}$) is Collapsible. However, to now prove the Chen Conjecture it is sufficient to prove, for any finite $\mathcal{B}$ expanded with all constants such that Pol($\mathcal{B}$) has EGP, that there exists polynomially (in i) computable pp-definitions (over $\mathcal{B}$) of the relations τ_i (where α and β are suitably chosen to witness EGP). A first step towards this is to establish whether there are even polynomially sized pp-definitions of these τ_i.

The appearance of a co-NP-complete QCSP is likely to be an anomaly of our introduction of infinite signatures. Such a QCSP is unlikely to exist with a finite signature (at least, nothing like this is hitherto known). Indeed, its presence might be used as an argument against the acceptance of infinite signatures, if it is interpreted as an aberration. For the reader in this mind, we ask to please review the earlier paean to infinite signatures.

Acknowledgements. We thank Hubie Chen and Michał Wrona for many useful discussions, as well as several anonymous referees for their advising on two previous drafts.

References

1 Manuel Bodirsky and Hubie Chen. Quantified equality constraints. *SIAM J. Comput.*, 39(8):3682–3699, 2010. doi:10.1137/080725209.

2 Manuel Bodirsky and Jan Kára. The complexity of equality constraint languages. *Theory of Computing Systems*, 3(2):136–158, 2008. A conference version appeared in the proceedings of CSR'06.

3 Ferdinand Börner, Andrei A. Bulatov, Hubie Chen, Peter Jeavons, and Andrei A. Krokhin. The complexity of constraint satisfaction games and qcsp. *Inf. Comput.*, 207(9):923–944, 2009. doi:10.1016/j.ic.2009.05.003.

4 A. Bulatov, A. Krokhin, and P. G. Jeavons. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005.

5 Catarina Carvalho, Florent R. Madelaine, and Barnaby Martin. From complexity to algebra and back: digraph classes, collapsibility and the PGP. In *30th Annual IEEE Symposium on Logic in Computer Science (LICS)*, 2015.

6 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.

7 Hubie Chen. Quantified constraint satisfaction and the polynomially generated powers property. *Algebra universalis*, 65(3):213–241, 2011. An extended abstract appeared in ICALP B 2008. doi:10.1007/s00012-011-0125-4.

8 Hubie Chen. Meditations on quantified constraint satisfaction. In *Logic and Program Semantics - Essays Dedicated to Dexter Kozen on the Occasion of His 60th Birthday*, pages 35–49, 2012. doi:10.1007/978-3-642-29485-3_4.

9 Hubie Chen and Peter Mayr. Quantified constraint satisfaction on monoids, 2016.

10 Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. SIAM Monographs on Discrete Mathematics and Applications 7, 2001.

11 Uwe Egly, Thomas Eiter, Hans Tompits, and Stefan Woltran. Solving advanced reasoning tasks using quantified boolean formulas. In *Proc. 17th Nat. Conf. on Artificial Intelligence and 12th Conf. on Innovative Applications of Artificial Intelligence*, pages 417–422. AAAI Press/ The MIT Press, 2000.

12 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.

13 Vladimir Kolmogorov, Andrei A. Krokhin, and Michal Rolinek. The complexity of general-valued csps. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1246–1258, 2015. doi:10.1109/FOCS.2015.80.

14 Florent R. Madelaine and Barnaby Martin. On the complexity of the model checking problem. *CoRR*, abs/1210.6893, 2012. Extended abstract appeared at LICS 2011 under the name "A Tetrachotomy for Positive First-Order Logic without Equality". URL: http://arxiv.org/abs/1210.6893.

15 Barnaby Martin. On the chen conjecture regarding the complexity of qcsps. *CoRR*, abs/1607.03819, 2016. URL: http://arxiv.org/abs/1607.03819.

16 Barnaby Martin and Dmitriy Zhuk. Switchability and collapsibility of gap algebras. *CoRR*, abs/1510.06298, 2015. URL: http://arxiv.org/abs/1510.06298.

17 Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

18 R.M. Smullyan. *Godel's Incompleteness Theorems*. Oxford Logic Guides. Oxford University Press, 1992. URL: https://books.google.co.uk/books?id=04zalcCdKZsC.

19 James Wiegold. Growth sequences of finite semigroups. *Journal of the Australian Mathematical Society (Series A)*, 43:16–20, 8 1987. Communicated by H. Lausch. doi:10.1017/S1446788700028925.

20 D. Zhuk. The Size of Generating Sets of Powers. *ArXiv e-prints*, April 2015. arXiv:1504.02121.

Induced Embeddings into Hamming Graphs*

Martin Milanič[1], Peter Muršič[2], and Marcelo Mydlarz[3]

1 **University of Primorska, IAM and FAMNIT, Koper, Slovenia**
martin.milanic@upr.si
2 **MSIS Department and RUTCOR, Rutgers University, Piscataway, NJ, USA**
mursic.peter@gmail.com
3 **Instituto de Industria, Universidad Nacional de General Sarmiento, Los Polvorines, Argentina and CONICET, Buenos Aires, Argentina**
mmydlarz@ungs.edu.ar

Abstract

Let d be a positive integer. Can a given graph G be realized in $\mathbb{R}^d$ so that vertices are mapped to distinct points, two vertices being adjacent if and only if the corresponding points lie on a common line that is parallel to some axis? Graphs admitting such realizations have been studied in the literature for decades under different names. Peterson asked in [Discrete Appl. Math., 2003] about the complexity of the recognition problem. While the two-dimensional case corresponds to the class of line graphs of bipartite graphs and is well-understood, the complexity question has remained open for all higher dimensions.

In this paper, we answer this question. We establish the NP-completeness of the recognition problem for any fixed dimension, even in the class of bipartite graphs. To do this, we strengthen a characterization of induced subgraphs of 3-dimensional Hamming graphs due to Klavžar and Peterin. We complement the hardness result by showing that for some important classes of perfect graphs – including chordal graphs and distance-hereditary graphs – the minimum dimension of the Euclidean space in which the graph can be realized, or the impossibility of doing so, can be determined in linear time.

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

Keywords and phrases gridline graph, Hamming graph, induced embedding, NP-completeness, chordal graph

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.28

1 Introduction

The main question addressed in this paper is the following: How difficult is it to determine if a given graph G can be realized in $\mathbb{R}^d$ so that vertices are mapped to distinct points and two vertices are adjacent if and only if the corresponding points are on a common line that is parallel to some axis? Let us refer to any such mapping as a *d-realization* of G and say that a graph is *d-realizable* if it has a d-realization. The class of d-realizable graphs was studied in the literature for decades, under diverse names such as arrow graphs (Cook, 1974 [13]), $(d-1)$-plane graphs and $(d-1)$-line graphs of d-partite d-uniform hypergraphs (Bermond et al., 1977 [3]; see also [29]), d-dimensional cellular graphs (Gurvich and Temkin, 1992 [25]), d-dimensional chessboard graphs (Staton and Wingard, 1998 [50]), and d-dimensional gridline

* The authors acknowledge the financial support from the Slovenian Research Agency (research core funding No. I0-0035 and P1-0285, projects N1-0032, J1-5433, J1-6720, J1-6743, and J1-7051).

graphs (Peterson, 2003 [46]). Recently, Sangha and Zito studied d-realizable graphs in the more general context of the so-called Line-of-Sight (LoS) networks [49] and showed that the independent set problem, known to be polynomially solvable in the class of 2-realizable graphs, is NP-complete in the class of 3-realizable graphs. For the small-dimensional cases, $d \in \{2, 3\}$, Peterson suggested an application of d-realizable graphs to robotics [46]: if the movement of a robot is restricted to be along axis-parallel directions only and turns are allowable only at certain points, then a shortest path in a d-realized graph gives the number of turns required. Further possible applications of d-realizable graphs belong to the area of wireless networks, via their connection with Line-of-Sight networks [23].

Despite many studies on d-realizable graphs in the literature, determining the computational complexity of recognizing d-realizable graphs has been elusive except for $d \in \{1, 2\}$, when d-realizable graphs coincide with complete graphs and with line graphs of bipartite graphs, respectively (and can be recognized in polynomial time). The main aim of this paper is to settle the question about recognition complexity of d-realizable graphs for $d \geq 3$, asked explicitly by Peterson in 2003 [46]. We show that for all $d \geq 3$, determining if a given graph is d-realizable is NP-complete, even for bipartite graphs. We also identify some tractable cases. We characterize d-realizable graphs (for any positive integer d) in the class of HHD-free graphs, a large class of perfect graphs containing chordal graphs and distance-hereditary graphs. The characterization leads to a linear time recognition algorithm.

Our approach is based on the fact that a graph G is d-realizable if and only if G is an induced subgraph of a Cartesian product of d complete graphs. Given two graphs G and H, their *Cartesian product* is the graph $G\Box H$ with vertex set $V(G) \times V(H)$ in which two vertices (u_1, u_2) and (v_1, v_2) are adjacent if and only if either $u_1v_1 \in E(G)$ and $u_2 = v_2$, or $u_1 = v_1$ and $u_2v_2 \in E(H)$. The Cartesian product is associative and commutative (in the sense that $G\Box H \cong H\Box G$ where $\cong$ denotes the graph isomorphism relation). Another name for Cartesian products of complete graphs is *Hamming graphs*; a Hamming graph is *d-dimensional* if it is the Cartesian product of d nontrivial complete graphs. The 3-dimensional Hamming graphs having all factors of the same size were studied in the literature under the name *cubic lattice graphs* [37, 12, 11, 1, 16], hence, 3-realizable graphs are exactly the induced subgraphs of cubic lattice graphs. Our results are based on a characterization of induced subgraphs of d-dimensional Hamming graphs due to Klavžar and Peterin [33], expressed in terms of the existence of a particular edge labeling. For the 3-dimensional case, we develop a more specific characterization based on induced cycles of the graph and use it to prove hardness of recognizing 3-realizable graphs via a reduction from the 3-edge-coloring problem in cubic graphs. The hardness of the 3-dimensional case forms the basis for establishing hardness for all higher dimensions.

Since a d-realizable graph is also $(d+1)$-realizable, the notion of d-realizability suggests a natural graph parameter. The *Cartesian dimension* of a graph $G = (V, E)$, denoted $\mathrm{Cdim}(G)$, is defined as the minimum non-negative integer d such that G is *d-realizable*, if such an integer exists, and ∞, otherwise. The infinite case can indeed occur, even some small graphs – the diamond, the 5-cycle, and the complete bipartite graph $K_{2,3}$, for example – cannot be realized in any dimension. Note that $\mathrm{Cdim}(G)$, when finite and strictly positive, is the minimum positive integer d such that G is an induced subgraph of the Cartesian product of d complete graphs. This point of view adds the Cartesian dimension of a graph to the list of graph dimensions studied in the literature related to various embeddings of graphs into Cartesian product graphs [24, 20, 27, 34]. Other dimensions were studied related to the strong product [22, 15, 32, 47] and the direct product of graphs [41, 48].

Related work. As already mentioned, concepts equivalent to d-realizable graphs were studied in the literature in various contexts [13, 3, 29, 25, 50, 46, 49]. Much further work in the literature deals exclusively with the two-dimensional case [26, 28, 14, 46, 2], which (as we will discuss in Section 2) corresponds to the class of line graphs of bipartite graphs, one of the basic building blocks in the structural decomposition of perfect graphs [10].

Among the many dimension parameters of graphs defined via product graphs, let us mention two that seem to be most closely related to the Cartesian dimension. A d-realization is said to be *irredundant* [34] (or: *d-dimensionally spanning* [49]) if for each $i \in \{1, \ldots, d\}$ some pair of adjacent vertices of G is mapped to a pair of points spanning a line that is parallel to the i-th coordinate axis. Based on this notion, Klavžar et al. [34] defined the Hamming dimension of a graph G, denoted by $\mathrm{Hdim}(G)$, as the largest integer d such that G has an irredundant d-realization, if such an integer exists, and ∞, otherwise. Note that the Cartesian dimension can be defined analogously, with "smallest" instead of "largest"; in particular, $\mathrm{Cdim}(G) \leq \mathrm{Hdim}(G)$. Strict inequality is possible (for example, if P_4 denotes the 4-vertex path, then $\mathrm{Cdim}(P_4) = 2$ and $\mathrm{Hdim}(P_4) = 3$) and the two dimensions are finite on the same set of graphs.

The second relevant dimension is a Dushnik-Miller type dimension of a graph, the so-called *product dimension*. This parameter, denoted simply by $\dim(G)$, is defined analogously to the Cartesian dimension but with respect to the direct product. Given two graphs G and H, their direct product is the graph $G \times H$ with vertex set $V(G) \times V(H)$ in which two vertices (u_1, u_2) and (v_1, v_2) are adjacent if and only if $u_1v_1 \in E(G)$ and $u_2v_2 \in E(H)$. The product dimension was introduced by Nešetřil and Rödl in [41] and studied by Lovász et al. in [39] and more recently by Chandran et al. [8]; see also [21]. Unlike the Cartesian dimension, the product dimension is finite for all graphs. The problem of computing the product dimension of a given graph was shown to be NP-hard [40], even in the special case of recognizing three-dimensional instances [36]. The Cartesian and the product dimensions of graphs are closely related in the two-dimensional case: since the Cartesian product of two complete graphs is isomorphic to the complement of their direct product, we have $\mathrm{Cdim}(G) \leq 2$ if and only if $\mathrm{Hdim}(\overline{G}) \leq 2$, where $\overline{G}$ denotes the complement of G.

The Cartesian dimension of graphs introduced in this paper should not be confused with any of the "Cartesian dimensions" of a graph studied by Burosch and Ceccherini [7]. They are defined similarly to the Hamming dimension $\mathrm{Hdim}(G)$ from [34], but with respect to various inclusion relations and with the relaxation that the factors are not restricted to be complete.

Structure of the paper. In Section 2 we collect the necessary definitions, summarize some characterizations of the two-dimensional case and a necessary condition for the general, d-dimensional case. In Section 3 we review a characterization of induced subgraphs of d-dimensional Hamming graphs due to Klavžar and Peterin and introduce two related results regarding the three-dimensional case. We build on these results in Section 4, where the NP-completeness of recognizing d-realizable graphs is established for all $d \geq 3$. A linear time algorithm for computing the Cartesian dimension of a given HHD-free graph is developed in Section 5, after the general problem is reduced to the biconnected case. We conclude the paper in Section 6. Due to space limitations, several proofs are omitted.

2 Preliminaries

All graphs considered in this paper will be finite, simple and undirected. By K_n, P_n, and C_n we denote the complete graph, the path, and the cycle with n vertices. By $K_{m,n}$ we denote the complete bipartite graph with parts of sizes m and n; the *claw* is the graph $K_{1,3}$. A *clique* (resp., *independent set*) in a graph G is a set of pairwise adjacent (resp., pairwise non-adjacent) vertices. By $\alpha(G)$ we denote the independence number of G, that is, the maximum size of an independent set in G. A *triangle* in G is a clique of size 3 in G. The *diamond* is the graph obtained by removing an edge from a K_4. For a vertex v in G, the *neighborhood* of v is the set of vertices in G adjacent to v. It is denoted by $N_G(v)$ (or simply by $N(v)$ if the graph will be clear from the context). The *degree* of v (in G) is the size of its neighborhood. A graph is *cubic* if all its vertices have degree 3. The *girth* of a graph G is the length of the shortest cycle in G (and ∞ if G is acyclic). Given a graph G and a set $U \subseteq V(G)$, we denote by $G[U]$ the subgraph of G induced by U. Given a set of graphs $\mathcal{F}$, a graph G is said to be *$\mathcal{F}$-free* if no induced subgraph of G is isomorphic to a graph from $\mathcal{F}$. A *cut vertex* in a connected graph G is a vertex whose removal disconnects the graph. Given a graph G, a *block* of G is a maximal connected subgraph of G without cut vertices. A graph G is *biconnected* if G itself is its only block. The disjoint union of two graphs G and H is denoted by $G + H$. For graph theoretic terms not defined here, see, e.g., [51].

Given a positive integer d, a *d-realization* of a graph $G = (V, E)$ is an injective mapping $\varphi_G : V \to \mathbb{R}^d$ such that two vertices $u, v \in V$ are adjacent if and only if $\varphi_G(u)$ and $\varphi_G(v)$ differ in exactly one coordinate. A graph G is said to be *d-realizable* if it has a d-realization. Note that G is d-realizable if and only if G has a d-realization $\varphi_G : V \to \mathbb{N}^d$. The *Cartesian dimension* of a graph $G = (V, E)$, denoted $\mathrm{Cdim}(G)$, is defined as the minimum non-negative integer d such that G is *d-realizable*, if such an integer exists, and ∞, otherwise. (Note that K_1 is the only graph of Cartesian dimension 0.)

Clearly, the only graphs of Cartesian dimension 1 are complete graphs of order at least two. Graphs of Cartesian dimension at most 2 coincide with line graphs of bipartite graphs, for which various characterizations and linear time recognition algorithms are known. Recall that a graph G is said to be *bipartite* it has a *bipartition*, that is, a pair (X, Y) of disjoint independent sets such that $X \cup Y = V(G)$. The *line graph* of a graph G is the graph denoted by $L(G)$ with vertex set $E(G)$, in which two distinct vertices are adjacent if and only if they have a common endpoint as edges in G. Line graphs of bipartite graphs were studied in the literature under various names such as graphs of $(0, 1)$-matrices [28], matrix graphs [14], two-dimensional chessboard graphs [50], (two-dimensional) gridline graphs [46], cellular graphs [25], and rooks graphs [2]. The characterization of line graphs of bipartite graphs in terms of forbidden induced subgraph was discovered and rediscovered many times: by Chartrand in 1964 [9], by Hedetniemi in 1971 [28], by Harary and Holzman in 1974[26], by Staton and Wingard in 1998 [50], and by Peterson in 2003 [46]. Furthermore, Staton and Wingard [50] and Peterson [46] established the connection with the Cartesian dimension. These characterizations are summarized in the following theorem.

▶ **Theorem 1.** *For every graph G, the following conditions are equivalent:*

1. $\mathrm{Cdim}(G) \leq 2$.
2. *G is the line graph of a bipartite graph.*
3. *G is $\{$claw, diamond, C_5, $C_7, \ldots\}$-free.*

For any positive integer d, Staton and Wingard proved the following necessary condition for a graph to be d-realizable.

▶ **Theorem 2** (Staton and Wingard [50]). *Every d-realizable graph is $\{K_{1,d+1}$, diamond, $K_{2,3}, C_5\}$-free.*

Staton and Wingard asked whether for $d \geq 3$, the list of forbidden induced subgraphs for the class of d-realizable graphs given by Theorem 2 is complete. This is not the case: Peterson constructed an infinite family of graphs that are minimally forbidden for d-realizability for all $d \geq 3$ [46, Figure 4] (see also [45]). However, the complete list of forbidden induced subgraphs is not known for any $d \geq 3$.

3 The Klavžar-Peterin characterization

In this section, we recall the characterization of induced subgraphs of d-dimensional Hamming graphs due to Klavžar and Peterin [33] and strengthen it in the 3-dimensional case. The characterization is expressed in terms of the existence of a particular edge labeling. Given a graph G, a *d-edge-labeling of G* is a mapping from $E(G)$ to some set L of *labels*, where $|L| = d$ (we often have $L = \{1, \ldots, d\}$). Given a d-edge-labeling ℓ of G and a set $F \subseteq E(G)$, we say that F is *ℓ-monochromatic* (or simply *monochromatic* if the labeling is clear from the context) if the labeling is constant on F, that is, if $e, e' \in F$ implies $\ell(e) = \ell(e')$. We extend the definition of monochromaticity to subgraphs of G in the obvious way. A *(d-)edge-coloring* is a (d-)edge-labeling such that no two incident edges share the same label. In the case of edge-colorings, labels may also be referred to as *colors*.

We say that a d-edge-labeling of G is a *(d-)KP-labeling* if it satisfies the following two conditions:

- **Condition 1**: every triangle is monochromatic.
- **Condition 2**: for every pair of distinct non-adjacent vertices u, v, there exist different labels i and j which both appear on every induced u, v-path.

Note that in a KP-labeling, every induced P_3 will be 2-edge-colored due to Condition 2; in particular, this implies that for triangle-free graphs, KP-labelings coincide with edge-colorings.

Since induced subgraphs of Hamming graphs are exactly the graphs of finite Cartesian dimension, the result of Klavžar and Peterin given by [33, Theorem 3.3] can be equivalently stated as follows.

▶ **Theorem 3** (Klavžar and Peterin [33]). *Let G be a connected graph. Then $\mathrm{Cdim}(G) < \infty$ if and only if G has a KP-labeling.*

The proof of Theorem 3 given in [33] actually shows the following more specific equivalence:

▶ **Theorem 4.** *For every connected graph G and a positive integer d, we have $\mathrm{Cdim}(G) \leq d$ if and only if G has a d-KP-labeling.*

We can find d-realizations of two graphs G and H such that $\mathrm{Cdim}(G) \leq \mathrm{Cdim}(H) = d$ when $d > 1$, using d-tuples over disjoint sets for the two graphs. The case $d = 1$ is exceptional: by definition, two different 1-tuples result in a pair of adjacent vertices. Thus, as all graphs of Cartesian dimension 1 are complete, the Cartesian dimension of any disconnected graph is at least 2. We record these observations for later use.

▶ **Observation 5.** *For every two graphs G and H, we have $\mathrm{Cdim}(G + H) = \max\{\mathrm{Cdim}(G), \mathrm{Cdim}(H), 2\}$.*

We now present two results for the 3-dimensional case. Both are related to the Klavžar-Peterin characterization and will be needed in our hardness proof for recognizing 3-realizable

graphs developed in Section 5. First, we show that the defining properties of a 3-KP-labeling are satisfied for a graph as soon as they are satisfied for the family of all its induced subgraphs isomorphic to a cycle or to a P_3.

▶ **Theorem 6.** *Let G be a graph. A 3-edge-labeling of G is a KP-labeling if and only if it satisfies the following two conditions:*

- **Condition 3***: for every induced cycle C of G, the restriction of the labeling to $E(C)$ is a KP-labeling of C.*
- **Condition 4***: no induced P_3 is monochromatic.*

Proof. The necessity of the two conditions is easy to see. If G is 3-KP-labeled and H is an induced subgraph of G, then the restriction of the labeling to $E(H)$ is a 3-KP-labeling of H, hence Condition 3 is necessary. Condition 4 follows from Condition 2.

In order to prove sufficiency, note that Condition 3 immediately implies Condition 1. Now, by way of contradiction suppose that there is a 3-edge-labeling $\ell : E(G) \to \{1, 2, 3\}$ satisfying Conditions 3 and 4, but not Condition 2. Since G violates Condition 2, it contains two different induced paths of length at least two, say P and Q, intersecting at their endpoints – call these vertices u and v – such that no pair of different labels appears on both P and Q. Due to Condition 4, on each of the paths P and Q at least two different labels appear. Since no pair of different labels appears on both P and Q, we may assume that P and Q take – alternatingly – labels $1, 2$ and $1, 3$, respectively. Moreover, assume that

(∗) P and Q were chosen so as to minimize $|V(P)| + |V(Q)|$.

Given a path R and two of its vertices x and y, denote by R_{xy} the subpath of R between x and y, and by V_R^{-xy} the set $V(R) \setminus \{x, y\}$. We say that a path is *k-labeled* if exactly k different labels appear on its edges.

We claim that $V_P^{-uv} \cap V_Q^{-uv} = \emptyset$. Indeed, suppose for a contradiction that $w \in V_P^{-uv} \cap V_Q^{-uv}$. Then, P_{uw} and Q_{uw} would be both 2-labeled (u and w cannot be adjacent due to $(*)$), only agreeing on label 1; thus, $P_{uw} \cup Q_{uw}$ would be 3-labeled, contradicting $(*)$.

For $t \in \{u, v\}$ and $xy \in E(G)$ with $(x, y) \in V_P^{-uv} \times V_Q^{-uv}$, a cycle $C = P_{tx}$-xy-Q_{yt} such that either P_{tx}-xy or xy-Q_{yt} is an induced path will be called a *PQ-cycle*. Note that a PQ-cycle cannot be 3-labeled: if – say – $P' = P_{tx}$-xy was an induced path, then P' and Q_{yt} would make evident a violation to $(*)$.

Condition 3 implies that the cycle $C_0 = P \cup Q$ cannot be induced. Let xy be a chord in C_0 ($\{x, y\} \cap \{u, v\} = \emptyset$) such that $x \in V(P)$ is closest to u (where the distance is measured within P), and y is the neighbor of x in Q closest to v (where the distance is measured within Q). Observe that each of $C_1 = P_{ux}$-xy-Q_{yu} and $C_2 = P_{vx}$-xy-Q_{yv} is either a PQ-cycle or a triangle, implying that neither of them is 3-labeled. Neither of them can be monochromatic either: if – say – C_1 was monochromatic then, as $E(C_0) \subset E(C_1) \cup E(C_2)$ while C_1 and C_2 share the label of xy, it would follow that C_2 was 3-labeled. Thus, C_1 and C_2 are 2-labeled.

As C_1 and C_2 are 2-labeled, they share exactly one label. By definition, any PQ-cycle contains a P_3 from either P or Q, hence (recalling that P and Q alternate labels $1, 2$ and $1, 3$, respectively), C_1 and C_2 share label 1. Such is then the label of xy. However, one of the two edges incident to x in P is also labeled with 1, forming with xy a monochromatic induced P_3 (as part of either C_1 or C_2), which contradicts Condition 4. ◀

Next, we characterize 3-KP-labelings of cycles. By Condition 1 in the definition of a KP-labeling, every 3-KP-labeling of a 3-cycle is constant. The next lemma analyzes longer cycles.

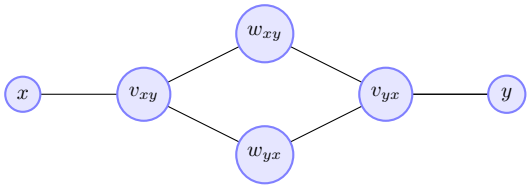


Figure 1 A gadget replacing each edge xy.

▶ **Lemma 7.** *Let C be a cycle of length at least 4. A 3-edge-coloring of C with colors $1, 2, 3$ is a KP-labeling if and only if*

- *either it is a 2-edge-coloring of C, or*
- *possibly after permuting the labels $1, 2, 3$, cycle C contains a cyclically ordered sequence of 6 distinct (not necessarily consecutive) edges labeled $1, 2, 3, 1, 2, 3$, respectively. We call this the* **123123-*condition***.

4 NP-completeness of testing realizability in $d \geq 3$ dimensions

In this section, we show that for every $d \geq 3$, determining whether $\mathrm{Cdim}(G) \leq d$ is NP-complete. First we establish the result for $d = 3$ and then derive from it the general case.

▶ **Theorem 8.** *Given a graph G, determining whether $\mathrm{Cdim}(G) \leq 3$ is NP-complete, even for connected bipartite graphs of maximum degree at most 3.*

Proof. A polynomially checkable certificate of the fact that $\mathrm{Cdim}(G) \leq 3$ is any 3-realization of G of the form $\varphi_G : V \to \mathbb{N}^3$. Therefore, the problem is in NP (on any class of input graphs).

To show hardness, we make a reduction from the 3-edge-coloring problem in cubic graphs, proved to be NP-complete by Holyer [30]. Let G be a cubic graph that is the input for the 3-edge-coloring problem. We may assume that G is connected. Construct a graph G' from G by replacing each edge xy of G with the structure shown in Fig. 1. Formally,

$$V(G') = V(G) \cup \bigcup_{xy \in E(G)} \{v_{xy}, w_{xy}, w_{yx}, v_{yx}\},$$

$$E(G') = \bigcup_{xy \in E(G)} \{xv_{xy}, v_{xy}w_{xy}, v_{xy}w_{yx}, v_{yx}w_{xy}, v_{yx}w_{yx}, yv_{yx}\}.$$

Letting $V_1 = V(G) \cup \bigcup_{xy \in E(G)} \{w_{xy}, w_{yx}\}$ and $V_2 = \bigcup_{xy \in E(G)} \{v_{xy}, v_{yx}\}$, we see that (V_1, V_2) is a bipartition of G'. Thus, G' is a bipartite graph with vertices of degrees 2 and 3 only. We will show that G is 3-edge-colorable if and only if $\mathrm{Cdim}(G') \leq 3$.

We first prove the (simpler) backward direction. Let $\mathrm{Cdim}(G') \leq 3$. By Theorem 4, G' has a 3-KP-labeling. Then for each $xy \in E(G)$ the 4-cycle $C = v_{xy}$-w_{xy}-v_{yx}-w_{yx}-v_{xy} in G' must be 2-KP-labeled. This implies that the edges xv_{xy} and yv_{yx} must have the same label ℓ_{xy} – the one not used in C. Since G' is triangle-free, any KP-labeling of G' is an edge-coloring (otherwise, Condition 4 would be violated). Therefore, by labeling each edge $xy \in E(G)$ with ℓ_{xy}, we get a 3-edge-coloring of G.

Now suppose that G has a 3-edge-coloring using colors $1, 2, 3$. For each edge xy of G labeled $i \in \{1, 2, 3\}$, let $\{j, k\} = \{1, 2, 3\} \setminus \{i\}$ and label the associated edges of G' as follows: edges xv_{xy} and yv_{yx} with i, edges $v_{xy}w_{xy}$ and $v_{yx}w_{yx}$ with j, and edges $v_{xy}w_{yx}$ and $v_{yx}w_{xy}$ with k.

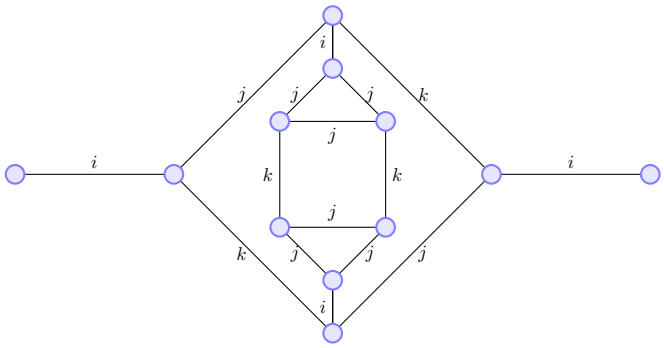


Figure 2 A gadget replacing each edge for proving hardness in cubic graphs.

We claim that the so obtained labeling of G' is a KP-labeling. By Theorem 6, it suffices to check that Conditions 3 and 4 are satisfied. The latter condition is obviously satisfied.

In order to verify that Condition 3 holds, note that G' has two types of induced cycles:

- 4-cycles. They only appear in the gadget of Fig. 1; they are properly 2-edge-colored and hence KP-labeled by Lemma 7.
- Cycles of length greater than 4. Each such cycle C has length $4p$ for some $p \geq 3$, and arises from a (not necessarily induced) p-cycle C' in G. We will show that such cycles satisfy the 123123-condition and apply Lemma 7. Let $x_1, x_2, \ldots, x_p$ be a cyclic order of vertices in C'. Without loss of generality, let $1, 2, 3, 1$ be the labels (in this order) on some shortest path from x_1 to x_2 in C. Then, the sequence of labels on the edges of any shortest path from x_2 to x_3 in C is one of the following: $(2, 1, 3, 2)$, $(2, 3, 1, 2)$, $(3, 1, 2, 3)$, or $(3, 2, 1, 3)$. Thus, along cycle C we find 6 distinct edges labeled $1, 2, 3, 1, 2, 3$ in order. This shows that C satisfies the 123123-condition.

It follows that Condition 3 is satisfied, hence by Theorem 6 G' has a 3-KP-labeling. By Theorem 4, we conclude that $\mathrm{Cdim}(G') \leq 3$. ◀

▶ Remark. A simple modification of the above construction, using a somewhat more involved gadget, can be used to show NP-completeness of testing whether $\mathrm{Cdim}(G) \leq 3$ for cubic (non-bipartite) graphs. We omit the details but show the gadget in Fig. 2 together with edge labels indicating how to extend a 3-edge-coloring of G to a 3-KP-labeling of G'.

▶ Remark. Recall that Peterson constructed an infinite family of graphs that are minimally forbidden for 3-realizability [46]. All those graphs are of girth 3. The above proof implies that the landscape of forbidden induced subgraphs for 3-realizability is much more complicated, consisting of graphs of arbitrarily large girth. To see this, note that for every positive integer g, there exists a graph F_g of maximum degree at most 3 and of girth at least g with $\mathrm{Cdim}(F_g) > 3$. This follows from the proof of Theorem 8 and the fact that there exist cubic graphs of arbitrarily large girth that are not 3-edge-colorable [35]. Since $\mathrm{Cdim}(F_g) > 3$, graph F_g contains a forbidden induced subgraph for 3-realizability, say F'_g. Since every acyclic graph of maximum degree at most 3 is 3-realizable (this follows, e.g., from Corollary 15 in Section 5.2), graph F'_g has a cycle and is therefore of (finite) girth at least g.

From Theorem 8 we derive hardness of recognizing graphs of any constant Cartesian dimension.

▶ **Theorem 9.** *For every $d \geq 3$, determining whether a given graph G satisfies $\mathrm{Cdim}(G) \leq d$ is NP-complete, even for connected bipartite graphs.*

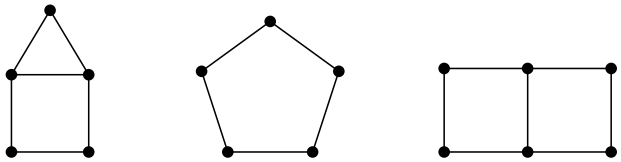


Figure 3 The house (left), the smallest hole (middle), and the domino (right).

Proof idea. The base case, $d = 3$, is given by Theorem 8. The inductive step can be established using the observation that for every connected bipartite graph G, the Cartesian product $G\Box K_2$ is also connected and bipartite, and satisfies $\mathrm{Cdim}(G\Box K_2) = \mathrm{Cdim}(G)+1$. ◀

5 Tractable cases: chordal graphs and distance-hereditary graphs

Since bipartite graphs are perfect, Theorem 8 implies that the problem of recognizing graphs of Cartesian dimension 3 is NP-complete in the class of perfect graphs. In this section, we show that the problem can be solved in linear time in two well-studied classes of perfect graphs: chordal graphs and distance-hereditary graphs. A graph G is *chordal* if it has no induced cycle of length at least four and *distance-hereditary* if in every connected induced subgraph of G, the distance between any pair of vertices is the same as in G. We characterize chordal graphs and distance-hereditary graphs of given Cartesian dimension. The characterizations will imply linear time algorithms for computing the Cartesian dimension of a given chordal or distance-hereditary graph.

We develop a unified approach that will imply both results, by considering the class of HHD-free graphs. We define a *hole* to be a cycle of length at least five.[1] A graph G is said to be *HHD-free* if it does not contain an induced subgraph isomorphic to the house, a hole, or the domino (see Fig. 3).

HHD-free graphs were introduced by Olariu [44] as a class of perfect graph generalizing both chordal and distance-hereditary graphs. They can be equivalently defined as the *(5,2)-chordal graphs*, that is, graphs in which every cycle of length at least five has at least two chords (see, e.g., [4]). Jamison and Olariu [31] characterized HHD-free graphs in terms of properties of the Lexicographic Breadth First Search algorithm, and Nikolopoulos and Palios gave an $O(|V(G)||E(G)|)$ time recognition algorithm [43]. Many other studies looked into metric, structural, and algorithmic properties of HHD-free graphs (see, e.g., [18, 6, 19, 42, 19, 17, 5]).

We characterize HHD-free graphs of a given Cartesian dimension and derive the corresponding results for chordal and distance-hereditary graphs as corollaries. We do this by first showing that the problem of computing the Cartesian dimension of an arbitrary graph can be reduced to its blocks (Lemma 10), and by identifying two particularly nice cases of this reduction (Lemmas 11 and 12). Next, we characterize biconnected HHD-free graphs of a given Cartesian dimension. To this end, we apply the necessary conditions for graphs of finite Cartesian dimension given by Theorem 2 to reduce the problem to the biconnected {diamond, $K_{2,3}$}-free HHD-free graphs, which we characterize in Lemma 13. Finally, the simple structure of the biconnected {diamond, $K_{2,3}$}-free HHD-free graphs (they can only be complete or 4-cycles) is used to prove the desired characterization (Theorem 14) and a linear time algorithm for computing the Cartesian dimension of an HHD-free graphs (Theorem 16).

[1] We remark that the terminology on holes is not completely uniform in the graph theory literature. In many papers, holes are defined as cycles of length at least four.

5.1 Reduction to blocks

For a graph G and a vertex $v \in V(G)$, we set $\alpha_G(v) = \alpha(G[N(v)])$ and $\alpha_1(G) = \max\{\alpha_G(v) : v \in V(G)\}$. Note that $\alpha_1(G)$ is the maximum value of n such that $K_{1,n}$ is an induced subgraph of G. Hence, by Theorem 2, every graph G has $\mathrm{Cdim}(G) \geq \alpha_1(G)$. The following lemma specifies the reduction for the problem of computing the Cartesian dimension of a given graph to the biconnected case.

▶ **Lemma 10.** *Let G be a connected graph with a cut vertex v, let $V(G) = \{v\} \cup V_1 \cup V_2$ where V_1 and V_2 are disjoint non-empty subsets of $V(G) \setminus \{v\}$ such that no vertex from V_1 is adjacent to a vertex in V_2, and let $G_i = G[\{v\} \cup V_i]$ for $i \in \{1, 2\}$. Then,* $\mathrm{Cdim}(G) = \max\{\mathrm{Cdim}(G_1), \mathrm{Cdim}(G_2), \alpha_G(v)\}$.

Lemma 10 has two useful consequences. For a connected graph G, we denote by C_G the set of cut vertices of G and by $\mathcal{B}_G$ the set of blocks of G. The *block-cutpoint tree* of a connected graph G is the bipartite graph T with vertex set $\mathcal{B}_G \cup C_G$ in which $B \in \mathcal{B}_G$ is adjacent to $v \in C_G$ if and only if $v \in V(B)$. It is well known that T is a tree such that all leaves of T are blocks of G (see, e.g., [51]). A class of graphs is *hereditary* if it is closed under vertex deletion. We say that a graph G is *maxstar-dimensional* if $\mathrm{Cdim}(G) = \alpha_1(G)$.

▶ **Lemma 11.** *Let $\mathcal{G}$ be a hereditary class of graphs such that every biconnected graph in $\mathcal{G}$ is maxstar-dimensional. Then every connected graph in $\mathcal{G}$ is maxstar-dimensional.*

Let us call a graph *star-dimensional* if $\mathrm{Cdim}(G) = \alpha_G(v)$ for every $v \in V(G)$.

▶ **Lemma 12.** *Let $\mathcal{G}$ be a hereditary class of graphs such that every biconnected graph in $\mathcal{G}$ is star-dimensional. Then, every connected graph $G \in \mathcal{G}$ with a cut vertex satisfies*

$$\mathrm{Cdim}(G) = \max_{v \in C_G} \sum_{B \in \mathcal{B}_G : v \in B} \mathrm{Cdim}(B)\,.$$

5.2 Cartesian dimension of HHD-free graphs

The following lemma characterizes biconnected {diamond, $K_{2,3}$}-free HHD-free graphs. In the proof we will need the notion of a *block graph*, that is, a connected graph every block of which is complete.

▶ **Lemma 13.** *Let G be a biconnected {diamond, $K_{2,3}$}-free HHD-free graph. Then, G is either complete or a C_4.*

Proof. Let G be a biconnected {diamond, $K_{2,3}$}-free HHD-free graph. Consider first the case when G is chordal. Since connected diamond-free chordal graphs are exactly the block graphs (see, e.g., [38]), G is a block graph. Thus, since G is biconnected, it is complete.

Suppose now that G is not chordal. Since G has no induced cycles of length 5 or more but is not chordal, G has an induced C_4, say C. We want to show that $G = C$. First, note that every vertex of G not in C has at most one neighbor in C. Indeed, the neighborhood on C of a vertex $v \in V(G) \setminus V(C)$ consisting of at least three neighbors, exactly two neighbors that are adjacent, or exactly two neighbors that are non-adjacent, would lead to an induced subgraph of G isomorphic to an diamond, a house, or a $K_{2,3}$, respectively.

Let (v_1, v_2, v_3, v_4) be a cyclic order of vertices along C and let U denote the set of vertices in $V(G) \setminus V(C)$ adjacent to a vertex of C. Since every vertex in U has exactly one neighbor in C, the set U can be partitioned into pairwise disjoint sets, $U = U_1 \cup U_2 \cup U_3 \cup U_4$, where U_i is the set of vertices in U adjacent to v_i. Note that since G is domino-free, no vertex in

U_i is adjacent to a vertex in U_{i+1} (indices modulo 4). Moreover, since G is C_5-free, no vertex in U_i is adjacent to a vertex in U_{i+2} (indices modulo 4). Thus, if $i \neq j$, then no vertex in U_i is adjacent to a vertex in U_j.

Suppose for a contradiction that $G \neq C$. Since G is connected, the fact that $G \neq C$ implies that one of the sets U_i is non-empty, say (w.l.o.g.) $U_1 \neq \emptyset$. Let $X = V(C) \setminus \{v_1\}$. Since G is biconnected, it contains a U_1, X-path avoiding v_1. Let P be a shortest such path. Let us enumerate the vertices of P along the path as $w_1, \ldots, w_k$ where $w_1 \in U_1$ and $w_k \in X$, more specifically, $w_k = v_i$ for some (unique) $i \in \{2, 3, 4\}$. By minimality, P is an induced path in $G - v_1$; moreover, since there are no edges connecting a vertex in U_1 with a vertex in U_j for $j \neq 1$, we infer that $k \geq 4$. By the minimality of P, no internal vertex of P is in $U_1 \cup X$, moreover, $w_{k-1} \in U_i$ and $V(P) \cap (U_2 \cup U_3 \cup U_4) = \{w_{k-1}\}$. It follows that w_1 and possibly w_k are the only neighbors of v_1 on P. Now, if $i \in \{2, 4\}$, then $V(P) \cup \{v_1\}$ induces a cycle of length at least five in G, which is not possible. Similarly, if $i = 3$, then $V(P) \cup \{v_1, v_2\}$ induces a cycle of length at least six in G, again a contradiction. This shows that $G = C$, as claimed, and completes the proof. ◀

It is not difficult to verify that every graph $G \in \{C_4\} \cup \{K_n : n \geq 1\}$ is star-dimensional, with

$$\mathrm{Cdim}(G) = \alpha_1(G) = \begin{cases} 0, & \text{if } G \text{ is a } K_1; \\ 1, & \text{if } G \text{ is a } K_n \text{ with } n \geq 2; \\ 2, & \text{if } G \text{ is a } C_4. \end{cases}$$

By Lemma 13, every biconnected {diamond, $K_{2,3}$}-free HHD-free graph is star-dimensional.

Recall that the inequality $\mathrm{Cdim}(G) \geq \alpha_1(G)$ holds for every graph G, where $\alpha_1(G)$ is the maximum value of n such that $K_{1,n}$ is an induced subgraph of G. Lemmas 11 and 13 imply that equality holds in the case of HHD-free graphs of finite Cartesian dimension.

▶ **Theorem 14.** *For every connected HHD-free graph G,*

$$\mathrm{Cdim}(G) = \begin{cases} \alpha_1(G), & \textit{if } G \textit{ is } \{\textit{diamond}, K_{2,3}\}\textit{-free;} \\ \infty, & \textit{otherwise.} \end{cases}$$

Since the house, the domino, and each hole contain an induced cycle of length at least four, every chordal graph is HHD-free. Every distance-hereditary graph is also HHD-free; in fact, distance-hereditary graphs are known to be exactly the gem-free HHD-free graphs (see, e.g. [4]), where the *gem* is the graph obtained from the four-vertex path by adding to it a universal vertex. Theorem 14 therefore implies the following result.

▶ **Corollary 15.** *If a connected graph G is chordal or distance-hereditary, then*

$$\mathrm{Cdim}(G) = \begin{cases} \alpha_1(G), & \textit{if } G \textit{ is } \{\textit{diamond}, K_{2,3}\}\textit{-free;} \\ \infty, & \textit{otherwise.} \end{cases}$$

Observation 5 and Lemma 12 imply a linear time algorithm for computing the Cartesian dimension of a given HHD-free graph. We summarize its pseudocode in Algorithm 1 below and prove its correctness in Theorem 16.

▶ **Theorem 16.** *Algorithm 1 runs in time $O(|V(G)| + |E(G)|)$ and correctly computes the Cartesian dimension of a given HHD-free graph G (in particular, G may be a chordal graph or a distance-hereditary graph).*

Algorithm 1: Computing the Cartesian dimension of an HHD-free graph

Input: An HHD-free graph $G = (V, E)$.
Output: The value of $\mathrm{Cdim}(G)$.

compute the connected components $G_1, \ldots, G_r$ of G;
if $r > 1$ **then**
  run the algorithm recursively on each component of G;
  return $\max\{\max_{1 \le i \le r} \mathrm{Cdim}(G_i), 2\}$;
`// from now on,` G `is connected`
compute T, the block-cutpoint tree of G, C_G, the set of cut vertices of G, and $\mathcal{B}_G$, the set of its blocks;
if *G has a block that is not complete or a C_4* **then**
  return ∞;
`// from now on, each block of` G `is either complete or a` C_4
foreach $B \in \mathcal{B}_G$ **do**
  $$\mathrm{Cdim}(B) \leftarrow \begin{cases} 0, & \text{if } |V(B)| = 1 \\ 1, & \text{if } |V(B)| \ge 2 \text{ and } B_i \text{ is complete} \\ 2, & \text{if } B \text{ is a } C_4; \end{cases}$$
if $|\mathcal{B}_G| = 1$ **then**
  let $B \in \mathcal{B}_G$ and **return** $\mathrm{Cdim}(B)$;
foreach $v \in \mathcal{C}_G$ **do**
  $\alpha_G(v) \leftarrow \sum_{B \in \mathcal{B}_G : v \in B} \mathrm{Cdim}(B)$;
return $\max_{v \in C_G} \alpha_G(v)$;

▶ Remark. Lemma 10 determines how to efficiently combine KP-labelings of the blocks of a given graph G into a KP-labeling of G. Moreover, the proof of [33, Theorem 3.3] shows that a d-realization of a given d-KP-labeled graph can be computed in polynomial time. Hence, there exists a polynomial time algorithm that takes as input an HHD-free graph G of finite Cartesian dimension and outputs a d-realization of G where $d = \mathrm{Cdim}(G)$.

6 Conclusion

The main contribution of the present work is settling the computational complexity status of recognizing d-realizable graphs for any $d \ge 3$, answering thereby a question by Peterson from 2003. While the hardness result is valid already for the class of bipartite graphs, we identified an important class of perfect graphs for which the problem is solvable in linear time – the class of HHD-free graphs. Besides the question of identifying further graph classes where the problem of d-realizability is (in)tractable, the main question left open by this work is to determine the complexity status of the problem of deciding if a given graph G is d-realizable for some d (or, equivalently, whether its Cartesian dimension is finite). It would also be interesting to study in more detail the relation between the Cartesian and the Hamming dimensions of a graph, as both parameters can be defined in terms of the set of integers d such that the graph has an irredundant d-realization.

Acknowledgement. M. Mydlarz is grateful to Vašek Chvátal for insightful comments.

References

1 Martin Aigner. The uniqueness of the cubic lattice graph. *J. Combinatorial Theory*, 6:282–297, 1969.

2 Lowell W. Beineke, Izak Broere, and Michael A. Henning. Queens graphs. *Discrete Math.*, 206(1-3):63–75, 1999.

3 Jean-Claude Bermond, Marie-Claude Heydemann, and Dominique Sotteau. Line graphs of hypergraphs. I. *Discrete Math.*, 18(3):235–241, 1977.

4 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999.

5 Andreas Brandstädt, Van Bang Le, and Thomas Szymczak. Duchet-type theorems for powers of HHD-free graphs. *Discrete Math.*, 177(1-3):9–16, 1997.

6 Hajo J. Broersma, Elias Dahlhaus, and Ton Kloks. Algorithms for the treewidth and minimum fill-in of HHD-free graphs. In *Graph-theoretic concepts in computer science (Berlin, 1997)*, volume 1335 of *Lecture Notes in Comput. Sci.*, pages 109–117. Springer, Berlin, 1997.

7 Gustav Burosch and Pier Vittorio Ceccherini. On the Cartesian dimensions of graphs. *J. Combin. Inform. System Sci.*, 19(1-2):35–45, 1994. International Conference on Graphs and Hypergraphs (Varenna, 1991).

8 L. Sunil Chandran, Rogers Mathew, Deepak Rajendraprasad, and Roohani Sharma. Product dimension of forests and bounded treewidth graphs. *Electron. J. Combin.*, 20(3):Paper 42, 14, 2013.

9 Gary Theodore Chartrand. *GRAPHS AND THEIR ASSOCIATED LINE-GRAPHS*. ProQuest LLC, Ann Arbor, MI, 1964. Thesis (Ph.D.)–Michigan State University.

10 Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Ann. of Math. (2)*, 164(1):51–229, 2006.

11 Curtis R. Cook. Further characterizations of cubic lattice graphs. *Discrete Math.*, 4:129–138, 1973.

12 Curtis R. Cook. A note on the exceptional graph of the cubic lattice graph characterization. *J. Combinatorial Theory Ser. B*, 14:132–136, 1973.

13 Curtis R. Cook. Representations of graphs by n-tuples. In *Proceedings of the Fifth Southeastern Conference on Combinatorics, Graph Theory, and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1974)*, pages 303–316. Congressus Numerantium, No. X, Winnipeg, Man., 1974. Utilitas Math.

14 Curtis R. Cook, B. Devadas Acharya, and V. Mishra. Adjacency graphs. In *Proceedings of the Fifth Southeastern Conference on Combinatorics, Graph Theory and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1974)*, pages 317–331. Congressus Numerantium, No. X. Utilitas Math., Winnipeg, Man., 1974.

15 Alexander K. Dewdney. The embedding dimension of a graph. *Ars Combin.*, 9:77–90, 1980.

16 Thomas A. Dowling. Note on: "A characterization of cubic lattice graphs". *J. Combinatorial Theory*, 5:425–426, 1968.

17 Feodor F. Dragan and Falk Nicolai. LexBFS-orderings and powers of HHD-free graphs. *Int. J. Comput. Math.*, 71(1):35–56, 1999.

18 Feodor F. Dragan, Falk Nicolai, and Andreas Brandstädt. LexBFS-orderings and powers of graphs. In *Graph-theoretic concepts in computer science (Cadenabbia, 1996)*, volume 1197 of *Lecture Notes in Comput. Sci.*, pages 166–180. Springer, Berlin, 1997.

19 Feodor F. Dragan, Falk Nicolai, and Andreas Brandstädt. Powers of HHD-free graphs. *Int. J. Comput. Math.*, 69(3-4):217–242, 1998.

20 David Eppstein. The lattice dimension of a graph. *European J. Combin.*, 26(5):585–592, 2005.

21 Anthony B. Evans, Garth Isaak, and Darren A. Narayan. Representations of graphs modulo n. *Discrete Math.*, 223(1-3):109–123, 2000.

22 Shannon L. Fitzpatrick and Richard J. Nowakowski. The strong isometric dimension of finite reflexive graphs. *Discuss. Math. Graph Theory*, 20(1):23–38, 2000.

23 Alan Frieze, Jon Kleinberg, R. Ravi, and Warren Debany. Line-of-sight networks. *Combin. Probab. Comput.*, 18(1-2):145–163, 2009.

24 Ronald L. Graham and Peter M. Winkler. On isometric embeddings of graphs. *Trans. Amer. Math. Soc.*, 288(2):527–536, 1985.

25 Vladimir A. Gurvich and Mikhail A. Temkin. Cellular perfect graphs. *Dokl. Akad. Nauk*, 326(2):227–232, 1992.

26 F. Harary and C. Holzmann. Line graphs of bipartite graphs. *Rev. Soc. Mat. Chile*, 1:19–22, 1974.

27 Frank Harary. Cubical graphs and cubical dimensions. *Comput. Math. Appl.*, 15(4):271–275, 1988.

28 Stephen T. Hedetniemi. Graphs of (0, 1)-matrices. In *Recent Trends in Graph Theory (Proc. Conf., New York, 1970)*, pages 157–171. Lecture Notes in Mathematics, Vol. 186. Springer, Berlin, 1971.

29 Marie-Claude Heydemann and Dominique Sotteau. Line-graphs of hypergraphs. II. In *Combinatorics (Proc. Fifth Hungarian Colloq., Keszthely, 1976), Vol. I*, volume 18 of *Colloq. Math. Soc. János Bolyai*, pages 567–582. North-Holland, Amsterdam-New York, 1978.

30 Ian Holyer. The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, 1981.

31 Beverly Jamison and Stephan Olariu. On the semi-perfect elimination. *Adv. in Appl. Math.*, 9(3):364–376, 1988.

32 Janja Jerebic and Sandi Klavžar. On induced and isometric embeddings of graphs into the strong product of paths. *Discrete Math.*, 306(13):1358–1363, 2006.

33 Sandi Klavžar and Iztok Peterin. Characterizing subgraphs of Hamming graphs. *J. Graph Theory*, 49(4):302–312, 2005.

34 Sandi Klavžar, Iztok Peterin, and Sara Sabrina Zemljič. Hamming dimension of a graph—the case of Sierpiński graphs. *European J. Combin.*, 34(2):460–473, 2013.

35 Martin Kochol. Snarks without small cycles. *J. Combin. Theory Ser. B*, 67(1):34–47, 1996.

36 Luděk Kučera, Jaroslav Nešetřil, and Aleš Pultr. Complexity of dimension three and some related edge-covering characteristics of graphs. *Theoret. Comput. Sci.*, 11(1):93–106, 1980.

37 Renu Laskar. A characterization of cubic lattice graphs. *J. Combinatorial Theory*, 3:386–401, 1967.

38 Van Bang Le and Nguyen Ngoc Tuy. The square of a block graph. *Discrete Math.*, 310(4):734–741, 2010.

39 László Lovász, Jaroslav Nešetřil, and Aleš Pultr. On a product dimension of graphs. *J. Combin. Theory Ser. B*, 29(1):47–67, 1980.

40 J. Nešetřil and Aleš Pultr. A Dushnik-Miller type dimension of graphs and its complexity. In *Fundamentals of computation theory (Proc. Internat. Conf., Poznań-Kórnik, 1977)*, pages 482–493. Lecture Notes in Comput. Sci., Vol. 56. Springer, Berlin, 1977.

41 Jaroslav Nešetřil and Vojtěch Rödl. A simple proof of the Galvin-Ramsey property of the class of all finite graphs and a dimension of a graph. *Discrete Math.*, 23(1):49–55, 1978.

42 Stavros D. Nikolopoulos and Leonidas Palios. Recognizing HH-free, HHD-free, and Welsh-Powell opposition graphs. *Discrete Math. Theor. Comput. Sci.*, 8(1):65–82, 2006.

43 Stavros D. Nikolopoulos and Leonidas Palios. An $O(nm)$-time certifying algorithm for recognizing HHD-free graphs. *Theoret. Comput. Sci.*, 452:117–131, 2012.

44 Stephan Olariu. *Results on perfect graphs*. PhD thesis, School of Computer Science, McGill University, Montreal, 1986.

45 Dale Peterson. Gridline graphs and higher dimensional extensions. Technical report, RUTCOR, Rutgers University, 1995.

46 Dale Peterson. Gridline graphs: a review in two dimensions and an extension to higher dimensions. *Discrete Appl. Math.*, 126(2-3):223–239, 2003.

47 Svatopluk Poljak and Aleš Pultr. Representing graphs by means of strong and weak products. *Comment. Math. Univ. Carolin.*, 22(3):449–466, 1981.

48 Svatopluk Poljak, Vojtěch Rödl, and Aleš Pultr. On a product dimension of bipartite graphs. *J. Graph Theory*, 7(4):475–486, 1983.

49 Pavan Sangha and Michele Zito. Finding large independent sets in line of sight networks. In Daya Ram Gaur and N. S. Narayanaswamy, editors, *Algorithms and Discrete Applied Mathematics - Third International Conference, CALDAM 2017, Sancoale, Goa, India, February 16-18, 2017, Proceedings*, volume 10156 of *Lecture Notes in Computer Science*, pages 332–343. Springer, 2017.

50 William Staton and G. Clifton Wingard. On line graphs of bipartite graphs. *Util. Math.*, 53:183–187, 1998.

51 Douglas B. West. *Introduction to Graph Theory.* Prentice Hall, Inc., Upper Saddle River, NJ, 1996.

Structured Connectivity Augmentation*

Fedor V. Fomin[1], Petr A. Golovach[2], and Dimitrios M. Thilikos[3]

1 **Department of Informatics, University of Bergen, Norway**
2 **Department of Informatics, University of Bergen, Norway**
3 **CNRS, LIRMM, Université de Montpellier, France and National and Kapodistrian University of Athens, Greece**

Abstract

We initiate the algorithmic study of the following "structured augmentation" question: is it possible to increase the connectivity of a given graph G by superposing it with another given graph H? More precisely, graph F is the superposition of G and H with respect to injective mapping $\varphi\colon V(H) \to V(G)$ if every edge uv of F is either an edge of G, or $\varphi^{-1}(u)\varphi^{-1}(v)$ is an edge of H. Thus F contains both G and H as subgraphs, and the edge set of F is the union of the edge sets of G and $\varphi(H)$. We consider the following optimization problem. Given graphs G, H, and a weight function ω assigning non-negative weights to pairs of vertices of $V(G)$, the task is to find φ of minimum weight $\omega(\varphi) = \sum_{xy \in E(H)} \omega(\varphi(x)\varphi(y))$ such that the edge connectivity of the superposition F of G and H with respect to φ is higher than the edge connectivity of G. Our main result is the following "dichotomy" complexity classification. We say that a class of graphs $\mathcal{C}$ has *bounded vertex-cover number*, if there is a constant t depending on $\mathcal{C}$ only such that the vertex-cover number of every graph from $\mathcal{C}$ does not exceed t. We show that for every class of graphs $\mathcal{C}$ with bounded vertex-cover number, the problems of superposing into a connected graph F and to 2-edge connected graph F, are solvable in polynomial time when $H \in \mathcal{C}$. On the other hand, for *any* hereditary class $\mathcal{C}$ with unbounded vertex-cover number, both problems are NP-hard when $H \in \mathcal{C}$. For the unweighted variants of structured augmentation problems, i.e. the problems where the task is to identify whether there is a superposition of graphs of required connectivity, we provide necessary and sufficient combinatorial conditions on the existence of such superpositions. These conditions imply polynomial time algorithms solving the unweighted variants of the problems.

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

Keywords and phrases connectivity augmentation, graph superposition, complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.29

1 Introduction

In connectivity augmentation problems, the input is a (multi) graph and the objective is to increase edge or vertex connectivity by adding the minimum number (weight) of additional edges, called links. This is a fundamental combinatorial problem with a number of important applications, we refer to the books of Nagamochi and Ibaraki [12] and Frank [6] for a detailed introduction to the topic. In this paper we initiate the study of a "structural" connectivity augmentation problem, where the set of additional edges should satisfy some additional

* The two first authors were supported by the Research Council of Norway via the projects "CLASSIS" and "MULTIVAL". The third author has been supported by project "DEMOGRAPH" (ANR-16-CE40-0028).
Emails of authors: {fedor.fomin, petr.golovach}@ii.uib.no, sedthilk@thilikos.info .

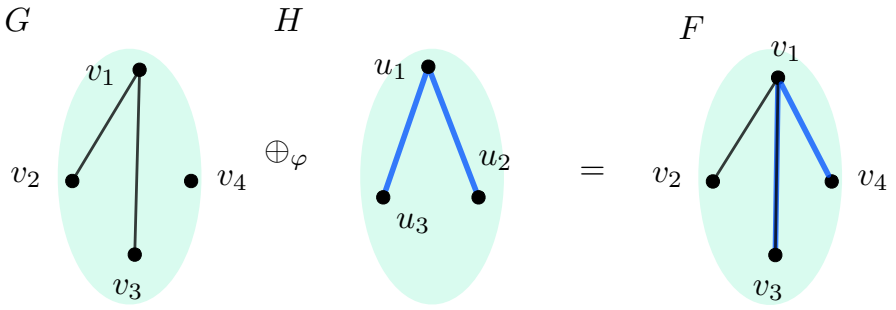


Figure 1 For injective mapping $\varphi\colon V(H) \to V(G)$ such that $\varphi(u_1) = v_1$, $\varphi(u_2) = v_4$, and $\varphi(u_3) = v_3$, we have $F = G \oplus_\varphi H$.

constrains. For example, such constrains can be that all new edges should be visible from one vertex, i.e. the new set of edges forms a star, forms a cycle, or can be controlled from a small set of vertices, i.e. the graph formed by the additional edges has a small vertex cover.

It is convenient to model such an augmentation problem as a *graph superposition* problem. Let G and H be simple graphs (i.e. graphs without loops and multiple edges), $|V(G)| \geq |V(H)|$, and let $\varphi\colon V(H) \to V(G)$ be an injective mapping of the vertices of H to the set of vertices of $V(G)$. We say that a simple graph F is the *superposition of G and H with respect to φ* and write $F = G \oplus_\varphi H$ if $V(F) = V(G)$ and two distinct vertices $u, v \in V(F)$ are adjacent in F if and only if $uv \in E(G)$ or $u, v \in \varphi(V(H))$ and $\varphi^{-1}(u)\varphi^{-1}(v) \in E(H)$. See Fig. 1 for an example. Thus graph F contains G and H as subgraphs, and the edge set of F is the union of the edge sets of G and $\varphi(H)$.

We study the algorithmic problem of increasing the edge-connectivity of graph G by superposing it with a graph H. We are interested in the weighted variant of the problem, where for every pair of vertices v and u of G, mapping the endpoints of an edge of H to u and v has a specified weight $\omega(uv)$. We consider the following problem.

Structured Connectivity Augmentation

Input:	Graphs G and H, a weight function $\omega\colon \binom{V(G)}{2} \to \mathbb{N}_0$, and a nonnegative integer W.
Task:	Decide whether there is an injective mapping $\varphi\colon V(H) \to V(G)$ such that graph $F = G \oplus_\varphi H$ is connected and the *weight* of the mapping $\omega(\varphi) = \sum_{xy \in E(H)} \omega(\varphi(x)\varphi(y)) \leq W$.

We also study the problem of obtaining a 2-edge connected graph F by superposing graphs G and H. More precisely, we consider the following problem.

Structured 2-Connectivity Augmentation

Input:	Connected graph G and a graph H, a weight function $\omega\colon \binom{V(G)}{2} \to \mathbb{N}_0$ and a nonnegative integer W.
Task:	Decide whether there is an injective mapping $\varphi\colon V(H) \to V(G)$ of weight at most W such that $F = G \oplus_\varphi H$ is 2-edge connected.

Our results. Our main result is the following "dichotomy" complexity classification of structured augmentation problems. We say that a class of graphs $\mathcal{C}$ has *bounded vertex-cover number*, if there is a constant t depending on $\mathcal{C}$ only such that the vertex-cover number of every graph from $\mathcal{C}$ does not exceed t. We show that for every class of graphs $\mathcal{C}$ with bounded

vertex-cover number, Structured Connectivity Augmentation and Structured 2-Connectivity Augmentation are solvable in polynomial time when $H \in \mathcal{C}$. We complement this result by showing that for *any* hereditary class $\mathcal{C}$ with unbounded vertex-cover number, both problems are NP-complete when $H \in \mathcal{C}$. Thus for any hereditary class $\mathcal{C}$ both problems with $H \in \mathcal{C}$ are NP-complete if and only if $\mathcal{C}$ has unbounded vertex-cover number.

The running times of our algorithms solving Structured Connectivity Augmentation and Structured 2-Connectivity Augmentation are of the form $|V(G)|^{\mathcal{O}(f(t))} \cdot \log W$, where f is some function and t is the vertex cover of H. Thus our algorithms are not fixed-parameter tractable when t is the parameter. We show that from the perspective of parameterized complexity, this situation is unavoidable. More precisely, we show that both problems are W[1]-hard when parameterized by t. We refer to the book of Downey and Fellows [2] for an introduction to parameterized complexity.

We also consider the unweighted variants of Structured Connectivity Augmentation and Structured 2-Connectivity Augmentation. In these cases, the weight is $\omega(uv) = 0$ for every pair of vertices of G and $W = 0$. The task is to identify whether there is a superposition of graphs G and H of edge connectivity 1 or 2, correspondingly. Here we obtain necessary and sufficient combinatorial conditions of the existence of an injective function φ such that $F = G \oplus_\varphi H$ is edge k-connected provided that G is edge $(k-1)$-connected, $k = 1, 2$. These conditions imply polynomial time algorithms solving the unweighted variants of the problems.

Due to space constraints some proof are either just sketched or omitted in this extended abstract. The full details are available in [4].

Related work. The problem of increasing graph connectivity by adding additional edges is the classic and well-studied problem. It was first studied by Eswaran and Tarjan [3] and Plesnik [13] who showed that increasing the edge connectivity of a given graph to 2 by adding minimum number of additional augmenting edges is polynomial time solvable. Subsequent work in [14, 5] showed that this problem is also polynomial time solvable for any given target value of edge connectivity to be achieved. However, if the set of augmenting edges is restricted, that is, there are pairs of vertices in the graph which do not constitute a new edge, or if the augmenting edges have (non-identical) weights on them, then the problem of computing the minimum size (or weight) augmenting set is NP-complete [3]. Augmentation problems with constraints like simplicity-preserving augmentations, augmentations with partition constraints, or planarity requirements can be found in the literature, see the book of Nagamochi and Ibaraki [12] for further references.

Strongly relevant to structural augmentation is the Minimum Star Augmentation problem, see e.g. [12, Section 3.3.3] and [10]. Here one wants to increase the edge-connectivity of a given graph by adding a new vertices and connecting it with a small number of edges to the remaining vertices of the graph. In our setting this corresponds to the case of graph G having an isolate vertex, and graph H being a star (a tree with vertex-cover number 1). Tibor and Szigeti [10] studied a generalization of this problem where one wants to make a graph edge r-connected by attaching p stars of specified degrees. In particular, they provided combinatorial conditions which are necessary and sufficient for such an augmentation. Again, this problem can be seen as a special case of structural augmentation, where graph G has p isolated vertices and graph H is the union of stars of specified degrees.

2 Preliminaries

We consider only finite undirected graphs. For a graph G, $\binom{V(G)}{2}$ denotes the set of unordered pairs of distinct vertices of G. For uniformity, we denote the elements of $\binom{V(G)}{2}$ in the same way as edges, i.e., we write $uv \in \binom{V(G)}{2}$. A subgraph H of G is *spanning* if $V(H) = V(G)$. For a graph G and a subset $U \subseteq V(G)$ of vertices, we write $G[U]$ to denote the subgraph of G induced by U. We write $G - U$ to denote the graph $G[V(G) \setminus U]$. Let $S \subseteq E(G)$ for a graph G. By $G - S$ we denote by $G - S$ the graph obtained by the deletion of the edges of S. We write $G - e$ instead of $G - \{e\}$ for an edge e. For a vertex v, we denote by $N_G(v)$ the *(open) neighborhood* of v, i.e., the set of vertices that are adjacent to v in G. Two nonadjacent vertices u and v are *(false) twins* if $N_G(u) = N_G(v)$. A set of edges with pairwise distinct end-vertices is called a *matching*. A matching M is *induced* if the end-vertices of M are pairwise nonadjacent. A vertex v is *saturated* in a matching M if v is incident to an edge of M. We say that the disjoint union of copies of K_2 is a *matching graph*. A graph class $\mathcal{C}$ is said to be *hereditary* if for every $G \in \mathcal{C}$ and every induced subgraph H of G, $H \in \mathcal{C}$. A set of vertices $X \subseteq V(G)$ is a *vertex cover* of a graph G if every edge of G has at least one of its end-vertices in X. The minimum size of a vertex cover is called the *vertex-cover number* of G and is denoted by $\beta(G)$.

Let k be a positive integer. A graph G is *(edge) k-connected* if for every $S \subseteq E(G)$ with $|S| \leq k - 1$, $G - S$ is connected. Since we consider only edge connectivity, whenever we say that a graph G is k-connected, we mean that G is edge k-connected. We assume that every graph is 0-connected. A set of edges $S \subseteq E(G)$ of a connected graph G is an *edge separator* if $G - S$ is disconnected. An edge e of a connected graph G is a *bridge* if $\{e\}$ is a separator. Clearly, a connected graph is 2-connected if and only if it has no bridge. Let B be the set of bridges of a connected graph G. We call a component of $G - B$ a *biconnected component* of G. In other words, a biconnected component is an inclusion-wise maximal induced 2-connected subgraph of G. We say that a biconnected component L of a graph G is a *pendant biconnected component* (or simply a *pendant*) if a unique bridge of G is incident to $V(L)$. A biconnected component is *trivial* if it has a single vertex. For a graph G, we denote by $c(G)$ the number a components of G, and for a connected graph G, $p(G)$ is the number of pendants. We also denote by $i(G)$ the number of isolated vertices of G.

Let S be an inclusion-wise minimal edge separator of a connected graph G. Then $G - S$ has exactly two components C_1 and C_2. Let G be a spanning subgraph of F. We say that an edge $e \in E(F) \setminus E(G)$ *covers* a minimal separator S of G if e has its end-vertices in C_1 and C_2. The following observation about separators is useful.

▶ **Observation 1.** *Let $k \geq 2$ be an integer and let a $(k-1)$-connected graph G be a spanning subgraph of F. Then F is k-connected if and only if for each edge separator S of G with $|S| = k - 1$, F has an edge that covers it.*

We also need some additional terminology and folklore observations for the augmentation of a connected graph to a 2-connected graph. Let G be a connected graph and let x and y be distinct vertices of G. We say that a bridge uv of G *belongs* to an (x, y)-path P if $uv \in E(P)$. Similarly, a biconnected component Q is *crossed* by P if $V(Q) \cap V(P) \neq \emptyset$. The following observation show that the choice of an (x, y)-path is irrelevant if the biconnected components containing the end-vertices are given.

▶ **Observation 2.** *Let distinct $\{x_1, y_1\}$ and $\{x_1, y_2\}$ be pairs of distinct vertices of a connected graph G such that x_1, x_2 are in the same biconnected component of G and, similarly, y_1, y_2*

are in the same biconnected component of G. Let also P_1 and P_2 be (x_1, y_1) and (x_2, y_2)-paths respectively. Then the following holds:

- *a bridge uv of G belongs to P_1 if and only if uv belongs to P_2,*
- *a biconnected component Q is crossed by P_1 if and only if Q is crossed by P_2.*

▶ **Observation 3.** *Let u and v be distinct nonadjacent vertices of a connected graph G and let F be a graph obtained from G by the addition of the edge uv. Then uv covers all bridges that belongs to a (u, v)-path P in G, and for the biconnected components $Q_1, \ldots, Q_s$ that are crossed by P, $F[V(Q_1) \cup \ldots \cup V(Q_s)]$ is a biconnected component of F.*

In the remaining part of the paper, we will be always assuming that in the instance of the structured augmentation problem, we have

(i) $|V(H)| \leq |V(G)|$;

(ii) Graph H has no isolated vertices.

Indeed, if $|V(H)| > |V(G)|$, then there is no superposition of G and H, and thus such an instance is a no-instance. For (ii), it is sufficient to observe that mapping of isolated vertices of H to vertices of G does not influence the connectivity of the superposition. Another technical detail should be mentioned here. In Theorems 5 and 7, we evaluate the running times of algorithms as a function of $|V(G)|$ and the vertex cover number of H. In order to do this, we should be able to recognize within this time the (trivial) no-instances, where $|V(H)| > |V(G)|$. We can verify this condition in time $|V(G)|^{\mathcal{O}(1)}$ just by refuting the instances of size more than $|V(G)|^{\mathcal{O}(1)}$ after reading the first $|V(G)|^{\mathcal{O}(1)}$ bits.

3 Augmenting by graphs with small vertex cover

In this section we consider the situation when graph H is from a graph class $\mathcal{C}$ with bounded vertex-cover number. In Subsection 3.1 we show that in this case Structured Connectivity Augmentation and Structured 2-Connectivity Augmentation are solvable in polynomial time. In Subsection 3.2 we show that this condition is tight by proving that for any hereditary graph class $\mathcal{C}$ with unbounded vertex-cover number, both problems are NP-hard. Due to space restrictions, we only sketch our results.

3.1 Algorithms

We start with a solution for Structured Connectivity Augmentation, which is simpler than the solution for Structured 2-Connectivity Augmentation.

Structured Connectivity Augmentation. We need the following lemma.

▶ **Lemma 4.** *Let G and H be graphs and let $\varphi\colon V(H) \to V(G)$ be an injection such that $F = G \oplus_\varphi H$ is connected. Let also X be a vertex cover of H of size t. Then there is a set $Y \subseteq V(H) \setminus X$ of size at most $2(t-1)$ such that for graph $H' = H[X \cup Y]$ and mapping $\psi = \varphi|_{X \cup Y}$, the vertices of $\psi(X \cup Y)$ are in the same connected component of $F' = G \oplus_\psi H'$.*

Let us remind, that, given a positive integer t, a graph class $\mathcal{C}$ has vertex-cover number at most t if every graph $H \in \mathcal{C}$ has a vertex cover of size at most t. We are ready to prove the main theorem about Structured Connectivity Augmentation.

▶ **Theorem 5.** *Let t be a positive integer and $\mathcal{C}$ be a graph class of vertex-cover number at most t. Then for any $H \in \mathcal{C}$,* Structured Connectivity Augmentation *is solvable in time $|V(G)|^{\mathcal{O}(t)} \cdot \log W$.*

Sketch of the proof. Let G and $H \in \mathcal{C}$ be graphs and let $\omega\colon \binom{V(G)}{2} \to \mathbb{N}_0$ be a weight function. We show that we can find in time $|V(G)|^{\mathcal{O}(t)} \cdot \log W$ an injective mapping $\varphi\colon V(H) \to V(G)$ such that $F = G \oplus_\varphi H$ is connected and $\omega(\varphi) = \sum_{xy \in E(H)} \omega(\varphi(x)\varphi(y))$ is minimum if φ exists.

Let us remind that without loss of generality, we can assume that $|V(H)| \leq |V(G)|$ and H has no isolated vertices.

We start from finding a vertex cover X of size at most t in H. Since we aim for an algorithm with running time $|V(G)|^{\mathcal{O}(t)} \cdot \log W$, vertex cover X can be found by brute-force checking of all subsets of $V(H)$ of size at most t. If we fail to find X of size at most t, it means that $H \notin \mathcal{C}$, in this case we return the answer NO and stop. Assume that X exists.

Suppose that there is an injective mapping $\varphi\colon V(H) \to V(G)$ such that $F = G \oplus_\varphi H$ is connected and assume that for φ, $\omega(\varphi)$ is minimum. By Lemma 4, there is a set $Y \subseteq V(H) \setminus X$ of size at most $2(t-1)$ such that for $H' = H[X \cup Y]$ and $\psi = \varphi|_{X \cup Y}$, the vertices of $\psi(X \cup Y)$ are in the same component of $F' = G \oplus_\psi H'$. Considering all possibilities, we guess Y in time $|V(H)|^{\mathcal{O}(t)}$.

Now we consider all possible injective mapping $\psi\colon X \cup Y \to V(G)$ such that the vertices of $\psi(X \cup Y)$ are in the same connected component of $F' = G \oplus_\psi H'$, where $H' = H[X \cup Y]$. Notice that there are at most $|V(G)|^{3t-2}$ such mappings that can be generated in time $|V(G)|^{\mathcal{O}(t)}$. If we fail to find ψ, we reject the current choice of Y. Otherwise, for every ψ, we try to extend it to an injection $\varphi\colon V(H) \to V(G)$ such that $F = G \oplus_\varphi H$ is connected, and among all extensions we choose one that provides the minimum weight $\omega(\varphi)$.

Let $Z = V(H) \setminus (X \cup Y)$. The vertices of $\psi(X \cup Y)$ are in the same component of F'. Denote this component by F_0 and denote by $F_1, \ldots, F_r$ the other components of this graphs. Recall that Z is an independent set of H and each vertex of Z has an incident edge with one endpoint in X. It follows that for an injection $\varphi\colon V(H) \to V(G)$ such that $\psi = \varphi|_{X \cup Y}$, $F = G \oplus_\varphi H$ is connected if and only if for every $i \in \{1, \ldots, r\}$, there is $v \in V(F_i)$ such that $v \in \varphi(Z)$. Hence, if $r > |Z|$, we cannot extend ψ. In this case we discard the current choice of ψ.

Assume from now that Y and ψ are fixed, $F' = G \oplus_\psi H'$ is connected and $r \leq |Z|$. For $z \in Z$ and $v \in V(G) \setminus \psi(X \cup Y)$, we define the weight of mapping z to v as

$$w(z, v) = \sum_{u \in N_G(v) \cap \psi(N_H(z))} \omega(uv),$$

that is, $w(z, x)$ is the weight of edges that is added to the weight of mapping if we decide to extend ψ by mapping z to v. Let $W = \max\{w(z, v) \mid z \in Z, v \in V(G) \setminus \psi(X \cup Y)\} + 1$. We construct the weighted auxiliary bipartite graph $\mathcal{G}$ with the bipartition (A, B) of its vertex set and the weight function $f\colon E(\mathcal{G}) \to \mathbb{N}_0$ as follows.

- Set $A = (V(F_0) \setminus \psi(X \cup Y)) \cup V(F_1) \cup \ldots \cup V(F_r) = V(G) \setminus \psi(X \cup Y)$.
- Construct a set of vertices S_0 of size $|V(F_0)| - |X \cup Y|$ and sets S_i of size $|V(F_i)| - 1$ for $i \in \{1, \ldots, r\}$.
- Set $B = Z \cup S_0 \cup \ldots \cup S_r$.
- For each $z \in Z$ and $v \in A$, construct an edge zv and set $f(zv) = w(z, v)$.
- For each $u \in S_0$ and $v \in V(F_0) \setminus \psi(X \cup Y)$, construct an edge uv and set $f(uv) = W$.
- For each $\in \{1, \ldots, r\}$, do the following: for each $u \in S_i$ and $v \in V(F_i)$, construct an edge uv and set $f(uv) = W$.

We find a matching M in $\mathcal{G}$ that saturates every vertex of A and has the minimum weight using the Hungarian algorithm [7, 11] in time $\mathcal{O}(|V(G)|^3 \cdot \log W)$.

Observe that a matching that saturates every vertex of A exists, because $r \leq Z$. We can construct such a matching by selecting one vertex in $V(F_i)$ for each $i \in \{1, \ldots, r\}$ and matching it with a vertex of Z. Then we complement this set of edges to a matching saturating A by adding edges incident to $S_0 \cup \ldots \cup S_r$. For the matching M that has minimum weight, we can also observe the following.

First, note that

- every vertex of Z is saturated by M. (1)

Indeed, targeting towards a contradiction, assume that $z \in Z$ is not saturated. Since $|V(H)| \leq |V(G)|$, there is $uv \in M$ such that $u \in S_0 \cup \ldots \cup S_r$ and $v \in A$. We replace uv by zv in M. Because $f(uv) = W > w(zv)$, we obtain a matching with a smaller weight. This contradicts the choice of M.

Next, we claim that

- there is $zv \in M$ such that $z \in Z$ and $v \in V(F_i)$. (2)

Indeed, this is because the vertices of $V(F_i)$ are adjacent to $|V(F_i)| - 1$ vertices of S_i and all other their neighbors are in Z.

Finally, we have that among all matching saturating A, M is a matching satisfying (1) and (2) such that for $M' = \{zv \in M \mid z \in Z\}$, $f(M')$ is minimum. To see it, observe that $f(uv) = W$ for $uv \in M \setminus M'$. Hence, $f(M \setminus M') = (|A| - |Z|)W$, because $|M \setminus M'| = |A| - |Z|$ by (1). Therefore, $f(M') = f(M) - f(M \setminus M') = f(M) - (|A| - |Z|)W$.

For every $z \in Z$, we define $\varphi(z) = v$, where $zv \in M'$ and $\varphi(x) = \psi(x)$ for $x \in X \cup Y$. Clearly, φ is an extension of ψ. By (1), φ is an injective mapping of $V(H)$ to $V(G)$. By (2) and the choice of X and Y, we obtain that $G \oplus_\varphi H$ is connected. We claim that φ is an extension of ψ such that $F = G \oplus_\varphi H$ is connected that has the minimum total weight $\omega(\varphi) = \sum_{xy \in E(H)} \omega(\varphi(x)\varphi(y))$.

Recall that we try all possible choices of Y and for every choice of Y, we consider all possible choices of ψ. If we fail to find an injection $\varphi \colon V(H) \to V(G)$ such that φ is an extension of ψ and $F = G \oplus_\varphi H$ is connected we return the answer NO. Otherwise, we return φ that provides the minimum weight.

To complete the proof, observe that the total running time of the algorithm is $|V(G)|^{\mathcal{O}(t)} \cdot \log W$. ◀

Structured 2-Connectivity Augmentation. As it could be expected, the algorithm for Structured 2-Connectivity Augmentation is more technical. We start with a lemma, which is similar to Lemma 4. We show it by making use of Observations 1 and 3.

▶ **Lemma 6.** *Let G and H be graphs such that G is connected, and let $\varphi \colon V(H) \to V(G)$ be an injection such that $F = G \oplus_\varphi H$ is connected. Suppose that X is a vertex cover of H and $t = |X|$. Then there is a set $Y \subseteq V(H) \setminus X$ of size at most $2(t-1)$ such that for $H' = H[X \cup Y]$ and $\psi = \varphi|_{X \cup Y}$, the vertices of $\psi(X \cup Y)$ are in the same biconnected component of $F' = G \oplus_\psi H'$.*

▶ **Theorem 7.** *Let t be a positive integer and $\mathcal{C}$ be a graph class of vertex-cover number at most t. Then for any $H \in \mathcal{C}$,* Structured 2-Connectivity Augmentation *is solvable in time $|V(G)|^{\mathcal{O}(2^t)} \log W$.*

Sketch of the proof. Let G and H be graphs such that G is connected and $H \in \mathcal{C}$. Let $\omega \colon \binom{V(G)}{2} \to \mathbb{N}_0$ be a weight function. Similarly to the proof of Theorem 5 we show that we can find in time $|V(G)|^{\mathcal{O}(2^t)} \cdot \log W$ the minimum value of $\omega(\varphi) = \sum_{xy \in E(H)} \omega(\varphi(x)\varphi(y))$

for an injective mapping $\varphi\colon V(H) \to V(G)$ such that $F = G \oplus_\varphi H$ is connected if such a mapping φ exists.

The first steps of our algorithm are the same as in the proof of Theorem 5. Again, we remind that $|V(H)| \leq |V(G)|$ and that H has no isolated vertices.

Next, we find a vertex cover X of minimum size in H of size at most t in time $|V(G)|^{\mathcal{O}(t)}$. If we fail to find X of size at most t, then $H \notin \mathcal{C}$. We return NO and stop. From now on we assume that X exists.

Suppose that there is an injective mapping $\varphi\colon V(H) \to V(G)$ such that $F = G \oplus_\varphi H$ is 2-connected and assume that for φ, $\omega(\varphi)$ is minimum. By Lemma 6, there is a set $Y \subseteq V(H) \setminus X$ of size at most $2(t-1)$ such that for $H' = H[X \cup Y]$ and $\psi = \varphi|_{X \cup Y}$, the vertices of $\psi(X \cup Y)$ are in the same biconnected component of $F' = G \oplus_\psi H'$. Considering all possibilities, we guess Y in time $|V(H)|^{\mathcal{O}(t)}$.

Now we consider all possible injective mapping $\psi\colon X \cup Y \to V(G)$ such that the vertices of $\psi(X \cup Y)$ are in the same biconnected component of $F' = G \oplus_\psi H'$ where $H' = H[X \cup Y]$. Notice that there at most $|V(G)|^{3t-2}$ such mappings that can be generated in time $|V(G)|^{\mathcal{O}(t)}$. If we fail to find ψ, we reject the current choice of Y. Otherwise, for every ψ, we try to extend it to an injection $\varphi\colon V(H) \to V(G)$ such that $F = G \oplus_\varphi H$ is 2-connected, and among all extensions we choose one that provides the minimum weight $\omega(\varphi)$.

Let $Z = V(H) \setminus (X \cup Y)$. The vertices of $\psi(X \cup Y)$ are in the same biconnected component of F'. Denote this biconnected component by F_0 and denote by $F_1, \ldots, F_r$ the pendant biconnected components of F' that are distinct from F_0. Recall that Z is an independent set of H and each vertex of Z has an incident edge with one endpoint in X. By Observation 1, we obtain the following crucial property.

For an injection $\varphi\colon V(H) \to V(G)$ such that $\psi = \varphi|_{X \cup Y}$, $F = G \oplus_\varphi H$ is 2-connected if and only if

- **(i)** for every $i \in \{1, \ldots, r\}$, there is $v \in V(F_i)$ such that $v \in \varphi(Z)$, and
- **(ii)** if v is the unique element of $V(F_i) \cap \varphi(Z)$ and v is incident to a bridge vu of G, then there is $x \in X$ such that $\varphi(x) \neq u$ and x is adjacent to $\varphi^{-1}(v)$ in H.

Similarly to the proof of Theorem 5, we solve auxiliary matching problems to find the minimum weight of φ but now, due the condition (ii), the algorithm becomes more complicated and we are using dynamic programming.

For $z \in Z$ and $v \in V(G) \setminus \psi(X \cup Y)$, we define the weight of mapping z to v as

$$w(z, v) = \sum_{u \in N_G(v) \cap \psi(N_H(z))} \omega(uv), \tag{3}$$

that is, $w(z, x)$ is the weight of edges that is added to the weight of mapping if we decide to extend ψ by mapping z to v. Our aim is to find the extension φ of ψ that satisfies (i) and (ii) such that the total weight of the mapping of the vertices of Z to verices of $V(G) \setminus \psi(X \cup Y)$ by φ is minimum.

Since X is a vertex cover of H of size t, the set Z can be partitioned into $s \leq 2^t$ classes of false twins $Z_1, \ldots, Z_s$. Let $p_i = |Z_i|$ for $i \in \{1, \ldots, s\}$. We exploit the following property of false twins in Z: if $x, y \in Z_i$, then $w(x, v) = w(y, v)$ for $v \in V(G) \setminus \psi(X \cup Y)$.

For each s-tuple of integers $(q_1, \ldots, q_s)$ such that $0 \leq q_i \leq p_i$, for $i \in \{1, \ldots, s\}$ and each $h \in \{0, \ldots, r\}$, we define

$$\alpha_h(q_1, \ldots, q_s) = \min_{\xi} \sum_{z \in Z'} w(z, \xi(z)), \tag{4}$$

where $Z' \subseteq Z$ such that $|Z' \cap Z_i| = q_i$ for $i \in \{1, \ldots, s\}$ and the minimum is taken over all

injective mappings $\xi\colon Z' \to (V(F_0) \setminus \psi(X \cup Y)) \cup V(F_1) \cup \ldots \cup V(F_h)$ such that the following conditions are satisfied:

- **(a)** for every $i \in \{1, \ldots, h\}$, there is $v \in V(F_i)$ such that $v \in \xi(Z')$, and
- **(b)** if v is a unique element of $V(F_i) \cap \xi(Z')$ for some $i \in \{1, \ldots, h\}$ and v is incident to a bridge vu of G, then there is $x \in X$ such that $\psi(x) \neq u$ and x is adjacent to $\xi^{-1}(v)$ in H.

If such a mapping ξ does not exist, then we assume that $\alpha_h(q_1, \ldots, q_s) = +\infty$. Recall that if $x, y \in Z_i$, then $w(x, v) = w(y, v)$ for $v \in V(G) \setminus \psi(X \cup Y)$. It implies that the function $\alpha_h(q_1, \ldots, q_s)$ depends only on the values of $q_1, \ldots, q_s$.

We claim that computing $\alpha_r(p_1, \ldots, p_s)$ is equivalent to finding an extension φ of ψ of minimum weight such that $F = G \oplus_\varphi H$ is 2-connected.

Assume that $\alpha_r(p_1, \ldots, p_s) < +\infty$. Notice that $Z' = Z$ if $q_i = p_i$ for $i \in \{1, \ldots, s\}$. Let $\xi\colon Z \to (V(F_0) \setminus \psi(X \cup Y)) \cup V(F_1) \cup \ldots \cup V(F_h)$ be an injection that provides the minimum in (4), that is, $\alpha_r(p_1, \ldots, p_s) = \sum_{z \in Z} w(z\xi(z))$. We define $\varphi(z) = \xi(z)$ for $z \in Z$ and $\varphi(x) = \psi(x)$ for $x \in X \cup Y$. Clearly, φ is an extension of ψ. Because ξ is an injection, we have that φ is an injective mapping. Since ξ satisfies (a) and (b), we obtain that φ satisfies (i) and (ii) and, therefore, $F = G \oplus_\varphi H$ is 2-connected. Let $R = \sum_{xy \in E(H),\ x,y \in X \cup Y} \omega(\psi(x)\psi(y))$. Then using (3), we have that

$$\begin{aligned}
\omega(\varphi) &= \sum_{xy \in E(H)} \omega(\varphi(x)\varphi(y)) \\
&= \sum_{xy \in E(H),\ x,y \in X \cup Y} \omega(\varphi(x)\varphi(y)) + \sum_{xz \in E(H),\ x \in X, z \in Z} \omega(\varphi(x)\varphi(y)) \\
&= R + \sum_{z \in Z} w(z, \varphi(z)) = R + \sum_{z \in Z} w(z, \xi(z)) = R + \alpha_r(p_1, \ldots, p_s). \qquad (5)
\end{aligned}$$

Let $\varphi'\colon V(H) \to V(G)$ be an injection that extends ψ such that $F' = G \oplus_{\varphi'} H$ is 2-connected. We define $\xi'\colon Z \to (V(F_0) \setminus \psi(X \cup Y)) \cup V(F_1) \cup \ldots \cup V(F_h)$ by setting $\xi'(z) = \varphi'(z)$ for $z \in Z$. Since φ' is an injection, ξ' is also an injection. Because F' is 2-connected, φ satisfies (i) and (ii). This implies that ξ' satisfies (a) and (b). Therefore, $\sum_{z \in Z} w(z, \xi'(z)) \geq \alpha_r(p_1, \ldots, p_s)$. Similarly to (5), we have that $\omega(\varphi') = R + \sum_{z \in Z} w(z, \xi'(z)) \geq R + \alpha_r(p_1, \ldots, p_s)$. We conclude that φ is an extension φ of ψ of minimum weight such that $F = G \oplus_\varphi H$ is 2-connected.

Suppose that $\alpha_r(p_1, \ldots, p_s) = +\infty$. It implies that there is no injection $\xi\colon Z \to (V(F_0) \setminus \psi(X \cup Y)) \cup V(F_1) \cup \ldots \cup V(F_h)$ satisfying (a) and (b). But this immediately implies that there is no injective extension φ of ψ satisfying (i) and (ii). This completes the proof of the claim.

We use dynamic programming to compute α_h consequently for $h = 0, 1, \ldots, r$.

We start with computing $\alpha_0(q_1, \ldots, q_s)$ for each s-tuple $(q_1, \ldots, q_s)$. Notice that the conditions (a) and (b) are irrelevant in this case, because they concern only $h \geq 1$. We construct the auxiliary complete bipartite graph $\mathcal{G}_0$ with the bipartition $(V(F_0) \setminus \psi(X \cup Y), Z')$ of its vertex set and define the weight of each edge zv for $z \in Z'$ and $v \in V(F_0) \setminus \psi(X \cup Y)$ as $w(z, v)$. We find a matching M in $\mathcal{G}_0$ that saturates every vertex of Z' and has the minimum weight using the Hungarian algorithm [7, 11] in time $\mathcal{O}(|V(G)|^3 \cdot \log W)$. If there is no matching saturating Z', we set $\alpha_0(q_1, \ldots, q_s) = +\infty$. Otherwise, $\alpha_0(q_1, \ldots, q_s) = w(M)$. It is straightforward to verify the correctness of computing $\alpha_0(q_1, \ldots, q_s)$ by the definition of this function.

Assume that $h \geq 1$ and we already computed the table of values of $\alpha_{h-1}(q_1, \ldots, q_s)$. We explain how to construct the table of values of $\alpha_{h-1}(q_1, \ldots, q_s)$. The the computation is based on

the observation that we can see an injective mapping $\xi\colon Z' \to (V(F_0) \setminus \psi(X \cup Y)) \cup V(F_1) \cup \ldots \cup V(F_h)$ as the union of two injections $\xi'\colon Z'' \to (V(F_0)\setminus\psi(X\cup Y))\cup V(F_1)\cup\ldots\cup V(F_{h-1})$ and $\lambda\colon Z''' \to V(F_h)$ for the appropriate partition (Z'', Z''') of Z'.

For each s-tuple of integers $(q_1, \ldots, q_s)$ such that $0 \leq q_i \leq p_i$ for $i \in \{1, \ldots, s\}$, we define

$$\alpha'_h(q_1, \ldots, q_s) = \min_{\lambda} \sum_{z \in Z'} w(z, \xi(z)), \tag{6}$$

where $Z' \subseteq Z$ such that $|Z' \cap Z_i| = q_i$ for $i \in \{1, \ldots, s\}$ and the minimum is taken over all injective mappings $\lambda\colon Z' \to V(F_h)$ such that the following conditions are fulfilled:

(a*) there is $v \in V(F_h)$ such that $v \in \lambda(Z')$, and
(b*) if v is the unique element of $V(F_h) \cap \lambda(Z')$ and v is incident to a bridge vu of G, then there is $x \in X$ such that $\psi(x) \neq u$ and x is adjacent to $\lambda^{-1}(v)$ in H.

If such a mapping λ does not exist, then we assume that $\alpha'_h(q_1, \ldots, q_s) = +\infty$. As for $\alpha_h(q_1, \ldots, q_s)$, $\alpha'_h(q_1, \ldots, q_s)$ depends only on the values of $q_1, \ldots, q_s$, because if $x, y \in Z_i$, then $w(x, v) = w(y, v)$ for $v \in V(G) \setminus \psi(X \cup Y)$.

Let uv be the unique bridge of G with $v \in V(F_h)$. Suppose that for an s-tuple $(q_1, \ldots, q_s)$, we obtain that $|Z'| = 1$ and for the unique vertex $z \in Z'$, z has a unique neighbor $x \in X$ in H and $\psi(x) = u$. Then we set $\alpha'_h(q_1, \ldots, q_s) = +\infty$ if $|V(F_h)| = 1$ and $\alpha'_h(q_1, \ldots, q_s) = \min\{w(zv') \mid v' \in V(F_h) \setminus \{v\}\}$ otherwise. For other s-tuples $(q_1, \ldots, q_s)$, we compute $\alpha'_h(q_1, \ldots, q_s)$ as follows. We construct the auxiliary complete bipartite graph $\mathcal{G}_h$ with the bipartition $(V(F_h), Z')$ of its vertex set and define the weigh of each edge zv for $z \in Z'$ and $v \in V(F_0) \setminus \psi(X \cup Y)$ as $w(zv)$. We find a matching M in $\mathcal{G}_h$ that saturates every vertex of Z' and has the minimum weight using the Hungarian algorithm [7, 11] in time $\mathcal{O}(|V(G)|^3 \cdot \log W)$. If there is no matching saturating Z', we set $\alpha'_h(q_1, \ldots, q_s) = +\infty$. Otherwise, $\alpha'_h(q_1, \ldots, q_s) = w(M)$. It is again straightforward to verify the correctness of computing $\alpha'_h(q_1, \ldots, q_s)$ using the definition of this function.

Now, to compute $\alpha_h(q_1, \ldots, q_s)$, we use the equation:

$$\alpha_h(q_1, \ldots, q_s) = \min\{\alpha_{h-1}(q'_1, \ldots, q'_s) + \alpha'_h(q''_1, \ldots, q''_s)\}, \tag{7}$$

where the minimum is taken over all s-tuples $(q'_1, \ldots, q'_s)$ and $(q''_1, \ldots, q''_s)$ such that $q_i = q'_i + q''_i$ for $i \in \{1, \ldots, s\}$.

To evaluate the running time, observe that there are at most $|V(G)|^s$ s-tuples $(q_1, \ldots, q_s)$. Since $s \leq 2^t$, it implies that the table of values of $\alpha_0(q_1, \ldots, q_s)$ can be computed in time $|V(G)|^{\mathcal{O}(2^t)} \cdot \log W$. Similarly, the table of values of $\alpha'_h(q_1, \ldots, q_s)$ for each $h \in \{1, \ldots, r\}$ can be computed in the same time. To compute $\alpha_h(q_1, \ldots, q_s)$ for a given s-tuple $(q_1, \ldots, q_s)$ using (7), we have to consider at most $|V(G)|^s$ pairs of s-tuples $(q'_1, \ldots, q'_s)$ and $(q''_1, \ldots, q''_s)$. Hence, we can compute the table of values $\alpha_h(q_1, \ldots, q_s)$ from the tables of values of $\alpha_{h-1}(q_1, \ldots, q_s)$ and $\alpha'_h(q_1, \ldots, q_s)$ in time $|V(G)|^{\mathcal{O}(2^t)} \cdot \log W$ for each $h \in \{1, \ldots, r\}$. We conclude that the total running time is $|V(G)|^{\mathcal{O}(2^t)} \cdot \log W$. ◀

3.2 Hardness of structured augmentation

In this section we show that Theorems 5 and 7 are tight in the sense that if the vertex-cover number of graphs in a hereditary graph class $\mathcal{C}$ is unbounded, then both structured augmentation problems are NP-complete. Our hardness proof actually holds for for any k-edge connectivity augmentation. For a positive integer k, we define the following problem:

STRUCTURED k-CONNECTIVITY AUGMENTATION

Input: Graphs G and H such that G is edge $(k-1)$-connected, a weight function $\omega\colon \binom{V(G)}{2} \to \mathbb{N}_0$ and a nonnegative integer W.

Task: Decide whether there is an injective $\varphi\colon V(H) \to V(G)$ such that $F = G \oplus_\varphi H$ is edge k-connected and the *weight* of the mapping $\omega(\varphi) = \sum_{xy \in E(H)} \omega(\varphi(x)\varphi(y)) \leq W$.

Let us note that for $k = 1$ this is STRUCTURED CONNECTIVITY AUGMENTATION and for $k = 2$ this is STRUCTURED 2-CONNECTIVITY AUGMENTATION.

▶ **Theorem 8.** *Let k be a positive integer. Let also $\mathcal{C}$ be a hereditary graph class. Then if the vertex-cover number of $\mathcal{C}$ is unbounded, then* STRUCTURED k-CONNECTIVITY AUGMENTATION *is* NP*-complete for $H \in \mathcal{C}$ in the strong sense.*

Also we observe that it is unlikely that we can avoid the dependency on t in the exponents of polynomial bounding the running time when solving STRUCTURED k-CONNECTIVITY AUGMENTATION for H with $\beta(H) \leq t$.

▶ **Proposition 9.** *For every positive integer k,* STRUCTURED k-CONNECTIVITY AUGMENTATION *is* W[1]*-hard when parameterized by $\beta(H)$ even if the weight of every pair of vertices of G is restricted to be ether 0 or 1.*

This proposition implies that unless FPT =W[1], we cannot solve STRUCTURED k-CONNECTIVITY AUGMENTATION for $k = 1, 2$ in time $f(\beta(H)) \cdot |V(G)|^{\mathcal{O}(1)}$. Hence the running time of the form $|V(G)|^{f(t)}$ of algorithms solving STRUCTURED k-CONNECTIVITY AUGMENTATION for graphs H with $\beta(H) \leq t$ is probably unavoidable.

4 Augmenting unweighted graphs

In this section we investigate unweighted STRUCTURED CONNECTIVITY AUGMENTATION and STRUCTURED 2-CONNECTIVITY AUGMENTATION. Let us remind that in the unweighted cases of the structured augmentation problems the task is to identify whether there is a superposition of graphs G and H of edge connectivity 1 or 2, correspondingly. In other words, we have the weight $\omega(uv) = 0$ for every pair of vertices of G and $W = 0$. We obtain structural characterizations of yes-instances for both problems.

For STRUCTURED CONNECTIVITY AUGMENTATION, we show the following theorem.

▶ **Theorem 10.** *Let G and H be graphs such that H has no isolated vertices and $|V(H)| \leq |V(G)|$. Then there is an injective mapping $\varphi\colon V(H) \to V(G)$ such that $F = G \oplus_\varphi H$ is connected if and only if $c(G) \leq |V(H)| - c(H) + 1$ and one of the following holds:*

(i) *H is connected,*

(ii) *H is disconnected graph and $i(G) \leq |V(H)| - c(H)$.*

Now we consider the case STRUCTURED 2-CONNECTIVITY AUGMENTATION.

▶ **Theorem 11.** *Let G and H be graphs such that G is connected, H has no isolated vertices and $|V(H)| \leq |V(G)|$. Then there is an injective mapping $\varphi\colon V(H) \to V(G)$ such that $F = G \oplus_\varphi H$ is 2-connected if and only if one of the following holds:*

(i) *G is 2-connected,*

(ii) *G is not 2-connected and $p(G) \leq |V(H)|$,*

unless G is a star $K_{1,n}$ where n is odd and H is a matching graph.

Theorems 10 and 11 immediately imply the next corollary.

▶ **Corollary 12.** *Unweighted* Structured 1-Connectivity Augmentation *and* Structured 2-Connectivity Augmentation *are solvable in time* $\mathcal{O}(|V(G)|+|E(G)|+|E(H)|)$.

5 Conclusion

We initiated the investigation of the structured connectivity augmentation problems where the aim is to increase the edge connectivity of the input graphs by adding edges when the added edges compose a given graph. In particular, we proved that Structured Connectivity Augmentation and Structured 2-Connectivity Augmentation are solvable in polynomial time when H is from a graph class $\mathcal{C}$ with bounded vertex-cover number. It is natural to ask about increasing connectivity of a $(k-1)$-connected graph to a k-connected graph for every positive integer k. For the "traditional" edge connectivity augmentation problem (see [6, 12]), the augmentation algorithms are based on the classic work of Dinits, Karzanov, and Lomonosov [1] about the structure of minimum edge separators. However, for the structural augmentation, the structure of the graph H is an obstacle for implementing this approach directly. Due to this, we could not push further our approach to establish the complexity of Structured k-Connectivity Augmentation for $k>2$ when H is of bounded vertex cover. This remains a natural open question. Recall that our hardness results showing that it is NP-hard to increase the connectivity of a $(k-1)$-connected graph to a k-connected graph when H belongs to a class with unbounded vertex cover number are proved for every k.

As the first step, it could be interesting to consider the variant of the problem for multigraphs. In this case, we allow parallel edges and assume that for a mapping $\phi\colon V(H)\to V(G)$, the multiplicity of $\phi(x)\phi(y)$ in $G\oplus_\phi H$ is the sum of the multiplicities of $\phi(x)\phi(y)$ in G and xy in H. Notice that all our algorithmic and hardness results can be restated for this variant of the problem. Actually, some of the proofs for this variant of the problem become even simpler.

The question of obtaining a k-connected graph for $k\geq 3$ is also open for the unweighted problem. Here we ask whether it is possible to derive structural necessary and sufficient conditions for a $(k-1)$-connected graph G and a graph H such that there exists an injective mapping $\phi\colon V(H)\to V(G)$ such that $G\oplus_\phi H$ is k-connected.

Another direction of the research is to consider vertex connectivity. As it is indicated by the existing results about vertex connectivity augmentation (see, e.g., [8, 9]), this variant of the problem could be more complicated.

References

1 E. A. Dinic, A. V. Karzanov, and M. V. Lomonosov. The structure of a system of minimal edge cuts of a graph. In *Studies in discrete optimization (Russian)*, pages 290–306. Izdat. "Nauka", Moscow, 1976.

2 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

3 K. Eswaran and R. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976. doi:10.1137/0205044.

4 Petr A. Golovach Fedor V. Fomin and Dimitrios M. Thilikos. Structured connectivity augmentation. *CoRR*, abs/1706.04255, 2017. URL: http://arxiv.org/abs/1706.04255.

5 András Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM J. Discrete Math.*, 5(1):25–53, 1992. doi:10.1137/0405003.

6 András Frank. *Connections in combinatorial optimization*, volume 38 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2011.

7 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

8 Bill Jackson and Tibor Jordán. Independence free graphs and vertex connectivity augmentation. *J. Comb. Theory, Ser. B*, 94(1):31–77, 2005. doi:10.1016/j.jctb.2004.01.004.

9 Tibor Jordán. On the optimal vertex-connectivity augmentation. *J. Comb. Theory, Ser. B*, 63(1):8–20, 1995. doi:10.1006/jctb.1995.1002.

10 Tibor Jordán and Zoltán Szigeti. Detachments preserving local edge-connectivity of graphs. *SIAM J. Discrete Math.*, 17(1):72–87, 2003. doi:10.1137/S0895480199363933.

11 H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955. doi:10.1002/nav.3800020109.

12 Hiroshi Nagamochi and Toshihide Ibaraki. *Algorithmic aspects of graph connectivity*, volume 123 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2008. doi:10.1017/CBO9780511721649.

13 Ján Plesník. Minimum block containing a given graph. *Arch. Math. (Basel)*, 27(6):668–672, 1976. doi:10.1007/BF01224737.

14 Toshimasa Watanabe and Akira Nakamura. Edge-connectivity augmentation problems. *Journal of Computer and System Sciences*, 35(1):96–144, 1987. doi:10.1016/0022-0000(87)90038-9.

Combinatorial Properties and Recognition of Unit Square Visibility Graphs*

Katrin Casel[1], Henning Fernau[2], Alexander Grigoriev[3], Markus L. Schmid[4], and Sue Whitesides[5]

1 **Fachbereich 4 – Abt. Informatikwissenschaften, Universität Trier, Germany**
casel@uni-trier.de
2 **Fachbereich 4 – Abt. Informatikwissenschaften, Universität Trier, Germany**
fernau@uni-trier.de
3 **School of Business and Economics, Maastricht University, The Netherlands**
a.grigoriev@maastrichtuniversity.nl
4 **Fachbereich 4 – Abt. Informatikwissenschaften, Universität Trier, Germany**
mschmid@uni-trier.de
5 **Department of Computer Science, University of Victoria, BC, Canada**
sue@uvic.ca

Abstract

Unit square (grid) visibility graphs (USV and USGV, resp.) are described by axis-parallel visibility between unit squares placed (on integer grid coordinates) in the plane. We investigate combinatorial properties of these graph classes and the hardness of variants of the recognition problem, i. e., the problem of representing USGV with fixed visibilities within small area and, for USV, the general recognition problem.

1998 ACM Subject Classification F.2.2 Computations on discrete structures

Keywords and phrases Visibility graphs, visibility layout, NP-completeness, exact algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.30

1 Introduction

A visibility representation of a graph G is a set $\mathcal{R} = \{R_i \mid 1 \leq i \leq n\}$ of geometric objects (e. g., bars, rectangles, etc.) along with some kind of geometric visibility relation $\sim$ over $\mathcal{R}$ (e. g., axis-parallel visibility), such that $G = (\{v_i \mid 1 \leq i \leq n\}, \{\{v_i, v_j\} \mid R_i \sim R_j\})$. In this work, we focus on rectangle visibility graphs, which are represented by axis aligned rectangles in the plane and vertical and horizontal axis parallel visibility between them. In particular, we consider the more restricted variant of *unit square visibility graphs* (see [12]), and, in addition, we consider the case where the unit squares are placed on an integer grid (an alternative characterisation of the well-known class of graphs with rectilinear drawings).

The study of visibility representations is of interest, both for applications and for graph classes, and has remained an active research area[1] mainly because axis-aligned visibilities give rise to graph and network visualizations that satisfy good readability criteria: straight

* We acknowledge the support of the first author by the Deutsche Forschungsgemeinschaft, grant FE 560/6-1, and the support of the last author by the NSERC Discovery Grant program of Canada.

[1] The 24th International Symposium on Graph Drawing and Network Visualization (GD 2016) featured an entire session on visibility representation (see [3, 10, 11, 24]), and the joint workshop day of the Symposium on Computational Geometry (SoCG) and the ACM Symposium on Theory of Computing (STOC) included a workshop on geometric representations of graphs in 2016.

edges, and edges that cross only at right angles. These properties are highly desirable in the design of layouts of circuits and communication paths. Indeed, the study of graphs arising from vertical visibilities among disjoint, horizontal line segments ("bars") in the plane originated during the 1980's in the context of VLSI design problems; see [16, 30, 29].

Because bar visibility graphs are necessarily planar, this model has been extended in various ways in order to represent larger classes of graphs. Such extensions include new definitions of visibility (e. g., sight lines that may penetrate up to k bars [13] or other geometric objects [4]), vertex representations by other objects (e. g., rectangles, L-shapes [18], and sets of up to t bars [23]), extensions to higher dimensional objects (see, e. g., [8] for visibility representation in 3D by axis aligned horizontal rectangles with vertical visibilities, or [19], which studies visibility representations by unit squares floating parallel to the x, y-plane and lines of sight that are parallel to the z axis). The desire for polysemy, that is, the expression of more than one graph by means of one underlying set of objects, has also provided impetus in the study of visibility representations (see for example [6] and [28]).

Rectangle visibility graphs have the attractive property, for visualization purposes, that they yield right angle crossing drawings (RAC graphs (see [15]), whose edges are drawn as sequences of horizontal and vertical segments forming a polyline with orthogonal bends), which have seen considerable interest in the graph drawing community. Unit square graphs form a subfamily of L-visibility graphs (see [18]) and their grid variant a subfamily of RACs with no bends (note that RAC recognition for 0-bends is NP-hard [2]).

Using visibilities among objects is but one example of the use of binary geometric relations for this purpose; other geometric relations include intersection relations (e.g., of strings or straight line segments in the plane, of boxes in arbitrary dimension), proximity relations (e.g., of points in the plane), and contact relations. In the literature, for the resulting graph classes, combinatorial aspects, relationships to other graph classes, as well as computational aspects are studied (see [20] for a survey focusing on contact representations of rectangles).

Finally, we note that visibility properties among sets of objects have been studied in a number of contexts, including motion planning and computer graphics. In [26] it is proposed to find shortest paths for mobile robots moving in a cluttered environment by looking for shortest paths in the visibility graph of the points located at the vertices of polygonal obstacles. This led to a search for fast algorithms to compute visibility graphs of polygons, as well as to a search for finding shortest paths without computing the entire visibility graph.

We extend the known combinatorial properties of unit square visibility graphs from [12], and proof their recognition problem to be NP-hard (this requires a reduction that is highly non-trivial on a technical level with the main difficulty to identify graph structures that can be shown to be representable by unit square layouts in a unique way to gain sufficient control for designing suitable gadgets). With respect to unit square *grid* visibility graphs, we extend known combinatorial properties and consider variants of its recognition problem.

Due to space constraints, we only provide proof sketches (details can be found in [9]).

2 Preliminaries

A *visibility layout*, or simply *layout*, is a set $\mathcal{R} = \{R_i \mid 1 \leq i \leq n\}$ with $n \in \mathbb{N}$, where R_i are closed and pairwise disjoint axis-parallel rectangles in the plane; the *position* of such a rectangle is the coordinate of its lower left corner. For every $R_i, R_j \in \mathcal{R}$, a closed non-degenerate axis-parallel rectangle S (i. e., a non-empty closed rectangle that is not a line segment) is a *visibility rectangle for R_i and R_j* if one side of S is contained in R_i and the opposite side in R_j. We define $R_i \rightarrow_{\mathcal{R}} R_j$ ($R_i \downarrow_{\mathcal{R}} R_j$), if there is a visibility rectangle S for R_i

and R_j, such that the left side (upper side) of S is contained in R_i, the right side (lower side) of S is contained in R_j and $S \cap R_k = \emptyset$, for every $R_k \in \mathcal{R} \setminus \{R_i, R_j\}$. Let $\leftrightarrow_\mathcal{R}$ and $\updownarrow_\mathcal{R}$ be the symmetric closures of $\rightarrow_\mathcal{R}$ and $\downarrow_\mathcal{R}$, respectively. Finally, $R_i \sim_\mathcal{R} R_j$ if $R_i \leftrightarrow_\mathcal{R} R_j$ or $R_i \updownarrow_\mathcal{R} R_j$ ($\sim_\mathcal{R}$ is the *visibility relation* (*with respect to* $\mathcal{R}$)). If the layout $\mathcal{R}$ is clear from the context or negligible, we drop the subscript $\mathcal{R}$. We denote $R_i \sim R_j$, $R_i \leftrightarrow R_j$ and $R_i \rightarrow R_j$ also as R_i *sees* R_j, R_i *horizontally sees* R_j and R_i *sees* R_j *from the left*, respectively, and analogous terminology applies to vertical visibilities. For $S, T \subseteq \mathcal{R}$, we use $S \rightarrow_\mathcal{R} T$ as shorthand form for $\bigwedge_{R \in S, R' \in T} R \rightarrow_\mathcal{R} R'$.

A layout $\mathcal{R} = \{R_i \mid 1 \leq i \leq n\}$ *represents* the undirected graph $\mathsf{G}(\mathcal{R}) = (\{v_i \mid 1 \leq i \leq n\}, \{\{v_i, v_j\} \mid 1 \leq i, j \leq n, R_i \sim R_j\})$, which is then called a *visibility graph*, and the class of visibility graphs is denoted by V. A graph is a *weak* visibility graph, if it can be obtained from a visibility graph by deleting some edges and the corresponding class of graphs is denoted by $\mathsf{V_w}$. As a convention, for a visibility graph $G = (V, E)$ and a layout representing it we denote by R_v the rectangle for $v \in V$ and define $R_{V'} = \{R_x \mid x \in V'\}$ for every $V' \subseteq V$. We call layouts $\mathcal{R}_1$ and $\mathcal{R}_2$ *isomorphic* if $\mathsf{G}(\mathcal{R}_1)$ and $\mathsf{G}(\mathcal{R}_2)$ are isomorphic. Furthermore, we call $\mathcal{R}_1$ and $\mathcal{R}_2$ *V-isomorphic* if, for some $x \in \{\rightarrow_{\mathcal{R}_1}, \rightarrow_{\mathcal{R}_1}^{-1}\}$ and $y \in \{\downarrow_{\mathcal{R}_1}, \downarrow_{\mathcal{R}_1}^{-1}\}$, the relational structure $(\mathcal{R}_1, \rightarrow_{\mathcal{R}_1}, \downarrow_{\mathcal{R}_1})$ is isomorphic to $(\mathcal{R}_2, x, y)$ or $(\mathcal{R}_2, y, x)$.[2]

Unit square visibility graphs (USV) and *unit square grid visibility graphs* (USGV) are represented by *unit square layouts*, where every $R \in \mathcal{R}$ is the unit square, and *unit square grid layouts*, where additionally the position of every R is from $\mathbb{N} \times \mathbb{N}$.[3] The weak classes $\mathsf{USV_w}$ and $\mathsf{USGV_w}$ are defined accordingly.

For a graph $G = (V, E)$, $N(v)$ is the *neighbourhood* of $v \in V$, $\vec{E}$ denotes an oriented version of E, i. e., $E = \{\{u, v\} \mid (u, v) \in \vec{E}\}$, and $f \colon \vec{E} \to E, (u, v) \mapsto \{u, v\}$ is a bijection. Let L, R and D, U be pairs of complementary values (for $X \in \{\mathsf{L}, \mathsf{R}, \mathsf{D}, \mathsf{U}\}$, $\overline{X}$ denotes its complement). An LRDU-*restriction* (for G) is a labeling $\sigma \colon \vec{E} \to \{\mathsf{L}, \mathsf{R}, \mathsf{D}, \mathsf{U}\}$ and it is *valid* if, for every $(u, v) \in \vec{E}$ with $\sigma((u, v)) = X$ and every $w \in V \setminus \{u, v\}$, $\sigma((u, w)) \neq X \neq \sigma((w, v))$ and $\sigma((v, w)) \neq \overline{X} \neq \sigma((w, u))$. Obviously, LRDU-restrictions only exist for graphs with maximum degree 4. A unit square grid visibility layout *satisfies* an LRDU-restriction σ if $\sigma((u, v)) = \mathsf{L}$ implies $R_v \rightarrow R_u$, $\sigma((u, v)) = \mathsf{R}$ implies $R_u \rightarrow R_v$, $\sigma((u, v)) = \mathsf{D}$ implies $R_u \downarrow R_v$ and $\sigma((u, v)) = \mathsf{U}$ implies $R_v \downarrow R_u$. An HV-*restriction* (for G) is a labeling $\sigma \colon E \to \{\mathsf{H}, \mathsf{V}\}$ and it is *valid* if, for every $u \in V$ at most two incident edges are labeled H and at most two incident edges are labeled V. A unit square grid visibility layout *satisfies* an HV-restriction σ if $\sigma(\{u, v\}) = \mathsf{H}$ implies $R_v \leftrightarrow R_u$ and $\sigma(\{u, v\}) = \mathsf{V}$ implies $R_v \updownarrow R_u$.

For a class $\mathfrak{G}$ of undirected graphs, the *recognition problem for* $\mathfrak{G}$ (denoted by $\textsc{Rec}(\mathfrak{G})$) for short) is the problem to decide, for a given undirected graph G, whether or not $G \in \mathfrak{G}$. In the following, we shall consider the problems $\textsc{Rec}(\mathsf{USGV})$ and $\textsc{Rec}(\mathsf{USV})$.

We briefly recall some established geometric graph representations relevant to this work. A *rectilinear drawing* (see [17, 25]) of a graph $G = (V, E)$ is a pair of mappings $x, y \colon V \to \mathbb{Z}$, where, for every $v \in V$, $x(v)$ and $y(v)$ represent the x- and y-coordinates of v on the grid and, for every edge $\{u, v\} \in E$, $(x(u), y(u))$ and $(x(v), y(v))$ are the endpoints of a horizontal or vertical line segment that does not contain any $(x(w), y(w))$ with $w \in V \setminus \{u, v\}$. A graph has *resolution* $\frac{2\pi}{d}$ if it has a drawing in which the degree of the angle between any two edges incident to a common vertex is at least $\frac{2\pi}{d}$. We call such graphs *resolution-*$\frac{2\pi}{d}$ *graphs* and are mainly interested in the case $d = 4$, see [21]. For planar graphs, resolution-$\frac{2\pi}{4}$ graphs are

[2] By $\preceq^{-1}$, we denote the inverse of a binary relation $\preceq$.

[3] Note that in the grid case, if a unit square is positioned at (x, y), then this is the only unit square on coordinates (x', y'), $x' \in \{x - 1, x, x + 1\}$, $y' \in \{y - 1, y, y + 1\}$.

just rectilinear graphs, see [7]. A *bendless right angle crossing* (BRAC) *drawing* of a graph is a straight-line drawing in which every crossing of two edges is at right angles.[4] Note that in a BRAC-drawing or a resolution-$\frac{2\pi}{4}$ drawing, edges are not necessarily axis-parallel (like it is the case for visibility layouts and rectilinear drawings). A graph is called *rectilinear* or *BRAC graph* if it has a rectilinear or BRAC-drawing, respectively.

3 Unit Square Grid Visibility Graphs

The readability of graph drawings is mainly affected by its *angular resolution* (angles formed by consecutive edges incident to a common node) and its *crossing resolution* (angles formed at edge crossings); see the discussion in [1]. In this regard, resolution-$\frac{\pi}{2}$ graphs and BRAC graphs have an angular resolution and crossing resolution of $\frac{\pi}{2}$, respectively, while rectilinear drawings and unit square grid visibility layouts force both resolutions to be $\frac{\pi}{2}$.

The question arises of how these classes relate to each other and in this regard, we first note that USGV and rectilinear graphs coincide. More precisely, a unit square grid layout can be transformed into a rectilinear drawing by replacing every unit square on position (x, y) by a vertex on position (x, y) and translate the former visibilities into straight-line segments. Transforming a rectilinear drawing into a unit square grid layout requires scaling it first by factor 2 and then replacing each vertex on position (x, y) by a unit square on position (x, y) (without scaling, sides or corners of unit squares may overlap). This only results in a *weak* layout, since visibilities may be created that do not correspond to edges in the rectilinear drawing. However, any weak unit square grid visibility graph can be transformed into a unit square grid visibility graph (as formally stated below in Theorem 7).

Since all these graphs except the BRAC graphs have maximum degree 4, we only consider degree-4 BRAC graphs. Obviously, resolution-$\frac{\pi}{2}$ graphs and degree-4 BRAC graphs are both superclasses of USGV (and rectilinear graphs). Witnessed by K_3, the inclusion in degree-4 BRAC graphs is proper, while the analogous question w. r. t. resolution-$\frac{\pi}{2}$ graphs is open. Moreover, K_3 is also an example of a degree-4 BRAC graph that is not a resolution-$\frac{\pi}{2}$ graph; whether there exist resolution-$\frac{\pi}{2}$ graphs without a BRAC-drawing is open.

Due to the equivalence of USGV and rectilinear graphs, results for the latter graph class carry over to the former. In this regard, we first mention that the NP-hardness proof of recognizing resolution-$\frac{\pi}{2}$ graphs from [21] actually produces drawings with axis-aligned edges; thus, it also applies to rectilinear graphs (a similar reduction (for rectilinear graphs and presented in more detail) is provided in [17]). As shown in [17], the recognition problem for rectilinear graphs can be solved in time $O(24^k \cdot k^{2k} \cdot n)$, where k is the number of vertices with degree at least 3. In [25], it is shown that recognition remains NP-hard if we ask whether a drawing exists that satisfies a given HV-restriction[5] or a drawing that satisfies a given circular order of incident edges. However, checking the existence of a rectilinear drawing satisfying a given LRDU-restriction can be done in time $O(|E| \cdot |V|)$. Consequently, by trying all such labellings, we can solve the recognition problem for rectilinear graphs in time $2^{O(n)}$. In this regard, it is worth noting that the hardness reduction from [17] can be easily modified, such that it also provides lower complexity bounds subject to the Exponential-Time Hypothesis (ETH), thereby demonstrating that the $2^{O(n)}$ algorithm is optimal subject to ETH.

[4] In the literature (e. g., [15]), the edges of a RAC-drawing are usually allowed to have bends; the investigated questions are on finding RAC-drawings that minimise the number of bends and crossings.
[5] The definition of HV- and LRDU-restriction given above naturally extends to rectilinear drawings.

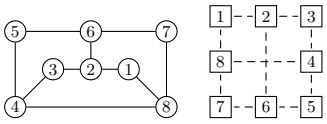


Figure 1 Necessarily non-planar visibility layout for a planar graph.

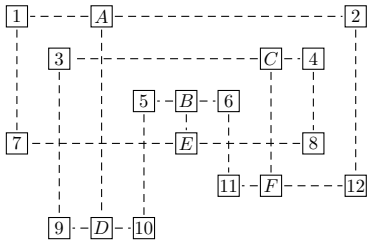


Figure 2 Subdivisions of $K_{3,3}$.

3.1 Combinatorial Properties of USGV

First, we shall see that the class USGV is downward closed w. r. t. the subgraph relation, i. e., if $G \in \mathsf{USGV}$, then all its subgraphs are in USGV (intuitively speaking, deletion of edges can be done by moving unit squares, while deletion of a vertex can be realised by deleting the corresponding unit square and then removing unwanted edges introduced by this operation). This observation will be a convenient tool for obtaining other combinatorial results.

▶ **Lemma 1.** *Let $G = (V, E) \in \mathsf{USGV}$, let $v \in V$ and $e \in E$. Then $(V, E \setminus \{e\}) \in \mathsf{USGV}$ and $(V \setminus \{v\}, E) \in \mathsf{USGV}$.*

It is straightforward to prove the following limitations of USGV.

▶ **Lemma 2.** *Let $G = (V, E) \in \mathsf{USGV}$. Then, (**1**) the maximum degree of G is 4, (**2**) for every $u, v \in V$, $|N(u) \cap N(v)| \leq 2$, and, (**3**) for every $\{u, v\} \in E$, $N(u) \cap N(v) = \emptyset$.*

A consequence of Lemma 2 is that no graph from USGV contains $K_{1,5}$, $K_{2,3}$ or K_3 as a subgraph, since they violate the first, second and third condition of Lemma 2, respectively. Obvious examples for graphs from USGV are subgraphs of a grid; as Lemma 1 shows, even non-induced subgraphs of a grid. In this context, notice that the problem of deciding if a given graph is such a *partial grid graph* is equivalent to deciding if it admits a unit-length VLSI layout, which, even restricted to trees, is an NP-hard problem; see [5] for details. Yet, USGV contains more, especially non-bipartite graphs, with the smallest example being C_5.

Next, we discuss planarity with a focus on the relationship between the planarity of graphs from USGV and planarity of their respective layouts (where a layout is called *planar* if it does not contain any crossing visibilities). In this regard, we first note that the planarity of a layout is obviously sufficient for the planarity of the represented graph. Moreover, it is trivial to construct non-planar layouts that nevertheless represent planar graphs. Figure 1(a) is an example of a planar unit square grid visibility graph, which can only be represented by non-planar layouts (e. g., the one of Figure 1(b)):

▶ **Proposition 3.** *There exists no planar unit square grid layout for the graph of Fig. 1(a).*

It is tempting to assume that graphs in USGV are necessarily planar, but, as demonstrated by Figure 2, USGV contains a subdivision of $K_{3,3}$. Hence, with Kuratowski's theorem, we conclude:

▶ **Theorem 4.** USGV *contains non-planar graphs.*

Next, we investigate possibilities to characterise USGV. In this regard, we first observe that a characterisation by forbidden induced subgraphs is not possible (note that under the assumption P ≠ NP, this also follows from the hardness of recognition).

▶ **Theorem 5.** USGV *does not admit a characterisation by a finite number of forbidden induced subgraphs.*

By Lemma 2, the classes of cycles, complete graphs and complete bipartite graphs within USGV are easily characterised: $C_i \in \mathsf{USGV}$ if and only if $i \geq 4$, $K_i \in \mathsf{USGV}$ if and only if $i \leq 2$, $K_{i,j} \in \mathsf{USGV}$ (with $i \leq j$) if and only if ($i = 1$ and $j \leq 4$) or ($i = 2$ and $j = 2$). Furthermore, the trees in USGV have a simple characterisation as well:

▶ **Theorem 6.** *A tree T is in* USGV *if and only if the maximum degree of T is at most four.*

By definition, $\mathsf{USGV} \subseteq \mathsf{USGV_w}$ and every $G' \in \mathsf{USGV_w}$ can be obtained from some $G \in \mathsf{USGV}$ by deleting some edges. Consequently, by Lemma 1, we conclude the following.

▶ **Theorem 7.** $\mathsf{USGV} = \mathsf{USGV_w}$.

3.2 Area-Minimisation

The *area-minimisation* version of the recognition problem is to decide whether a given graph has a drawing or layout of given width and height. The hardness of recognition for USGV and also for HV-restricted USGV carries over to the area-minimisation version, since an n-vertex graph has a layout if and only if it has a $(2n-1) \times (2n-1)$ layout. On the other hand, in the LRDU-restricted rectilinear (or unit square grid) case, recognition can be solved in polynomial time, so the authors of [25] provide a hardness reduction that proves the area-minimisation recognition problem NP-complete even for LRDU-restricted rectilinear graphs. However, this construction does not carry over to USGV, since the non-edges of a rectilinear drawing translate into non-visibilities, which require space as well;[6] moreover, it does not even work for the weak case of USGV, due to the necessary scaling by factor 2 to translate a rectilinear drawing into an equivalent weak unit square grid layout.

Next, we provide a reduction that shows the hardness of the area-minimisation version of Rec($\mathsf{USGV_w}$), which shall also imply several additional results. The problem 3-Partition (3Part) is defined as follows: Given $B \in \mathbb{N}$ and a multi-set $A = \{a_1, a_2, \ldots, a_{3m}\} \subseteq \mathbb{N}$ with $\frac{B}{4} < a_i < \frac{B}{2}$, $1 \leq i \leq 3m$, and $\sum_{i=1}^{3m} a_i = mB$, decide whether A can be partitioned into m multi-sets $A_1, \ldots, A_m$, such that $\sum_{a \in A_j} a = B$, $1 \leq j \leq m$ (note that the restriction $\frac{B}{4} < a_i < \frac{B}{2}$ enforces $|A_j| = 3$, $1 \leq j \leq m$). Given a 3Part instance, we construct a *frame graph* (see Figure 3) $G_f = (V_f, E_f)$ with:

$$
\begin{aligned}
V_f =& \{u_{i,j}, v_{i,j}, w_{i,1}, w_{i,2} \mid 1 \leq i \leq m, 0 \leq j \leq B\} \cup \{u_{m+1,0}, v_{m+1,0}, w_{m+1,1}, w_{m+1,2}\}, \\
E_f =& \{\{u_{i,j}, u_{i,j+1}\}, \{v_{i,j}, v_{i,j+1}\} \mid 1 \leq i \leq m, 0 \leq j \leq B-1\} \cup \\
& \{\{u_{i,B}, u_{i+1,0}\}, \{v_{i,B}, v_{i+1,0}\} \mid 1 \leq i \leq m\} \cup \{\{u_{i,j}, v_{i,j}\} \mid 1 \leq i \leq m, 1 \leq j \leq B\} \cup \\
& \{\{u_{i,0}, v_{i,0}\}, \{v_{i,0}, w_{i,1}\}, \{w_{i,1}, w_{i,2}\} \mid 1 \leq i \leq m+1\}.
\end{aligned}
$$

Next, we define a graph $G_A = (V_A, E_A)$ with $V_A = \{b_{i,j}, c_{i,j} \mid 1 \leq i \leq 3m, 1 \leq j \leq a_i\}$ and $E_A = \{\{b_{i,j}, b_{i,j+1}\}, \{c_{i,j}, c_{i,j+1}\} \mid 1 \leq i \leq 3m, 1 \leq j \leq a_i - 1\} \cup \{\{b_{i,j}, c_{i,j}\} \mid 1 \leq i \leq 3m, 1 \leq j \leq a_i\}$. Finally, we let $G = (V, E)$ with $V = V_f \cup V_A$ and $E = E_f \cup E_A$.

[6] In general, this space blow-up cannot be avoided, as witnessed by n isolated vertices which have a $1 \times n$ rectilinear drawing, but a smallest unit square grid layout of size $(2n-1) \times (2n-1)$

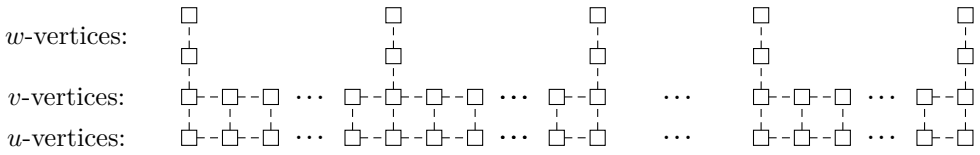


Figure 3 Unit square grid layout for the graph G_f.

The idea is that G_f forms m size-B compartments and the graphs on $b_{i,j}$, $c_{i,j}$ represent the a_i. In a layout respecting the size bounds, the way of allocating the graphs on $b_{i,j}$, $c_{i,j}$ to the compartments corresponds to a partition of A that is a solution for the 3Part-instance.

▶ **Lemma 8.** *(B, A) is a positive* 3Part*-instance if and only if G has a $(7\times(2(mB+m+1)-1))$ unit square grid layout.*

Since the reduction defined above is polynomial in m and B, and 3Part is strongly NP-complete (see [22, Theorem 4.4]), we can conclude the following:

▶ **Theorem 9.** *The area-minimisation variant of* Rec(USGV$_\mathsf{w}$) *is* NP*-complete.*

The area minimisation variant implicitly solves the general recognition problem, so the question arises whether it is also hard to decide if a graph from USGV$_\mathsf{w}$ (given as a layout) can be represented by a layout satisfying given size bounds. Since our reduction always produces a graph in USGV$_\mathsf{w}$ (with an obvious layout), independent of the 3Part-instance, it shows that the hardness remains if the input graph is given directly as a layout. Moreover, the problem is still NP-complete for the LRDU-restricted variant (the LRDU-restriction then simply enforces the structure shown in Figure 3).

The reduction also yields a (substantially simpler) alternative proof for the hardness of the area-minimisation recognition problem for LRDU-restricted rectilinear graphs [25] (more precisely, it can be shown that (B, A) is a positive 3Part-instance if and only if G has a $(4 \times (mB + m + 1))$ rectilinear drawing), and the hardness also carries over to the variant where the input graph is already given as a rectilinear drawing.

We conclude this section by pointing out that it is open whether the LRDU-restricted area-minimisation variant of Rec(USGV) can be solved in polynomial-time. Intuitively, reducing the size of a rectilinear drawing is difficult, since space can be saved by placing non-adjacent vertices on the same line, which is not possible for *non-weak* unit square grid layouts. However, computing a size-minimal unit square grid layout includes finding out to what extend the scaling by 2 is really necessary, which seems difficult as well.

4 Unit Square Visibility Graphs

Obviously, a larger class of graphs can be represented if the unit squares are not restricted to integer coordinates (see Figure 4 for some examples). In [12], cycles, complete graphs, complete bipartite graphs and trees in USV are characterised as follows: $C_i \in$ USV, for every $i \in \mathbb{N}$, $K_i \in$ USV if and only if $i \leq 4$, $K_{i,j} \in$ USV with $i \leq j$ if and only if $(1 \leq i \leq 2$ and $i \leq j \leq 6)$ or $(i = 3$ and $3 \leq j \leq 4)$,[7] and a tree T is in USV if and only if it is the union of two subdivided caterpillar forests with maximum degree 3 (note that [23] provides an algorithm that efficiently checks this property).

[7] For the more general question of representing bipartite graphs as rectangle visibility graphs, we refer to [14]. In particular, a linear upper bound on the number of edges, compared to the number of vertices, is known.

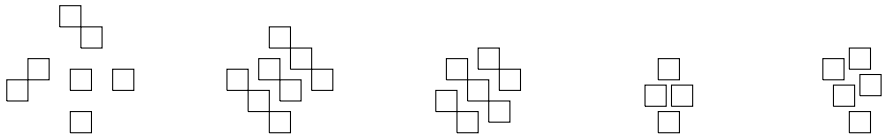


Figure 4 Visibility layouts for $K_{1,6}$, $K_{2,6}$, $K_{3,4}$, K_4 and a K_5 with one missing edge.

Next, we observe that every graph with at most 4 vertices is in USV, while K_5 is not (it is not hard to find layouts for graphs with at most 4 vertices; $K_5 \notin$ USV is shown in [12]).

▶ **Proposition 10.** *Every graph with at most* 4 *vertices is in* USV*.*

A crucial difference between USGV and USV is that for the latter, the degree is not bounded, as witnessed by layouts of the following form: □□□□□□□. However, if a unit square sees at least 7 other unit squares, then these must be placed in such a way that visibilities or "paths" between some of them are enforced (note that any $K_{1,n}$ may exist as induced subgraph, as can be demonstrated by modifying the above example layout such that between each two consecutive neighbours another "visibility-blocking" unit square is inserted). In [12], it is formally proven that in graphs from USV any vertex of degree at least 7 must lie on a cycle. In particular, these observations point out that an analogue of Lemma 1 is not possible for USV.

For the class of trees within USV, as long as we consider trees with maximum degree strictly less or larger than 6, a much simpler characterisation (compared to the one mentioned at the beginning of this section) applies:

▶ **Theorem 11.** *Let T be a tree with maximum degree k. If $k \leq 5$, then $T \in$* USV*, and if $k \geq 7$, then $T \notin$* USV*.*

Figure 5(a) shows an example of a tree from USV with maximum degree 6 and Figure 5(b) its representing layout. It can be easily verified that any node of degree 6 must be represented V-isomorphically to Figure 4(a) (note that this also holds for nodes A and B in Figures 5(a) and (b)). Figure 4(a) also demonstrates that not all trees with maximum degree 6 can be represented: let R denote the square below the central square in the layout, then it is impossible for R to see 5 additional unit squares that exclusively see R. On the other hand, USV contains trees with arbitrarily many degree-6 vertices, e. g., trees of the form depicted in Figure 5(c) (it is straightforward to see that they can be represented as the union of two forests of caterpillars with maximum degree 3). This reasoning shows that not all planar graphs are in USV, while it follows from [30] that all planar graphs are (non-unit square) rectangle visibility graphs (also see [29]).

Finally, we note that USV is a proper subset of $\mathsf{USV_w}$ (e. g., $K_{1,7}$ is a separating example):

▶ **Theorem 12.** $\mathsf{USV} \subsetneq \mathsf{USV_w}$.

4.1 The Recognition Problem

The recognition problem for USV consists in checking whether a given graph can be represented by a unit square layout. We first observe that this problem is in NP (note that this is not completely trivial, since we cannot naively guess a layout) and the main result of this section shall be its hardness (see Theorem 20).

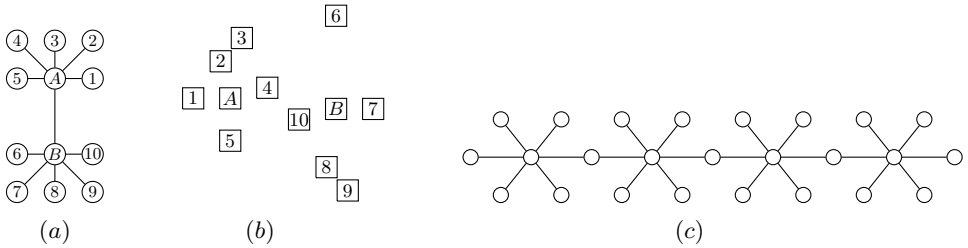


Figure 5 Illustration for trees from USV with maximum degree 6.

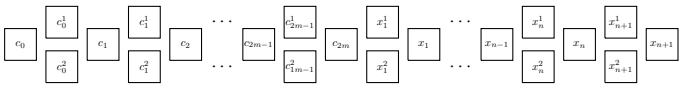


Figure 6 The backbone-gadget.

▶ **Theorem 13.** Rec(USV) ∈ NP.

We prove the NP-hardness by a reduction from NAE-3SAT, i. e., the not-all-equal 3-satisfiability problem [27]. To this end, let $F = \{c_1, \ldots, c_m\}$ be a 3-CNF formula over the variables $x_1, \ldots, x_n$, such that no variable occurs more than once in any clause, and, for the sake of convenience, let $c_i = \{y_{i,1}, y_{i,2}, y_{i,3}\}$, $1 \leq i \leq m$.

The general idea of the reduction is as follows: We identify graph structures that can be shown to have a (more or less) unique representation as a unit square layout. With these main building blocks, we construct a sequence of clause and variable gadgets, called *backbone* (see Figure 6), that can only be represented by a layout in a linear way, say horizontally. Furthermore, every clause gadget is vertically connected to its three literals, two of which are below and the other one above the backbone, or the other way around. The allocation of literal vertices to a variable x_i is done by a path of all literal vertices corresponding to x_i that is connected to the variable vertex for x_i. Such paths must lie either completely above or below the backbone. Interpreting the situation that a path lies above the backbone as assigning *true* to the corresponding literal, yields a not-all-equal satisfying assignment, as it is not possible that all the paths for a clause lie on the same side of the backbone.

We assume that each clause of F contains at most one negated variable, which is no restriction to not-all-equal satisfiability as a clause over literals l_1, l_2, l_3 is not-all-equal satisfied by an assignment if and only if a clause over literals $\bar{l}_1, \bar{l}_2, \bar{l}_3$ is. Furthermore, we also assume that every literal occurs at least three times in the formula. We first transform F into $F' = \{c_1, \ldots, c_{2m}\}$, where $c_{m+i} = c_i$ for $i = 1, \ldots, m$. Then, we transform F' into a graph $G = (V, E)$ as follows. The set of vertices is defined by $V = V_c \cup V_x \cup V_h$, where

$$V_c = \{c_j, c_j^1, c_j^2 \mid 0 \leq j \leq 2m-1\} \cup \{c_{2m}\} \cup \{l_j^1, l_j^2, l_j^3 \mid 1 \leq j \leq 2m\},$$
$$V_x = \{x_i, x_i^1, x_i^2 \mid 1 \leq i \leq n+1\} \cup \{t_i, \vec{t_i}, \overleftarrow{t_i}, f_i^1, \vec{f_i^1}, \overleftarrow{f_i^1}, f_i^2, \vec{f_i^2}, \overleftarrow{f_i^2} \mid 1 \leq i \leq n\},$$
$$V_h = \{h_{t_i}^r, h_{f_i^1}^r, h_{f_i^2}^r \mid 1 \leq i \leq n, 0 \leq r \leq 4\}.$$

Vertices c_j and x_i represent the corresponding clauses and variables and the vertices c_j^r, x_i^r, $r \in \{1, 2\}$ are used to enforce the *backbone* structure as described at the beginning of this section. The corresponding edges are implicitly defined, by requiring, for every $0 \leq i \leq 2m-1$ and $1 \leq i \leq n$, the following groups of 4 vertices to form a K_4: $\{c_j, c_j^1, c_j^2, c_{j+1}\}$, $\{x_i, x_{i+1}^1, x_{i+1}^2, x_{i+1}\}$, and $\{c_{2m}, x_1^1, x_1^2, x_1\}$. Also, for every $j \in \{1, 2\}$, the vertices $c_0^j, c_1^j, \ldots, c_{2m-1}^j, x_1^j, x_2^j, \ldots, x_{n+1}^j$ form a path in this order. Consequently, these

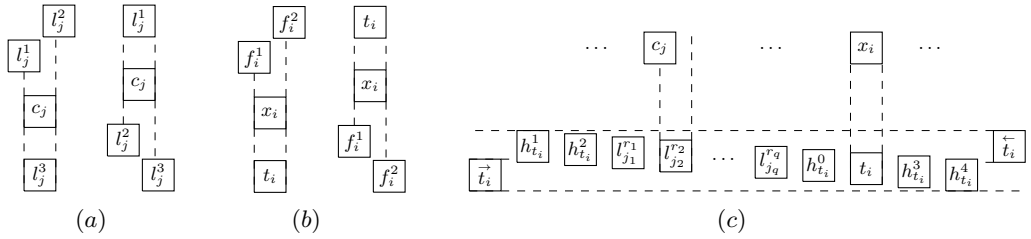


Figure 7 Possible placements of literal vertices, possible placements of assignment vertices, and the clause path for x_i.

vertices form the subgraph represented by the layout in Figure 6, which shall be the *backbone*. Vertices t_i, represent the literal x_i, f_i^1 represent the literal $\overline{x_i}$ in the first m clauses, and f_i^2 represent the literal $\overline{x_i}$ in the remaining clauses. Vertices l_j^1, l_j^2, l_j^3 represent the literals of clause c_j. These roles are reflected with edges $\{x_i, t_i\}$, $\{x_i, f_i^1\}$, $\{x_i, f_i^2\}$ for all $1 \leq i \leq n$ and $\{c_j, l_j^r\}$ for all $1 \leq j \leq 2m$ and $1 \leq r \leq 3$. The connection between literals and variable assignments is build by turning $l_{j,r}$ with $y_{j,r} = x_i$ into a path connected to t_i; analogously, $l_{j,r}$ with $y_{j,r} = \overline{x_i}$ in the first (the last, respectively) m clauses form a path connected to f_i^1 (f_i^2, respectively). More precisely, for every $1 \leq j \leq 2m$, $1 \leq i \leq n$ and $1 \leq r \leq 3$:

- if $y_{j,r} = x_i$, there are edges $\{l_j^r, \vec{t_i}\}$, $\{l_j^r, \overleftarrow{t_i}\}$,
- if $y_{j,r} = \overline{x_i}$ and $1 \leq j \leq m$, there are edges $\{l_j^r, \vec{f_i^1}\}$, $\{l_j^r, \overleftarrow{f_i^1}\}$ and $\{l_{j+m}^r, \vec{f_i^2}\}$, $\{l_{j+m}^r, \overleftarrow{f_i^2}\}$,
- there are edges $\{t_i, \vec{t_i}\}$, $\{t_i, \overleftarrow{t_i}\}$ and $\{\vec{t_i}, h_{t_i}^p\}$,$\{\overleftarrow{t_i}, h_{t_i}^p\}$ for all $0 \leq p \leq 4$,
- there are edges $\{f_i^s, \vec{f_i^s}\}$, $\{f_i, \overleftarrow{f_i^s}\}$ and $\{\vec{f_i^s}, h_{f_i^s}^p\}$,$\{\overleftarrow{f_i^s}, h_{f_i^s}^p\}$ for all $0 \leq p \leq 4$, $s \in \{1, 2\}$,

Moreover, for every i, $1 \leq i \leq n$,

- if $N(\vec{t_i}) = \{h_{t_i}^1, h_{t_i}^2, l_{j1}^{r_1}, l_{j2}^{r_2}, \ldots, l_{jq}^{r_q}, h_{t_i}^0, t_i, h_{t_i}^3, h_{t_i}^4\}$ with $j_1 < j_2 < \ldots < j_q$, then these vertices form a path in this order,
- if $N(\vec{f_i^s}) = \{h_{f_i^s}^1, h_{f_i^s}^2, l_{j1}^{r_1}, l_{j2}^{r_2}, \ldots, l_{jq}^{r_q}, h_{f_i^s}^0, f_i^s, h_{f_i^s}^3, h_{f_i^s}^4\}$ with $j_1 < j_2 < \ldots < j_q$ and $s \in \{1, 2\}$, then these vertices form a path in this order,

Next, we assume that the formula F' is not-all-equal satisfiable and show how a layout for G can be constructed. First, we represent the backbone as illustrated in Figure 6. If a variable x_i is assigned the value *true*, then we place the unit squares $R_{\{x_i, t_i, f_i^1, f_i^2\}}$ as illustrated on the left side of Figure 7(b), and otherwise as illustrated on the right side. The edges for the vertices $t_i, \vec{t_i}, \overleftarrow{t_i}, h_{t_i}^r$, $0 \leq r \leq 4$, and all l_j^r with $y_{j,r} = x_i$ can be realised as illustrated in Figure 7(c) (either placed above or below the backbone, according to the position of R_{t_i}). An analogous construction applies to the unit squares for l_j^r with $y_{j,r} = \overline{x_i}$, with the only difference that we have two such paths (one for the first m clauses and one for the remaining clauses) and that they both lie on the opposite side of the backbone with respect to R_{t_i}. Moreover, in these paths, the $R_{l_j^r}$ must be horizontally shifted such that they can see their corresponding R_{c_j} from above or from below, according to whether the path lies above or below the backbone (as indicated in Figure 7(c)). As long as not all paths for the three literals of the same clause lie all above or all below the backbone, this is possible by arranging the unit squares as illustrated in Figure 7(a). However, if for some clause all paths lie on the same side of the backbone, then the literals of the clause are either all set to *true* or all set to *false*, which is a contradiction to the assumption that the assignment is not-all-equal satisfiable. Consequently, we can represent G as described.

▶ **Lemma 14.** *If F is not-all-equal satisfiable, then $G \in$ USV.*

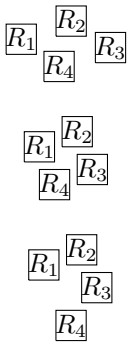


Figure 8 Re- presenting K_4.

Proving that a layout for G translates into a satisfying not-all-equal assignment for F, is much more involved. The general idea is to show that any layout for G must be V-isomorphic to the layout constructed above. However, this cannot be done separately for the individual gadgets, e. g., showing that the backbone must be represented as in Figure 6 (in fact, the structure of the backbone alone does not enforce such a layout) and the literal vertices must form a path as in Figure 7(c) and so on. Instead, the desired structure of the layout is only enforced by a rather complicated interplay of the different parts of G.

A main building stone is that a K_4 can only be represented in 3 different ways (up to V-isomorphism), which are illustrated in Figure 8. This observation is important, since the backbone is a sequence of K_4.

▶ **Lemma 15.** *Every layout for K_4 is V-isomorphic to one of the three layouts of Figure 8.*

We now assume that G can be represented by some layout $\mathcal{R}$. For every j, $1 \leq j \leq m$, we define $L_j = \{l_j^1, l_j^2, l_j^3\}$, for every i, $1 \leq i \leq n$, we define $A_i = \{t_i, f_i^1, f_i^2\}$, and, for every j, $1 \leq j \leq m-1$, we define $C_j^l = \{c_j, c_{j-1}, c_{j-1}^1, c_{j-1}^2\}$, $C_j^r = \{c_j, c_{j+1}, c_j^1, c_j^2\}$ and $C_j = C_j^l \cup C_j^r$.

We shall prove the desired structure of $\mathcal{R}$ by first considering the neighbourhood of c_j; once we have fixed the layout for this subgraph, the structure of the whole layout can be concluded inductively. The neighbourhood of c_j consists of C_j^l and C_j^r (two K_4 joined by c_j) and L_j, where all vertices of the two K_4 (except c_j) are not connected to any vertex of L_j. Intuitively speaking, this independence between L_j and the K_4 of the backbone will force the backbone to expand along one dimension, say horizontally (as depicted in Figure 6), while the visibilities between L_j and c_j must then be vertical (as depicted in Figure 7(a)). However, formally proving this turns out to be quite complicated.

The general proof idea is to somehow place the unit squares of R_{L_j} in such a way that they see R_{c_j} without creating unwanted visibilities. Then, the areas of visibility for the R_{L_j} are blocked for any unit squares from the backbone-neighbourhood R_{C_j}, since these are independent of R_{L_j}. For example, consider the situation depicted in Figure 9. Here, placing unit squares from R_{C_j} in the grey areas implies that they are within visibility of some unit squares from R_{L_j}. This leaves only few possibilities to place the unit squares from R_{C_j} and by applying arguments of this type, it can be concluded, by exhaustively searching all possibilities and under application of Lemma 15, that the only possible layouts have the above described form.

However, this argument is flawed: it is possible to place a unit square R_x within the grey areas, as long as the forbidden visibilities are blocked by other unit squares. This type of blocking would require a path between x and c_j or some vertex from L_j, respectively, which

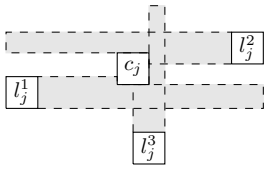


Figure 9 Possible placement of literal vertices for c_j.

does exist as structure in G. Consequently, in order to make the above described argument applicable, we first have to show that the existence of such visibility-blocking unit squares leads to a contradiction. This substantially increases the combinatorial depth of the already technical proof idea described above.

For the next lemma, which is the main tool in proving how the neighbourhood of c_j is represented, we need some notation. Let R_i, R_j, R_k be unit squares. If some (or every) visibility rectangle for R_i and R_k intersects R_j, then R_j is *strictly between* R_i and R_k (or R_j *blocks the view* between R_i and R_k, respectively).

▶ **Lemma 16.** *For all $1 \leq i \leq 2m$ and $r \in \{1,2,3\}$ and every $z \in N(c_i) \setminus \{l_i^r\}$ there exists no visibility rectangle for $R_{l_i^r}$ and R_z that is not intersected by R_{c_i}. In particular, this implies: R_z is not strictly between R_{c_i} and $R_{l_i^r}$, $R_{l_i^r}$ is not strictly between R_{c_i} and R_z, and, if R_{c_i} is strictly between $R_{l_i^r}$ and R_z, then R_{c_i} blocks the view between $R_{l_i^r}$ and R_z.*

By applying Lemma 16, we can now show that $R_{C_j^l}$ and $R_{C_j^r}$ cannot all see R_{c_j} from the same side, which can then be used in order to prove that either all R_{L_j} see R_{c_j} vertically or all of them see R_{c_j} horizontally:

▶ **Lemma 17.** *For every j, $1 \leq j \leq 2m-1$ and $y \in C_j \setminus \{c_j\}$, $R_{c_j} \to R_{C_j \setminus \{y, c_j\}}$ is not possible.*

▶ **Lemma 18.** *For every j, $1 \leq j \leq m$, either $R_{c_j} \leftrightarrow R_{L_j}$ or $R_{c_j} \updownarrow R_{L_j}$.*

We are now able to combine these lemmas in order to prove that a layout for G translates into a not-all-equal satisfying assignment for the formula F. To this end, we first note that the neighbourhood of a variable vertex x_i has an identical structure as the neighbourhood of the clause vertices, which implies that Lemmas 16, 17 and 18 also apply to this part of the graph. By combining Lemmas 16 and 18, we can show that for each clause c_j, either $R_{c_j} \leftrightarrow R_{C_j \setminus \{c_j\}}$ or $R_{c_j} \updownarrow R_{C_j \setminus \{c_j\}}$. By Lemma 15, this means that the two corresponding induced K_4 are represented as shown in Figure 6, and, furthermore, an inductive application of Lemma 17 forces them to form the shown horizontal or vertical backbone. Due to Lemma 18, the literal vertices and the assignment vertices corresponding to the same variable must all form a path on the same side of the backbone. We can now assign x_i the value *true* if and only if R_{t_i} is below the backbone. As long as, for the variables occurring in some clause c_j, $R_{f_i^1}$ is on the opposite side of R_{t_i}, clause c_j is not-all-equal satisfied, because then literals are set to *true* if and only if they are below the backbone and, due to Lemma 16, it is not possible that they all lie on the same side. However, if $R_{f_i^1}$ lies on the same side as R_{t_i}, which is possible, then $R_{f_i^2}$, again due to Lemma 16, must lie on the opposite side of R_{t_i} and, by the same argument, it follows that c_{j+m}, which is a copy of c_j, is not-all-equal satisfied (note that every clause has at most one negated variable).

▶ **Lemma 19.** *If $G \in$ USV, then F is not-all-equal satisfiable.*

▶ **Theorem 20.** Rec(USV) *is* NP*-complete.*

Since in our reduction the size of the graph is linear in the size of the formula, we can also conclude ETH-lower bounds for Rec(USV).

5 Conclusions

The hardness of Rec(USV_w) is still open (note that in our reduction, we heavily used the argument that certain constellations yield forbidden edges, which falls apart in the weak case) and we conjecture it to be NP-hard as well. Two open problems concerning graph classes related to USGV are mentioned in Section 3: (1) are USGV and the class of resolution-$\frac{\pi}{2}$ graphs identical, (2) are there resolution-$\frac{\pi}{2}$ graphs without BRAC-drawing? Note that a positive answer to (2) gives a negative answer to (1).

From a parameterised complexity point of view, our NP-completeness result shows that the number of different rectangle shapes (considered as a parameter) has no influence on the hardness of recognition. Another interesting parameter to explore would be the step size of the grid, i. e., for $k \in \mathbb{N}$, let USGV^k be defined like USGV, but for a $\{\frac{\ell}{k} \mid \ell \in \mathbb{N}\}^2$ grid. We note that these classes form an infinite hierarchy between $\mathsf{USGV} = \mathsf{USGV}^1$ and $\mathsf{USV} = \bigcup_k \mathsf{USGV}^k$, and it is hard to define them in terms of extensions of rectilinear graphs. Another interesting observation is that the hardness reduction for the recognition problem of rectilinear graphs from [17], if interpreted as reduction for Rec(USGV), does not work for USGV^2. The classes USGV^k might be practically more relevant, since placing objects in the plane with discrete distances is more realistic.

Acknowledgements. We thank the organizers of the Lorentz Center workshop 'Fixed Parameter Computational Geometry' in 2016 for the great atmosphere that stimulated this project.

References

1 E. N. Argyriou, M. A. Bekos, and A. Symvonis. Maximizing the total resolution of graphs. In U. Brandes and S. Cornelsen, editors, *Graph Drawing, GD 2010*, volume 6502 of *LNCS*, pages 62–67. Springer, 2011.

2 E. N. Argyriou, M. A. Bekos, and A. Symvonis. The straight-line RAC drawing problem is NP-hard. *Journal of Graph Algorithms and Applications*, 16(2):569–597, 2012.

3 A. Arleo, C. Binucci, E. Di Giacomo, W. S. Evans, L. Grilli, G. Liotta, H. Meijer, F. Montecchiani, S. Whitesides, and S. K. Wismath. Visibility representations of boxes in 2.5 dimensions. In Y. Hu and M. Nöllenburg, editors, *Graph Drawing and Network Visualization - 24th International Symposium, GD*, volume 9801 of *LNCS*, pages 251–265. Springer, 2016.

4 M. Babbitt, J. Geneson, and T. Khovanova. On k-visibility graphs. *Journal of Graph Algorithms and Applications*, 19(1):345–360, 2015.

5 S. N. Bhatt and S. S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Information Processing Letters*, 25(4):263–267, 1987.

6 T. C. Biedl, G. Liotta, and F. Montecchiani. On visibility representations of non-planar graphs. In S. P. Fekete and A. Lubiw, editors, *32nd International Symposium on Computational Geometry, SoCG*, volume 51 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

7 H. L. Bodlaender and G. Tel. A note on rectilinearity and angular resolution. *Journal of Graph Algorithms and Applications*, 8:89–94, 2004.

8 P. Bose, H. Everett, S. P. Fekete, M. E. Houle, A. Lubiw, H. Meijer, K. Romanik, G. Rote, T. C. Shermer, S. Whitesides, and C. Zelle. A visibility representation for graphs in three dimensions. *Journal of Graph Algorithms and Applications*, 2(2), 1998.

9 K. Casel, H. Fernau, A. Grigoriev, M L. Schmid, and S. Whitesides. Combinatorial properties and recognition of unit square visibility graphs, 2017. http://arxiv.org/abs/1706.05906.

10 S. Chaplick, G. Guśpiel, G. Gutowski, T. Krawczyk, and G. Liotta. The partial visibility representation extension problem. In Y. Hu and M. Nöllenburg, editors, *Graph Drawing and Network Visualization - 24th International Symposium, GD*, volume 9801 of *LNCS*, pages 266–279. Springer, 2016.

11 S. Chaplick, F. Lipp, J.-W. Park, and A. Wolff. Obstructing visibilities with one obstacle. In Y. Hu and M. Nöllenburg, editors, *Graph Drawing and Network Visualization - 24th International Symposium, GD*, volume 9801 of *LNCS*, pages 295–308. Springer, 2016.

12 A. M. Dean, J. A. Ellis-Monaghan, S. Hamilton, and G. Pangborn. Unit rectangle visibility graphs. *Electronic Journal of Combinatorics*, 15, 2008.

13 A. M. Dean, W. S. Evans, E. Gethner, J. D. Laison, M. A. Safari, and W. T. Trotter. Bar k-visibility graphs. *Journal of Graph Algorithms and Applications*, 11(1):45–59, 2007.

14 A. M. Dean and J. P. Hutchinson. Rectangle-visibility representations of bipartite graphs. *Discrete Applied Mathematics*, 75(1):9–25, 1997.

15 W. Didimo, P. Eades, and G. Liotta. Drawing graphs with right angle crossings. *Theoretical Computer Science*, 412(39):5156–5166, 2011.

16 P. Duchet, Y. Hamidoune, M. Las Vergnas, and H. Meyniel. Representing a planar graph by vertical lines joining different levels. *Discrete Mathematics*, 46(3):319–321, 1983.

17 P. Eades, S.-H. Hong, and S.-H. Poon. On rectilinear drawing of graphs. In D. Eppstein and E. R. Gansner, editors, *Graph Drawing, 17th International Symposium, GD 2009*, volume 5849 of *LNCS*, pages 232–243. Springer, 2010.

18 W. S. Evans, G. Liotta, and F. Montecchiani. Simultaneous visibility representations of plane st-graphs using L-shapes. *Theoretical Computer Science*, 645:100–111, 2016.

19 S. P. Fekete, M. E. Houle, and S. Whitesides. New results on a visibility representation of graphs in 3D. In F.-J. Brandenburg, editor, *Graph Drawing, Symposium on Graph Drawing, GD'95*, volume 1027 of *LNCS*, pages 234–241. Springer, 1996.

20 S. Felsner. Rectangle and square representations of planar graphs. In J. Pach, editor, *Thirty Essays on Geometric Graph Theory*, pages 213–248. Springer, New York, 2013.

21 M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F. T. Leighton, A. Symvonis, E. Welzl, and G. J. Woeginger. Drawing graphs in the plane with high resolution. In *31st Annual Symposium on Foundations of Computer Science, FOCS, Volume I*, pages 86–95. IEEE Computer Society, 1990.

22 M. R. Garey and D. S. Johnson. *Computers and Intractability.* New York: Freeman, 1979.

23 E. Gaub, M. Rose, and P. S. Wenger. The unit bar visibility number of a graph. *Journal of Graph Algorithms and Applications*, 20(2):269–297, 2016.

24 E. Di Giacomo, W. Didimo, W. S. Evans, G. Liotta, H. Meijer, F. Montecchiani, and S. K. Wismath. Ortho-polygon visibility representations of embedded graphs. In Y. Hu and M. Nöllenburg, editors, *Graph Drawing and Network Visualization - 24th International Symposium, GD*, volume 9801 of *LNCS*, pages 280–294. Springer, 2016.

25 J. Maňuch, M. Patterson, S.-H. Poon, and C. Thachuk. Complexity of finding non-planar rectilinear drawings of graphs. In U. Brandes and S. Cornelsen, editors, *Graph Drawing - 18th International Symposium, GD 2010*, volume 6502 of *LNCS*, pages 305–316. Springer, 2011.

26 N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In D. E. Walker and L. M. Norton, editors, *Proceedings of the 1st International Joint Conference on Artificial Intelligence, IJCAI*, pages 509–520. William Kaufmann, 1969.

27 T. J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th Ann. ACM Symp. Theory of Computing STOC*, pages 216–226. ACM Press, 1978.

28 I. Streinu and S. Whitesides. Rectangle visibility graphs: Characterization, construction, and compaction. In H. Alt and M. Habib, editors, *20th Annual Symposium on Theoretical Aspects of Computer Science, STACS*, volume 2607 of *LNCS*, pages 26–37. Springer, 2003.

29 R. Tamassia and I. G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete & Computational Geometry*, 1(4):321–341, 1986.

30 S. K. Wismath. Characterizing bar line-of-sight graphs. In J. O'Rourke, editor, *Proceedings of the First Annual Symposium on Computational Geometry*, pages 147–152. ACM, 1985.

Weighted Operator Precedence Languages*

Manfred Droste[1], Stefan Dück[2], Dino Mandrioli[3], and Matteo Pradella[4]

1 **Institute of Computer Science, Leipzig University, Germany**
droste@informatik.uni-leipzig.de
2 **Institute of Computer Science, Leipzig University, Germany**
dueck@informatik.uni-leipzig.de
3 **Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Italy**
dino.mandrioli@polimi.it
4 **Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Italy, and**
IEIIT, Consiglio Nazionale delle Ricerche, via Ponzio 34/5, 20133 Milano, Italy
matteo.pradella@polimi.it

Abstract

In the last years renewed investigation of operator precedence languages (OPL) led to discover important properties thereof: OPL are closed with respect to all major operations, are characterized, besides the original grammar family, in terms of an automata family (OPA) and an MSO logic; furthermore they significantly generalize the well-known visibly pushdown languages (VPL). In another area of research, quantitative models of systems are also greatly in demand. In this paper, we lay the foundation to marry these two research fields. We introduce weighted operator precedence automata and show how they are both strict extensions of OPA and weighted visibly pushdown automata. We prove a Nivat-like result which shows that quantitative OPL can be described by unweighted OPA and very particular weighted OPA. In a Büchi-like theorem, we show that weighted OPA are expressively equivalent to a weighted MSO-logic for OPL.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic, F.4.3 Formal Languages

Keywords and phrases Quantitative automata, operator precedence languages, input-driven languages, visibly pushdown languages, quantitative logic.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.31

1 Introduction

In the long history of formal languages the family of regular languages (RL) has always played a major role: thanks to its simplicity and naturalness, it enjoys many positive mathematical properties which have been thoroughly exploited in disparate practical applications; among them, those of main interest in this paper are the following:

- RL have been characterized in terms of various mathematical logics. Originally, Büchi, Elgot, and Trakhtenbrot [6, 18, 34] independently developed a monadic second order (MSO) logic defining exactly the RL family. This work has been followed by many further results; in particular those that exploited weaker but simpler logics such as first-order,

* This work was supported by Deutsche Forschungsgemeinschaft (DFG) Graduiertenkolleg 1763 (QuantLA).

propositional, and temporal ones culminated in the breakthrough of model checking to support automatic verification [28, 19, 7].

- Weighted RL have been introduced by Schützenberger in [32]: by assigning a weight in a suitable algebra to each language word, we may specify several attributes of the word, e.g., relevance, probability, etc. Much research then followed and extended Schützenberger's original work in various directions, cf. the books [4, 17, 23, 31, 13].

Unfortunately, all families with greater expressive power than RL – typically context-free languages (CFL), which are the most widely used family in practical applications – pay a price in terms of algebraic and logic properties and, consequently, of possible tools supporting their automatic analysis. For instance, for CFL, the containment problem is undecidable.

What was not possible for general CFL, however, has been possible for important subclasses of this family, which together we call *structured CFL*. Informally, by this term we denote those CFL where the syntactic tree-structure of their words is immediately "visible" in the words themselves. Two first equivalent examples of such families are parenthesis languages [27], which are generated by grammars whose right hand sides are enclosed within pairs of parentheses, and tree-automata [33], which generalize finite state machines (FSM) from the recognition of linear strings to tree-like structures. Among the many variations of parenthesis languages the recent family of *input-driven languages* [29, 35], alias *visibly pushdown languages (VPL)* [2], has received much attention in recent literature. For most of these structured CFL, including VPL, the relevant algebraic properties of RL still hold [2]. One of the most noticeable results has been a characterization of VPL in terms of a MSO logic that is a natural extension of Büchi's original one for RL [24, 2].

This fact has suggested to extend the investigation of weighted RL to various cases of structured languages. The result of such a fertile approach is a rich collection of *weighted logics*, first studied by Droste and Gastin [11], associated with *weighted tree automata* [16] and weighted extensions of VPA (the automata recognizing VPL) [26].

In an originally unrelated way *operator precedence languages (OPL)* have been defined and studied in two phases temporally separated by four decades. In his seminal work [20] Floyd was inspired by the precedence of multiplicative operations over additive ones in the execution of arithmetic expressions and extended such a relation to the whole input alphabet in such a way that it could drive a deterministic parsing algorithm that builds the syntax tree of any word that reflects the word's semantics; Fig. 1 and Section 2 give an intuition of how an OP grammar generates arithmetic expressions and assigns them a natural structure.

OPL do not cover all deterministic CFL, but they enjoy a distinguishing property, not possessed by general deterministic CFL, which we can intuitively describe as "*OPL are input driven but not visible*". They can be claimed as *input-driven* since the parsing actions on their words – whether to push or pop – depend exclusively on the input alphabet and on the relation defined thereon, but their structure is *not visible* in their words: e.g, they can include unparenthesized expressions where the precedence of multiplicative operators over additive ones is explicit in the syntax trees but hidden in their frontiers (see Fig. 1). Furthermore, unlike other structured CFL, OPL include deterministic CFL that are not real-time [25].

This recent remark suggested to resume their investigation systematically at the light of the recent technological advances and related challenges. Such a renewed investigation led to prove their closure under all major language operations [8] and to characterize them, besides Floyd's original grammars, in terms of an appropriate class of pushdown automata (OPA) and in terms of a MSO logic which is a fairly natural but not trivial extension of the previous ones defined to characterize RL and VPL [25]. Thus, OPL enjoy the same nice properties of RL and many structured CFL but considerably extend their applicability by breaking the barrier of visibility and real-time push-down recognition.

In this paper we join the two research fields above, namely we introduce *weighted OPL* and show that they are able to model system behaviors that cannot be specified by means of less powerful weighted formalisms such as weighted VPL. For instance, one might be interested in the behavior of a system which handles calls and returns but is subject to some emergency interrupts. Then it is important to evaluate how critically the occurrences of interrupts affect the normal system behavior, e.g., by counting the number of pending calls that have been preempted by an interrupt. As another example we consider a system logging all hierarchical calls and returns over words where this structural information is hidden. Depending on changing exterior factors like energy level, such a system could decide to log the above information in a selective way.

Our main contributions in this paper are the following.

- The model of *weighted OPA*, which have semiring weights at their transitions, significantly increases the descriptive power of previous weighted extensions of VPA, and has desired closure and robustness properties.
- For arbitrary semirings, there is a relevant difference in the expressive power of the model depending on whether it permits assigning weights to pop transitions or not. For commutative semirings, however, weights on pop transitions do not increase the expressive power of the automata. The difference in descriptive power between weighted OPA with arbitrary weights and without weights at pop transitions is due to the fact that OPL may be non-real-time and therefore OPA may execute several pop moves without advancing their reading heads.
- An extension of the classical result of Nivat [30] to weighted OPL. This robustness result shows that the behaviors of weighted OPA without weights at pop transitions are exactly those that can be constructed from weighted OPA with only one state, intersected with OPL, and applying projections which preserve the structural information.
- A weighted MSO logic and, for arbitrary semirings, a Büchi-Elgot-Trakhtenbrot-Theorem proving its expressive equivalence to weighted OPA without weights at pop transitions. As a corollary, for commutative semirings this weighted logic is equivalent to weighted OPA including weights at pop transitions.

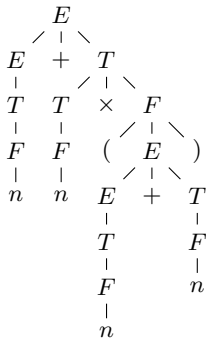
Various possibilities arise for future research concerning theory and applications of our new model which will be discussed in the conclusion. The full version of this paper [10] provides all omitted technicalities and more explanatory comments and examples.

2 Preliminaries

Consider the CFG of Fig. 1 (left) and the syntax tree (center) which makes the structure of its frontier $n+n\times(n+n)$ visible. To drive a parsing algorithm in the deterministic construction of the tree associated with the string, Floyd introduced three *precedence relations*, $\lessdot$ (*yields precedence*), $\doteq$ (*equal in precedence*), $\gtrdot$ (*takes precedence*), (algorithmically derived from the grammar) between terminal symbols (Fig. 1 right). They do not satisfy any order axioms and are used to mark, respectively, the beginning, the internal elements, and the end of a grammar right hand side in the substitution rules of a shift-reduce parsing algorithm. For a complete description of Floyd's parsing algorithms driven by these relations, see, e.g, [21].

In this paper, instead, we exploit the more recent characterization of OPL in terms of recognizing automata [25], which are defined on a given alphabet *and* precedence matrix. We define an *OP alphabet* as a pair (Σ, M), where Σ is an alphabet and M, the *operator precedence matrix (OPM)*, is a $|\Sigma\cup\{\#\}|^2$ array describing for each ordered pair of symbols at most one (operator precedence) relation, that is, every entry of M is either $\lessdot$, $\doteq$, $\gtrdot$, or

$E \to E + T \mid T$
$T \to T \times F \mid F$
$F \to n \mid (E)$



	$+$	$\times$	$($	$)$	n
$+$	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$
$\times$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\lessdot$
$($	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	$\lessdot$
$)$	$\gtrdot$	$\gtrdot$		$\gtrdot$	
n	$\gtrdot$	$\gtrdot$		$\gtrdot$	

Figure 1 A grammar generating arithmetic expressions (left), an example derivation tree (center), and the precedence matrix (right). E.g. $M[1,2] = \lessdot$ means that $+$ yields precedence to $\times$.

empty (no relation). *We use the symbol # to mark the beginning and the end of a word and always let* $\# \lessdot a$ *and* $a \gtrdot \#$ *for all* $a \in \Sigma$.

Let $w = (a_1 ... a_n) \in \Sigma^+$ be a non-empty word. We say $a_0 = a_{n+1} = \#$ and define a new *chain* relation $\curvearrowright$ on the set of all positions of $\#w\#$, inductively, as follows. Let $0 \leq i < j \leq n+1$. We write $i \curvearrowright j$ if there exists a sequence of positions $i = k_1 < ... < k_m = j$, $m \geq 3$, such that $a_{k_1} \lessdot a_{k_2} \doteq ... \doteq a_{k_{m-1}} \gtrdot a_{k_m}$ and either $k_s + 1 = k_{s+1}$ or $k_s \curvearrowright k_{s+1}$ for each $s \in \{1, ..., m-1\}$. We say that w is *compatible* with M if for $\#w\#$, we have $0 \curvearrowright n+1$. We denote by (Σ^+, M) the set of all non-empty words over Σ which are compatible with M. For a *complete* OPM M, i.e. one without empty entries, this is Σ^+.

The chain relation can be compared with the *nesting* or *matching relation* of [2], which is also originating from additional information on the alphabet. However, instead of partitioning the alphabet into three disjoint parts (calls, internals, and returns), we add a *binary* relation for every pair of symbols denoting their precedence relation. Therefore, in contrast to the nesting relation, the same symbol can be either call or return depending on its context, and the same position can be part of multiple chain relations.

▶ **Definition 1.** A *(nondeterministic) operator precedence automaton (OPA)* $\mathcal{A}$ over an OP alphabet (Σ, M) is a tuple $\mathcal{A} = (Q, I, F, \delta)$, where $\delta = (\delta_{\text{shift}}, \delta_{\text{push}}, \delta_{\text{pop}})$, consisting of

- a finite set of states Q, the set of initial states $I \subseteq Q$, the set of final states $F \subseteq Q$, and
- the transition relations $\delta_{\text{shift}}, \delta_{\text{push}} \subseteq Q \times \Sigma \times Q$, and $\delta_{\text{pop}} \subseteq Q \times Q \times Q$.

Let $\Gamma = \Sigma \times Q$. A *configuration* of $\mathcal{A}$ is a triple $C = \langle \Pi, q, w\# \rangle$, where $\Pi \in \bot\Gamma^*$ represents a stack, $q \in Q$ the current state, and w the remaining input to read. A *run* of $\mathcal{A}$ on $w = a_1 ... a_n$ is a finite sequence of configurations $C_0 \vdash ... \vdash C_m$, such that every transition $C_i \vdash C_{i+1}$ has one of the following forms, where a is the first component of the topmost symbol of the stack Π, or $\#$ if the stack is $\bot$, and b is the next symbol of the input to read:

$$\begin{aligned}
&\textit{push move}: && \langle \Pi, q, bx \rangle && \vdash \langle \Pi[b, q], r, x \rangle && \text{if } a \lessdot b \text{ and } (q, b, r) \in \delta_{\text{push}},\\
&\textit{shift move}: && \langle \Pi[a, p], q, bx \rangle && \vdash \langle \Pi[b, p], r, x \rangle && \text{if } a \doteq b \text{ and } (q, b, r) \in \delta_{\text{shift}},\\
&\textit{pop move}: && \langle \Pi[a, p], q, bx \rangle && \vdash \langle \Pi, r, bx \rangle && \text{if } a \gtrdot b \text{ and } (q, p, r) \in \delta_{\text{pop}}.
\end{aligned}$$

An *accepting run* of $\mathcal{A}$ on w is a run from $\langle \bot, q_I, w\# \rangle$ to $\langle \bot, q_F, \# \rangle$, where $q_I \in I$ and $q_F \in F$. The *language accepted by* $\mathcal{A}$, denoted $L(\mathcal{A})$, consists of all words over (Σ^+, M) which have an accepting run on $\mathcal{A}$. We say $L \subseteq (\Sigma^+, M)$ is an *OPL* if L is accepted by an OPA over (Σ, M). As proven in [25], the deterministic variant of an OPA, using a single initial state and transition functions instead of relations, is as expressive as nondeterministic OPA.

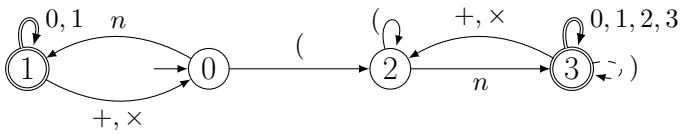


Figure 2 An OPA recognizing the language of the grammar of Fig. 1. The graphical notation is imported from [25]: pushes are normal arrows, shifts are dashed, pops are double arrows.

An example automaton is depicted in Fig. 2: with the OPM of Fig. 1 (right), it accepts the same language as the grammar of Fig. 1 (left).

▶ **Definition 2.** The logic MSO(Σ, M), short MSO, and its semantics is defined as in [25]

$$\beta ::= \mathrm{Lab}_a(x) \mid x \leq y \mid x \curvearrowright y \mid x \in X \mid \neg\beta \mid \beta \vee \beta \mid \exists x.\beta \mid \exists X.\beta$$

where $a \in \Sigma \cup \{\#\}$ and x, y, X are first resp. second order variables. The predicate $\mathrm{Lab}_a(x)$ asserts that position x is labeled a. The semantics of $\curvearrowright$ is defined by the chain relation.

▶ **Theorem 3** ([25]). *A language L over (Σ, M) is an OPL iff it is MSO-definable.*

3 Weighted OPL and Their Relation to Weighted VPL

In this section, we introduce a weighted extension of OPA. We show that weighted OPL include weighted visibly pushdown automata (VPL) and give examples showing how these weighted automata can express behaviors which were not expressible before.

Let $\mathbb{K} = (K, +, \cdot, 0, 1)$ be a *semiring*, i.e., $(K, +, 0)$ is a commutative monoid, $(K, \cdot, 1)$ is a monoid, $(x+y)\cdot z = x\cdot z + y\cdot z$, $x\cdot(y+z) = x\cdot y + x\cdot z$, and $0\cdot x = x\cdot 0 = 0$ for all $x, y, z \in K$. $\mathbb{K}$ is called *commutative* if $(K, \cdot, 1)$ is commutative. Important examples of commutative semirings include the Boolean semiring $\mathbb{B} = (\{0,1\}, \vee, \wedge, 0, 1)$, the semiring of the natural numbers $\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$, or the tropical semirings $\mathbb{R}_{\max} = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$ and $\mathbb{R}_{\min} = (\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$. Significant non-commutative semirings are $n \times n$-matrices over semirings $\mathbb{K}$ with matrix addition and multiplication as usual ($n \geq 2$), or the semiring $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\})$ of languages over Σ.

▶ **Definition 4.** A *weighted OPA (wOPA)* $\mathcal{A}$ over an OP alphabet (Σ, M) and a semiring $\mathbb{K}$ is a tuple $\mathcal{A} = (Q, I, F, \delta, \mathrm{wt})$, where $\mathrm{wt} = (\mathrm{wt}_{\mathrm{shift}}, \mathrm{wt}_{\mathrm{push}}, \mathrm{wt}_{\mathrm{pop}})$, consisting of

- an OPA $\mathcal{A}' = (Q, I, F, \delta)$ over (Σ, M) and
- the weight functions $\mathrm{wt}_{op} : \delta_{op} \to K$, $op \in \{\mathrm{shift}, \mathrm{push}, \mathrm{pop}\}$.

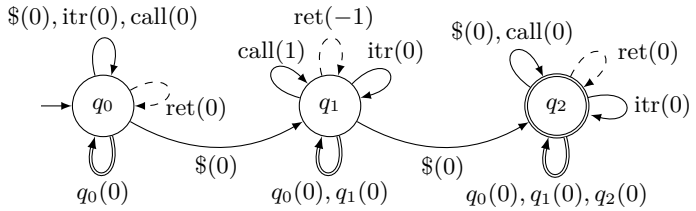
We call a wOPA *restricted*, denoted by *rwOPA*, if $\mathrm{wt}_{\mathrm{pop}}(q, p, r) = 1$ for each $(q, p, r) \in \delta_{\mathrm{pop}}$.

A *configuration* of a wOPA is a tuple $\langle \Pi, q, w\#, k\rangle$, where $(\Pi, q, w\#)$ is a configuration of the OPA $\mathcal{A}'$ and $k \in K$. A *run* of $\mathcal{A}$ is defined as for OPA, where, additionally, the weight k is updated by multiplying with the weight of the encountered transition, as follows.

$$\begin{aligned}
\langle \Pi, q, bx, k\rangle &\vdash \langle \Pi[b, q], r, x, k \cdot \mathrm{wt}_{\mathrm{push}}(q, b, r)\rangle && \text{if } a \lessdot b \text{ and } (q, b, r) \in \delta_{\mathrm{push}},\\
\langle \Pi[a, p], q, bx, k\rangle &\vdash \langle \Pi[b, p], r, x, k \cdot \mathrm{wt}_{\mathrm{shift}}(q, b, r)\rangle && \text{if } a \doteq b \text{ and } (q, b, r) \in \delta_{\mathrm{shift}},\\
\langle \Pi[a, p], q, bx, k\rangle &\vdash \langle \Pi, r, bx, k \cdot \mathrm{wt}_{\mathrm{pop}}(q, p, r)\rangle && \text{if } a \gtrdot b \text{ and } (q, p, r) \in \delta_{\mathrm{pop}}.
\end{aligned}$$

We call a run ρ *accepting* if it goes from $\langle \bot, q_I, w\#, 1\rangle$ to $\langle \bot, q_F, \#, k\rangle$, where $q_I \in I$ and $q_F \in F$. For such an accepting run, the *weight of* ρ is defined as $\mathrm{wt}(\rho) = k$. Finally, the *behavior of* $\mathcal{A}$ is a function $[\![\mathcal{A}]\!] : (\Sigma^+, M) \to K$, defined as

$$[\![\mathcal{A}]\!](w) = \sum_{\rho \text{ acc. run of } \mathcal{A} \text{ on } w} \mathrm{wt}(\rho) \ .$$



	call	ret	itr	$
call	⋖	≐	⋗	⋖
ret	⋗	⋗	⋗	⋖
itr	⋗		⋖	⋖
$	⋗	⋗	⋗	

Figure 3 The weighted OPA $\mathcal{A}_{\text{itr}}$ penalizing unmatched calls nondeterministically, and its precedence matrix (right). Weights are given in parentheses at transitions. The weight semiring is $\mathbb{Z}_{\max} = (\mathbb{Z} \cup \{-\infty\}, \max, +, -\infty, 0)$.

Every function $S : (\Sigma^+, M) \to K$ is called an *OP-series* (short: *series*, also *weighted language*). A wOPA $\mathcal{A}$ *accepts* a series S if $[\![\mathcal{A}]\!] = S$. A series S is called *recognizable* or a *wOPL* if there exists an wOPA $\mathcal{A}$ accepting it. S is *strictly recognizable* or an *rwOPL* if there exists an rwOPA $\mathcal{A}$ accepting it.

▶ **Example 5.** Consider a system that manages calls and returns (in VPL terminology) in a traditional LIFO policy but discards all pending calls if an interrupt (*itr*) occurs. Such a system can be naturally modeled by suitable OPA that can formalize various types of policies to manage interrupts [25][1]. We can use weights to , for instance, count the number of interrupted calls. A first simple wOPA could attach a negative weight to calls and a compensating one to corresponding returns so that the final weight assigned to the string would be "neutral" only if no call is discarded.

Consider now a more complex system where the penalties for unmatched calls may change nondeterministically. Here, we assume words to be separated into different intervals by the symbol \$, of which one nondeterministically chosen represents, e.g., a critical operating time, during which unmatched calls are penalized. The wOPA $\mathcal{A}_{\text{itr}}$ given in Fig. 3 formalizes such a system by assigning to an input sequence a global weight that is the maximal number of unmatched calls in one interval.

$\mathcal{A}_{\text{itr}}$ can be easily modified to formalize several variations of its policy: e.g., different policies could be associated with different intervals, different weights could be assigned to different types of calls and/or interrupts, different policies could also be defined by choosing different semirings, etc. Note that $\mathcal{A}_{\text{itr}}$ is restricted.

▶ **Example 6.** The automaton $\mathcal{A}_{\log}$, depicted in Fig. 4, chooses non-deterministically between logging everything and logging only 'important' information, e.g., only interrupts (this could be a system dependent on energy, WiFi, etc.). Notice that in this case assigning nontrivial weights to pop transitions is crucial. Let $\Sigma = \{\text{call}, \text{ret}, \text{itr}\}$, and define M as the obvious projection of $\mathcal{A}_{\text{itr}}$'s OPM. We employ the semiring $(\text{Fin}_{\Sigma'}, \cup, \circ, \emptyset, \{\varepsilon\})$ of all finite languages over $\Sigma' = \{c, r, p, i\}$. Then, $[\![\mathcal{A}_{\log}]\!](w)$ yields all possible logs on w.

The above example can be exploited to show by a pumping-like argument that wOPA are more expressive than rwOPA. This is due to the fact that a number of consecutive pops can attach to one position a product of size only bounded by the word-length and it is impossible to attach these weights at other positions without destroying their sequential order.

▶ **Proposition 7.** *There exist an OP alphabet (Σ, M), a semiring $\mathbb{K}$, and a weighted language $S : (\Sigma^+, M) \to K$ such that S is recognizable but not strictly recognizable.*

[1] A similar motivation inspired the recent extension of VPL as colored nested words by [1].

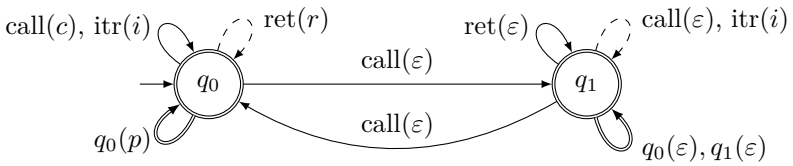


Figure 4 The wOPA $\mathcal{A}_{\text{log}}$ nondeterministically writes logs at different levels of detail.

	Σ_{call}	Σ_{ret}	Σ_{int}
Σ_{call}	$\lessdot$	$\doteq$	$\lessdot$
Σ_{ret}	$\gtrdot$	$\gtrdot$	$\gtrdot$
Σ_{int}	$\gtrdot$	$\gtrdot$	$\gtrdot$

E.g. $w = a\langle car\rangle$, over $\Sigma_{\text{int}} = \{a\}$, $\Sigma_{\text{call}} = \{\langle c\}$, $\Sigma_{\text{ret}} = \{r\rangle\}$

NWA: $q_0 \xrightarrow{a} q_1 \xrightarrow{\langle c} q_2 \xrightarrow{a} q_3 \xrightarrow{r\rangle} q_4$

OPA: $q_0 \xrightarrow{a} q_1' \Rightarrow q_1 \xrightarrow{\langle c} q_2 \xrightarrow{a} q_3' \Rightarrow q_3 \overset{r\rangle}{\dashrightarrow} q_4' \Rightarrow q_4$

Figure 5 The OPM M for VPL and an example of the translation of runs from NWA to OPA.

On the other hand, for commutative semirings rwOPA and wOPA are equally expressive.

▶ **Theorem 8.** *Let $\mathcal{A}$ be a wOPA over an OP alphabet (Σ, M) and a commutative semiring $\mathbb{K}$. Then, there exists an rwOPA $\mathcal{B}$ over (Σ, M) and $\mathbb{K}$ with $[\![\mathcal{A}]\!] = [\![\mathcal{B}]\!]$.*

Proof (Sketch). Let $\mathcal{A} = (Q, I, F, \delta, \text{wt})$ be a wOPA over (Σ, M) and $\mathbb{K}$. We construct an rwOPA $\mathcal{B}$ over (Σ, M) and $\mathbb{K}$ with the state set $Q' = Q \times Q \times Q$ and with the same behavior as $\mathcal{A}$ as follows. In the first state component $\mathcal{B}$ simulates $\mathcal{A}$. In the second and third state component of Q' the automaton $\mathcal{B}$ guesses the states q and r of the pop transition (q, p, r) of $\mathcal{A}$ which corresponds to the next push transition following after this configuration. This enables us to transfer the weight from the pop transition to the correct push transition. ◀

In the following, we show that rwOPL strictly include *weighted visibly pushdown languages* (wVPL). VPL is the class of languages corresponding to nested words [2] and recognized by *visibly pushdown automata* (VPA) or the expressively equivalent *nested word automata* (NWA). Let Σ be a *visibly pushdown alphabet* consisting of calls, internals, and returns. In [8], it was shown that for every VPA over Σ, there exists an equivalent OPA [25] over (Σ, M), where M is the OPM defined in Fig. 5.

In [26, 15], *weighted nested word automata* (wNWA) were introduced. These add semiring weights at every transition again depending on the information which symbols are calls, internals, or returns. Since every nested word has a unique representation over a visibly pushdown alphabet Σ, it can be interpreted as a compatible word of (Σ^+, M). In particular, we can interpret a wVPL, i.e., the language of a wNWA, as an OP-series $(\Sigma^+, M) \to \mathbb{K}$.

▶ **Theorem 9.** *Let $\mathbb{K}$ be a semiring and M be the OPM of Fig. 5. Then for every wNWA $\mathcal{A}$ as defined in [15], there exists an rwOPA $\mathcal{B}$ with $[\![\mathcal{A}]\!](w) = [\![\mathcal{B}]\!](w)$ for all $w \in (\Sigma^+, M)$.*

We give an intuition for this result as follows. Note that pushes, shifts, and pops significantly differ from calls, internals, and returns. Indeed, a return of a NWA reads and 'consumes' a symbol, while a pop of an OPA just pops the stack and leaves the next symbol untouched. Studying the OPM M and the example runs of Fig. 5, we see that every symbol of Σ_{ret} forces a shift transition of an OPA immediately followed by a pop. This suggests a fairly natural construction where we can simulate every weighted call by a weighted push, every

weighted internal by a weighted push together with a pop and every weighted return by a weighted shift together with a pop. Hence, we may omit weights at pop transitions.

Since OPA are strictly more expressive than VPA [8], this gives, together with Proposition 7, a complete picture of the expressive power of these three classes of weighted languages:

$$\text{wVPL} \subsetneq \text{rwOPL} \subsetneq \text{wOPL} \ .$$

Note that in the context of formal power series, wVPL strictly contain recognizable power series and wOPL form a proper subset of the class of algebraic power series, i.e., series recognized by weighted pushdown automata [23].

4 A Nivat Theorem

In this section, we show that strictly recognizable series are exactly those series which can be derived from a restricted weighted OPA with only one state, intersected with an unweighted OPL, and using an OPM-preserving projection of the alphabet.

In the following, we study closure properties of wOPL and rwOPL. As usual, we extend the operations $+$ and $\cdot$ to series by means of pointwise definitions.

▶ **Proposition 10.** *Let $S : (\Sigma^+, M) \to K$ be a recognizable (resp. strictly recognizable) series and $L \subseteq (\Sigma^+, M)$ an OPL. Then, the series $(S \cap L)(w) = \left\{ \begin{array}{ll} S(w) & , \text{ if } w \in L \\ 0 & , \text{ otherwise} \end{array} \right\}$ is recognizable (resp. strictly recognizable).*

Furthermore, if $\mathbb{K}$ is commutative, then the product of two recognizable (resp. strictly recognizable) series over (Σ^+, M) is again recognizable (resp. strictly recognizable).

Next, we show that recognizable series are closed under projections which preserve the OPM. For two OP alphabets (Σ, M), (Γ, M'), we write $h : (\Sigma, M) \to (\Gamma, M')$ for a mapping $h : \Sigma \to \Gamma$ such that for all $\bullet \in \{\lessdot, \doteq, \gtrdot\}$, we have $a \bullet b$ if and only if $h(a) \bullet h(b)$. We can extend h to a function $h : (\Sigma^+, M) \to (\Gamma^+, M')$ by setting $h(a_1 a_2 ... a_n) = h(a_1)h(a_2)...h(a_n)$. Let $S : (\Sigma^+, M) \to K$ be a series. We define $h(S) : (\Gamma^+, M') \to K$ for each $v \in (\Gamma^+, M')$ by

$$h(S)(v) = \sum_{w \in (\Sigma^+, M), h(w) = v} S(w) \ . \tag{1}$$

▶ **Proposition 11.** *Let $\mathbb{K}$ be a semiring, $S : (\Sigma^+, M) \to K$ recognizable (resp. strictly recognizable), and $h : (\Sigma, M) \to (\Gamma, M')$. Then, $h(S) : (\Gamma^+, M') \to K$ is recognizable (resp. strictly recognizable).*

Let h be a map between two alphabets. Given $h : \Gamma \to \Sigma$ and an OP alphabet (Σ, M), we define $h^{-1}(M)$ by setting $h^{-1}(M)_{a'b'} = M_{h(a')h(b')}$ for all $a', b' \in \Gamma$. As h is OPM-preserving, for every series $S : (\Sigma^+, M) \to K$, we get a series $h(S) : (\Gamma^+, h^{-1}(M)) \to K$, using the sum over all pre-images as in formula (1).

Let $\mathcal{N}(\Sigma, M, \mathbb{K})$ comprise all series $S : (\Sigma^+, M) \to K$ for which there exist an alphabet Γ, over $(\Gamma, h^{-1}(M))$ such that $S = h([\![\mathcal{B}]\!] \cap L)$.

Then, we can show that every rwOPL can be decomposed into the above introduced fragments. Using this decomposition and the closure properties of Prop. 10 and Prop. 11, we get the following Nivat-Theorem for weighted operator precedence automata.

▶ **Theorem 12.** *Let $\mathbb{K}$ be a semiring and $S : (\Sigma^+, M) \to K$ be a series. Then S is strictly recognizable if and only if $S \in \mathcal{N}(\Sigma, M, \mathbb{K})$.*

$$
\begin{aligned}
[\![\beta]\!]_{\mathcal{V}}(w,\sigma) &= \begin{cases} 1\,, & \text{if } (w,\sigma) \models \beta \\ 0\,, & \text{otherwise} \end{cases} & [\![\bigoplus_x \varphi]\!]_{\mathcal{V}}(w,\sigma) &= \sum_{i\in|w|} [\![\varphi]\!]_{\mathcal{V}\cup\{x\}}(w,\sigma[x\to i]) \\
[\![k]\!]_{\mathcal{V}}(w,\sigma) &= k \quad \text{for all } k\in\mathbb{K} & [\![\bigoplus_X \varphi]\!]_{\mathcal{V}}(w,\sigma) &= \sum_{I\subseteq|w|} [\![\varphi]\!]_{\mathcal{V}\cup\{X\}}(w,\sigma[X\to I]) \\
[\![\varphi\oplus\psi]\!]_{\mathcal{V}}(w,\sigma) &= [\![\varphi]\!]_{\mathcal{V}}(w,\sigma) + [\![\psi]\!]_{\mathcal{V}}(w,\sigma) & [\![\prod_x \varphi]\!]_{\mathcal{V}}(w,\sigma) &= \prod_{i\in|w|} [\![\varphi]\!]_{\mathcal{V}\cup\{x\}}(w,\sigma[x\to i]) \\
[\![\varphi\otimes\psi]\!]_{\mathcal{V}}(w,\sigma) &= [\![\varphi]\!]_{\mathcal{V}}(w,\sigma)\cdot[\![\psi]\!]_{\mathcal{V}}(w,\sigma)
\end{aligned}
$$

Figure 6 Semantics of weighted MSO logic for OPL.

5 Weighted MSO-Logic for OPL

We use modified ideas from Droste and Gastin [11], also incorporating the distinction into *boolean* formulas β and *weighted* formulas φ as in [5]. The boolean formulas model classical unweighted features, whereas weighted formulas may deal with quantitative aspects.

▶ **Definition 13.** We define the weighted logic MSO$(\mathbb{K},(\Sigma,M))$, short MSO$(\mathbb{K})$, as

$$
\begin{aligned}
&\beta ::= \mathrm{Lab}_a(x) \mid x\le y \mid x \curvearrowright y \mid x\in X \mid \neg\beta \mid \beta\vee\beta \mid \exists x.\beta \mid \exists X.\beta \\
&\varphi ::= \beta \mid k \mid \varphi\oplus\varphi \mid \varphi\otimes\varphi \mid \textstyle\bigoplus_x \varphi \mid \bigoplus_X \varphi \mid \prod_x \varphi
\end{aligned}
$$

where $a\in\Sigma\cup\{\#\}$, $k\in\mathbb{K}$; x,y are first order variables; and X is a second order variable.

Let $w\in(\Sigma^+,M)$ and $\varphi\in$ MSO$(\mathbb{K})$. As usual, let $[w]=\{1,...,|w|\}$ and $\mathcal{V}$ be a finite set of variables containing *free*(φ), all free variables of φ. A $(\mathcal{V},w)$-*assignment* σ is a function assigning to every first order variable of $\mathcal{V}$ an element of $[w]$ and to every second order variable a subset of $[w]$. We define $\sigma[x\to i]$ (and analogously $\sigma[X\to I]$) as the $(\mathcal{V}\cup\{x\},w)$-assignment mapping x to i and coinciding with σ on all variables different from x.

By following classical approaches, we consider the extended alphabet $\Sigma_{\mathcal{V}}=A\times\{0,1\}^{\mathcal{V}}$ together with its natural OPM $M_{\mathcal{V}}$ defined such that for all $(a,s),(b,t)\in\Sigma_{\mathcal{V}}$ and all $\bullet\in\{\lessdot,\doteq,\gtrdot\}$, we have $(a,s)\bullet(b,t)$ if and only if $a\bullet b$. We represent the word w together with the assignment σ as a word (w,σ) over $(\Sigma_{\mathcal{V}},M_{\mathcal{V}})$ such that 1 denotes every position where x resp. X holds. A word over $\Sigma_{\mathcal{V}}$ is called *valid*, if every first order variable is assigned to exactly one position. Being valid is a property which can be checked by an OPA.

We define the *semantics* of $\varphi\in$ MSO$(\mathbb{K})$ as a function $[\![\varphi]\!]_{\mathcal{V}}:(\Sigma_{\mathcal{V}}^+,M_{\mathcal{V}})\to K$ inductively for all valid $(w,\sigma)\in(\Sigma_{\mathcal{V}}^+,M_{\mathcal{V}})$ in Fig. 6. For not valid (w,σ), we set $[\![\varphi]\!]_{\mathcal{V}}(w,\sigma)=0$. We write $[\![\varphi]\!]$ for $[\![\varphi]\!]_{\mathrm{free}(\varphi)}$.

We can show that semantics $\varphi_{\mathcal{V}}$ for different $\mathcal{V}$ are consistent with each other as long as $\mathcal{V}$ contains all free variables of φ. If φ contains no free variables, φ is a *sentence* and $[\![\varphi]\!]:(\Sigma^+,M)\to K$.

▶ **Example 14.** Let us go back to the automaton $\mathcal{A}_{\mathrm{itr}}$ depicted in Fig. 3. The following boolean formula β defines three subsets of string positions, X_0,X_1,X_2, representing, respectively, the string portions where unmatched calls are not penalized, namely X_0,X_2, and the portion where they are, namely X_1:

$$
\begin{aligned}
\beta = \quad & x\in X_0 \leftrightarrow \exists y\exists z(y>x\wedge z>x\wedge \mathrm{Lab}_{\$}(y)\wedge\mathrm{Lab}_{\$}(z)) \\
& \wedge x\in X_1 \leftrightarrow \exists y\exists z(y\le x\le z\wedge \mathrm{Lab}_{\$}(y)\wedge\mathrm{Lab}_{\$}(z)\wedge(x\neq y\wedge x\neq z\to\neg\mathrm{Lab}_{\$}(x))) \\
& \wedge x\in X_2 \leftrightarrow \exists y\exists z(y<x\wedge z<x\wedge \mathrm{Lab}_{\$}(y)\wedge\mathrm{Lab}_{\$}(z))\ .
\end{aligned}
$$

Weight assignment is formalized by the formula φ which assigns weight 0 to calls, returns, and interrupts outside portion X_1; and weights $1, -1, 0$ to calls, returns, and interrupts, respectively, within portion X_1:

$$\begin{aligned}\varphi = \quad & (\neg((x \in X_0 \vee x \in X_2) \wedge (\mathrm{Lab}_{\mathrm{call}}(x) \vee \mathrm{Lab}_{\mathrm{ret}}(x) \vee \mathrm{Lab}_{\mathrm{itr}}(x))) \oplus 0) \\ & \otimes (\neg(x \in X_1 \wedge \mathrm{Lab}_{\mathrm{call}}(x)) \oplus 1) \otimes (\neg(x \in X_1 \wedge \mathrm{Lab}_{\mathrm{ret}}(x)) \oplus -1) \\ & \otimes (\neg(x \in X_1 \wedge \mathrm{Lab}_{\mathrm{itr}}(x)) \oplus 0) \otimes (\neg \mathrm{Lab}_{\$}(x) \oplus 0) \ .\end{aligned}$$

Then, the formula $\psi = \prod_x (\beta \otimes \varphi)$ defines the weight assigned by $\mathcal{A}_{\mathrm{itr}}$ to an input string through a single nondeterministic run and finally $\chi = \bigoplus_{X_0} \bigoplus_{X_1} \bigoplus_{X_2} \psi$ defines the global weight of every string in an equivalent way as the one defined by $\mathcal{A}_{\mathrm{itr}}$.

As shown by [11] in the case of words, the full weighted logic is strictly more powerful than weighted automata. A similar example also applies here. Therefore, in the following, we restrict our logic in an appropriate way.

▶ **Definition 15.** The *set of almost boolean formulas* is the smallest set of all formulas of MSO($\mathbb{K}$) containing all $k \in \mathbb{K}$ and all boolean formulas which is closed under $\oplus$ and $\otimes$.

Adapting ideas from [14], we can show by structural induction that almost boolean formulas describe precisely a certain form of wOPA's behaviors, called *OPL step functions*, which are all series S that can be written as $S = \sum_{i=1}^{n} k_i \mathbb{1}_{L_i}$, where L_i are OPL forming a partition of (Σ^+, M) and $k_i \in \mathbb{K}$ for each $i \in \{1, ..., n\}$. Furthermore, OPL step functions are recognizable by rwOPA and are closed under the natural extension of the semiring's $+$ and $\cdot$ to series.

▶ **Definition 16.** We call $\varphi \in \mathrm{MSO}(\mathbb{K})$ *restricted* if for all subformulas $\psi \otimes \theta$ of φ either ψ is almost boolean or all semiring weights occurring in ψ and θ commute elementwise, and additionally, for all subformulas $\prod_x \psi$ of φ, ψ is almost boolean.

In Example 14, the formula β is boolean, the formula φ is almost boolean, and ψ and χ are restricted. Notice that ψ and χ would be restricted even if $\mathbb{K}$ were not commutative.

▶ **Proposition 17.** *Let φ and ψ be two formulas of* MSO($\mathbb{K}$) *such that $[\![\varphi]\!]$ and $[\![\psi]\!]$ are recognizable (resp. strictly recognizable). Then we have*

- *$[\![\varphi \oplus \psi]\!]$, $[\![\sum_x \varphi]\!]$, and $[\![\sum_X \varphi]\!]$ are recognizable (resp. strictly recognizable).*
- *$[\![\varphi \otimes \psi]\!]$ is (resp. strictly) recognizable if $\varphi \otimes \psi$ is a subformula of a restricted formula.*
- *$[\![\prod_x \varphi]\!]$ is strictly recognizable if φ is an almost boolean formula of* MSO($\mathbb{K}$).

Proof (Sketch). Closure under $\oplus$ is dealt with by an usual disjoint union of two wOPA (resp. rwOPA). Closure under restricted $\otimes$ is dealt with by Proposition 10. For the sum quantification, we utilize Proposition 11. The closure under the restricted product quantification is non-trivial, but can be proved by adapting previous techniques to OPL step functions. ◀

Then, by induction on the structure of a weighted formula and using Proposition 17, we get

▶ **Proposition 18.** *For every restricted* MSO($\mathbb{K}$)*-sentence φ, there exists an rwOPA $\mathcal{A}$ with $[\![\mathcal{A}]\!] = [\![\varphi]\!]$.*

Now, we show that the converse of Proposition 18 holds as well.

▶ **Proposition 19.** *For every rwOPA $\mathcal{A}$, there exists a restricted* MSO($\mathbb{K}$)*-sentence φ with $[\![\mathcal{A}]\!] = [\![\varphi]\!]$. If $\mathbb{K}$ is commutative, then for every wOPA $\mathcal{A}$, there exists a restricted* MSO($\mathbb{K}$)*-sentence φ with $[\![\mathcal{A}]\!] = [\![\varphi]\!]$.*

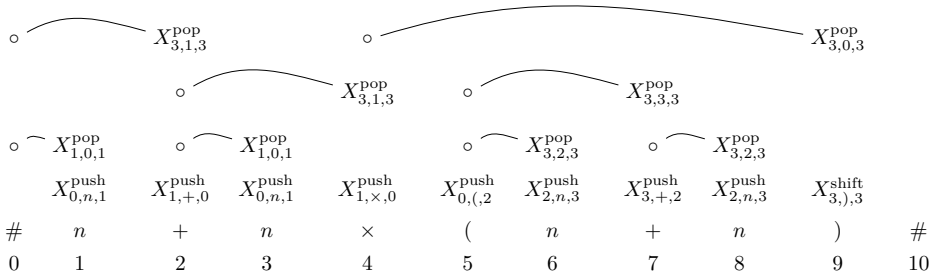


Figure 7 The string of Fig. 1 with the 2nd order variables evidenced for the automaton of Fig. 2.

Proof. The rationale adopted to build formula φ from $\mathcal{A}$ integrates the approach followed in [11, 15] with the one of [25]. On the one hand we need second order variables suitable to "carry" weights; on the other hand, unlike previous non-OP cases which are managed through real-time automata, an OPA can perform several transitions while remaining in the same position. Thus, we introduce the following second order variables: $X^{\text{push}}_{p,a,q}$ represents the set of positions where $\mathcal{A}$ performs a push move from state p, reading symbol a and reaching state q; $X^{\text{shift}}_{p,a,q}$ has the same meaning as $X^{\text{push}}_{p,a,q}$ for a shift operation; and $X^{\text{pop}}_{p,q,r}$ represents the set of positions of the symbol that is on top of the stack when $\mathcal{A}$ performs a pop transition from state p, with q on top of the stack, reaching r.

Let $\mathcal{V}$ consist of all $X^{\text{push}}_{p,a,q}$, $X^{\text{shift}}_{p,a,q}$, and $X^{\text{pop}}_{p,q,r}$ such that $a \in \Sigma$, $p, q, r \in Q$ and $(p, a, q) \in \delta_{\text{push}}$, resp. δ_{shift}, resp. $(p, q, r) \in \delta_{\text{pop}}$. We denote by $\bar{X}^{\text{push}}$, $\bar{X}^{\text{shift}}$, and $\bar{X}^{\text{pop}}$ enumerations over the respective set of second order variables. Using usual abbreviations for MSO-formulas and some adapted shortcuts from [25] and [11], we define the following unweighted formula ψ to characterize all accepted runs of $\mathcal{A}$

$$\psi = \mathit{Part}(\bar{X}^{\text{push}}, \bar{X}^{\text{shift}}) \wedge \mathit{Unique}(\bar{X}^{\text{pop}}) \wedge \mathit{InitFinal} \wedge \mathit{Tr}_{\text{push}} \wedge \mathit{Tr}_{\text{shift}} \wedge \mathit{Tr}_{\text{pop}} \ .$$

Here, the subformula *Part* will enforce the push and shift sets to be (together) a partition of all positions, while the *Unique* will make sure that we mark every position with at most one X^{pop}. *InitFinal* controls the initial and the acceptance condition and $\mathit{Tr}_{\text{push}}$, $\mathit{Tr}_{\text{shift}}$, and Tr_{pop} the respective transitions of the run according to their labels as follows.

$$\begin{aligned}
\mathit{Tr}_{\text{push}} &= \forall x. \textstyle\bigwedge_{p,q\in Q, a\in\Sigma} \left(x \in X^{\text{push}}_{p,a,q} \rightarrow \left[\text{Lab}_a(x) \wedge \exists z.(z \lessdot x \wedge (\text{Next}_p(z,x) \vee \text{Succ}_p(z,x)))\right]\right) \\
\mathit{Tr}_{\text{shift}} &= \forall x. \textstyle\bigwedge_{p,q\in Q, a\in\Sigma} \left(x \in X^{\text{shift}}_{p,a,q} \rightarrow \left[\text{Lab}_a(x) \wedge \exists z.(z \doteq x \wedge (\text{Next}_p(z,x) \vee \text{Succ}_p(z,x)))\right]\right) \\
\mathit{Tr}_{\text{pop}} &= \forall v. \textstyle\bigwedge_{p,q\in Q} \left(\left[\bigvee_{r\in Q} v \in X^{\text{pop}}_{p,q,r}\right] \leftrightarrow \left[\exists x \exists y \exists z.(\text{Tree}_{p,q}(x,z,v,y))\right]\right) \ .
\end{aligned}$$

The main idea is that for every $x \curvearrowright y$, we encode in $\text{Tree}(x, z, v, y)$ the two other 'critical' positions for this chain, namely z, which is the (either direct or hierarchical) successor of x in this chain and which is the position where we execute the push resulting from $x \lessdot z$; and v, which is the 'chain-predecessor' of y and the position we mark with the respective X^{pop} resulting from $v \gtrdot y$. E.g., with reference to Fig. 1 and Fig. 7, we have $\text{Tree}(4, 5, 9, 10)$.

Furthermore, $\text{Succ}_q(x, y)$ holds for two successive positions where the OPA reaches state q through a push or shift at position y, while $\text{Next}_q(x, y)$ holds when a pop move reaches state q while completing a chain $x \curvearrowright y$. Then $\text{Tree}_{p,q}$ explicitly controls the current state and the state on top of the stack when the pop move is executed as follows.

$$\begin{aligned}
\mathrm{Tree}(x,z,v,y) &:= x \curvearrowright y \wedge \begin{pmatrix} (x+1=z \ \vee \ x \curvearrowright z) \wedge \neg\exists t(z<t<y \wedge x \curvearrowright t) \wedge \\ (v+1=y \ \vee \ v \curvearrowright y) \wedge \neg\exists t(x<t<v \wedge t \curvearrowright y) \end{pmatrix} \\
\mathrm{Next}_r(x,y) &:= \exists z \exists v. \left(\mathrm{Tree}(x,z,v,y) \wedge \bigvee\nolimits_{p,q\in Q} v \in X^{\mathrm{pop}}_{p,q,r}\right) \\
\mathrm{Tree}_{i,j}(x,z,v,y) &:= \mathrm{Tree}(x,z,v,y) \wedge (\mathrm{Succ}_i(v,y) \vee \mathrm{Next}_i(v,y)) \wedge (\mathrm{Succ}_i(x,z) \vee \mathrm{Next}_i(x,z))
\end{aligned}$$

Notice that in the transition formulas, the partition (resp. uniqueness) axioms guarantee that in every run, the left side of the implication (resp. equivalence) is satisfied for only one triple (p,a,q), resp. (p,q,r). Thus, with arguments similar to [25], it can be shown that the sentences satisfying ψ are exactly those accepted by the unweighted OPA subjacent to $\mathcal{A}$.

Now, we add weights to ψ by defining the following restricted weighted formula

$$\begin{aligned}
\theta = \psi \otimes \prod_x \bigotimes_{p,q\in Q} \Big(& \bigotimes_{a\in\Sigma} (x \in X^{\mathrm{push}}_{p,a,q} \otimes \mathrm{wt}_{\mathrm{push}}(p,a,q)) \oplus (\neg(x \in X^{\mathrm{push}}_{p,a,q}) \otimes 1) \\
& \otimes \bigotimes_{a\in\Sigma} (x \in X^{\mathrm{shift}}_{p,a,q} \otimes \mathrm{wt}_{\mathrm{push}}(p,a,q)) \oplus (\neg(x \in X^{\mathrm{shift}}_{p,a,q}) \otimes 1) \\
& \otimes \bigotimes_{r\in Q} (x \in X^{\mathrm{pop}}_{p,q,r} \otimes \mathrm{wt}_{\mathrm{push}}(p,q,r)) \oplus (\neg(x \in X^{\mathrm{pop}}_{p,q,r}) \otimes 1)\Big) \ .
\end{aligned}$$

Here, the second part of θ multiplies up all weights of the encountered transitions. This is the crucial part where we either need that $\mathbb{K}$ is commutative or all pop weights are trivial because the product quantifier of θ assigns the pop weight at a different position than the occurrence of the respective pop transition in the automaton. Using only one product quantifier (weighted universal quantifier) this is unavoidable, since the number of pops at a given position is only bounded by the word length.

Since the subformulas $x \in X^{()}_{()} \otimes \mathrm{wt}(...)$ of θ are almost boolean, the subformula $\prod_x(...)$ of θ is $\prod$-restricted. Also, ψ is boolean and so θ is $\otimes$-restricted. Thus, θ is a restricted formula. Finally, we define $\varphi = \bigoplus_{X_1} \bigoplus_{X_2} ... \bigoplus_{X_m} \theta$. This implies $[\![\varphi]\!](w) = [\![\mathcal{A}]\!](w)$, for all $w \in (\Sigma^+, M)$. Therefore, φ is our required restricted sentence with $[\![\mathcal{A}]\!] = [\![\varphi]\!]$. ◀

By Proposition 18 and Proposition 19, we obtain the main result of this section.

▶ **Theorem 20.** *Let $\mathbb{K}$ be a semiring and $S : (\Sigma^+, M) \to K$ a series.*

1. *The following are equivalent:*
   - **(a)** *$S = [\![\mathcal{A}]\!]$ for some rwOPA.*
   - **(b)** *$S = [\![\varphi]\!]$ for some restricted sentence φ of* MSO($\mathbb{K}$).
2. *Let $\mathbb{K}$ be commutative. Then, the following are equivalent:*
   - **(a)** *$S = [\![\mathcal{A}]\!]$ for some wOPA.*
   - **(b)** *$S = [\![\varphi]\!]$ for some restricted sentence φ of* MSO($\mathbb{K}$).

Theorem 20 shows that the typical logical characterization of weighted languages does not generalize in the same way to the whole class wOPL: for non-rwOPL we need the extra hypothesis that $\mathbb{K}$ be commutative. Notice, however, that rwOPL may execute unbounded pop sequences; thus, they are powerful enough to include languages that are neither real-time nor visible. This remark naturally raises new intriguing questions which we will briefly address in the conclusion.

6 Conclusion

This paper moves a further step in the path of generalizing a series of results beyond the barrier of regular and structured – or visible – CFL [27, 33, 2, 25]. We introduced and

investigated weighted operator precedence automata and a corresponding weighted MSO logic. In our main results we show, for any semiring, that wOPA without pop weights and a restricted weighted MSO logic have the same expressive power. Furthermore, these behaviors can also be described as homomorphic images of the behaviors of particularly simple wOPA reduced to arbitrary unweighted OPA. If the semiring is commutative, these results apply also to wOPA with arbitrary pop weights.

Theorem 20 also raises the problems to find, for arbitrary semirings and for wOPA with pop weights, both an expressively equivalent weighted MSO logic and a Nivat-type result. In [16], very similar problems arose for weighted automata on unranked trees and weighted MSO logic. In [12], the authors showed that with another definition of the behavior of weighted unranked tree automata, an equivalence result for the restricted weighted MSO logic could be derived. Is there another definition of the behavior of wOPA (with pop weights) making them expressively equivalent to our restricted weighted MSO logic?

In [25], OPL of infinite words were investigated and shown to be practically important, so the problem arises to develop a theory of wOPA on infinite words. In order to define their quantitative behaviors, one could try to use valuation monoids as in [14, 9].

Finally, a new investigation field can be opened by exploiting the natural suitability of OPL towards parallel elaboration [3]. Computing weights, in fact, can be seen as a special case of semantic elaboration which can be performed hand-in-hand with parsing. In this case too, we can expect different challenges depending on whether the weight semiring is commutative or not and/or weights are attached to pop transitions too, which would be the natural way to follow the traditional semantic evaluation through synthesized attributes [22].

References

1 Rajeev Alur and Dana Fisman. Colored nested words. In Adrian Horia Dediu, Jan Janousek, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications, LATA 2016*, volume 9618 of *LNCS*, pages 143–155. Springer, 2016.

2 Rajeev Alur and Parthasarathy Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009.

3 Alessandro Barenghi, Stefano Crespi Reghizzi, Dino Mandrioli, Federica Panella, and Matteo Pradella. Parallel parsing made practical. *Sci. Comput. Program.*, 112(3):195–226, 2015. doi:10.1016/j.scico.2015.09.002.

4 Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages*, volume 12 of *EATCS Monographs in Theoretical Computer Science*. Springer, 1988.

5 Benedikt Bollig and Paul Gastin. Weighted versus probabilistic logics. In Volker Diekert and Dirk Nowotka, editors, *Developments in Language Theory, DLT 2009*, volume 5583 of *LNCS*, pages 18–38. Springer, 2009. doi:10.1007/978-3-642-02737-6_2.

6 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik und Grundlagen Math.*, 6:66–92, 1960.

7 Christian Choffrut, Andreas Malcher, Carlo Mereghetti, and Beatrice Palano. First-order logics: some characterizations and closure properties. *Acta Inf.*, 49(4):225–248, 2012.

8 Stefano Crespi Reghizzi and Dino Mandrioli. Operator precedence and the visibly pushdown property. *J. Comput. Syst. Sci.*, 78(6):1837–1867, 2012.

9 Manfred Droste and Stefan Dück. Weighted automata and logics for infinite nested words. *Inf. Comput.*, 253:448–466, 2017. doi:10.1016/j.ic.2016.06.010.

10 Manfred Droste, Stefan Dück, Dino Mandrioli, and Matteo Pradella. Weighted operator precedence languages. *CoRR*, abs/1702.04597, 2017. URL: http://arXiv.org/abs/1702.04597.

11 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007. extended abstract in ICALP 2005. doi:10.1016/j.tcs.2007.02.055.

12 Manfred Droste, Doreen Heusel, and Heiko Vogler. Weighted unranked tree automata over tree valuation monoids and their characterization by weighted logics. In Andreas Maletti, editor, *Conference Algebraic Informatics CAI 2015*, volume 9270 of *LNCS*, pages 90–102. Springer, 2015. doi:10.1007/978-3-319-23021-4_9.

13 Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata*. EATCS Monographs in Theoretical Computer Science. Springer, 2009.

14 Manfred Droste and Ingmar Meinecke. Weighted automata and weighted MSO logics for average and long-time behaviors. *Inf. Comput.*, 220:44–59, 2012. doi:10.1016/j.ic.2012.10.001.

15 Manfred Droste and Bundit Pibaljommee. Weighted nested word automata and logics over strong bimonoids. *Int. J. Found. Comput. Sci.*, 25(5):641–666, 2014. doi:10.1142/S0129054114500269.

16 Manfred Droste and Heiko Vogler. Weighted tree automata and weighted logics. *Theor. Comput. Sci.*, 366(3):228–247, 2006. doi:10.1016/j.tcs.2006.08.025.

17 Samuel Eilenberg. *Automata, Languages, and Machines*, volume 59-A of *Pure and Applied Mathematics*. Academic Press, 1974.

18 Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Am. Math. Soc.*, 98(1):21–52, 1961.

19 E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B*, pages 995–1072. MIT Press, 1990.

20 Robert W. Floyd. Syntactic analysis and operator precedence. *J. ACM*, 10(3):316–333, 1963.

21 D. Grune and C. J. Jacobs. *Parsing techniques: a practical guide*. Springer, New York, 2008.

22 Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

23 Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*, volume 6 of *EATCS Monographs in Theoretical Computer Science*. Springer, 1986.

24 Clemens Lautemann, Thomas Schwentick, and Denis Thérien. Logics for context-free languages. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, Selected Papers*, volume 933 of *LNCS*, pages 205–216. Springer, 1994.

25 Violetta Lonati, Dino Mandrioli, Federica Panella, and Matteo Pradella. Operator precedence languages: Their automata-theoretic and logic characterization. *SIAM J. Comput.*, 44(4):1026–1088, 2015. doi:10.1137/140978818.

26 Christian Mathissen. Weighted logics for nested words and algebraic formal power series. *Logical Methods in Computer Science*, 6(1), 2010. Selected papers of ICALP 2008.

27 Robert McNaughton. Parenthesis grammars. *J. ACM*, 14(3):490–500, 1967.

28 Robert McNaughton and Seymour Papert. *Counter-free Automata*. MIT Press, Cambridge, USA, 1971.

29 Kurt Mehlhorn. Pebbling mountain ranges and its application of DCFL-recognition. In *Automata, Languages and Programming, ICALP 1980*, volume 85 of *LNCS*, pages 422–435, 1980.

30 Maurice Nivat. Transductions des langages de Chomsky. *Ann. de l'Inst. Fourier*, 18:339–455, 1968.

31 Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science. Springer, 1978.

32 Marcel Paul Schützenberger. On the definition of a family of automata. *Inf. Control*, 4(2-3):245–270, 1961.

33 James Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journ. of Comp. and Syst.Sc.*, 1:317–322, 1967.

34 Boris A. Trakhtenbrot. Finite automata and logic of monadic predicates (in Russian). *Doklady Akademii Nauk SSR*, 140:326–329, 1961.

35 Burchard von Braunmühl and Rutger Verbeek. Input-driven languages are recognized in log n space. In *Proceedings of the Symposium on Fundamentals of Computation Theory*, volume 158 of *LNCS*, pages 40–51. Springer, 1983.

Model Checking and Validity in Propositional and Modal Inclusion Logics*

Lauri Hella[1], Antti Kuusisto[2], Arne Meier[3], and Jonni Virtema[4]

1 **University of Tampere, Finland**
`lauri.hella@uta.fi`
2 **University of Bremen, Germany**
`antti.j.kuusisto@gmail.com`
3 **Leibniz Universität Hannover, Germany**
`meier@thi.uni-hannover.de`
4 **University of Helsinki, Helsinki, Finland**
`jonni.virtema@helsinki.fi`

Abstract

Propositional and modal inclusion logic are formalisms that belong to the family of logics based on team semantics. This article investigates the model checking and validity problems of these logics. We identify complexity bounds for both problems, covering both lax and strict team semantics. By doing so, we come close to finalising the programme that ultimately aims to classify the complexities of the basic reasoning problems for modal and propositional dependence, independence, and inclusion logics.

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

Keywords and phrases Inclusion Logic, Model Checking, Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.32

1 Introduction

Team semantics is the mathematical framework of modern logics of dependence and independence, which, unlike Tarski semantics, is not based on singletons as satisfying elements (e.g., first-order assignments or points of a Kripke structure) but on sets of such elements. More precisely, a first-order team is a set of first-order assignments that have the same domain of variables. As a result, a team can be interpreted as a database table, where variables correspond to attributes and assignments to records. Team semantics originates from the work of Hodges [17], where it was shown that Hintikka's IF-logic can be based on a compositional (as opposed to game-theoretic) semantics. In 2007, Väänänen [24] proposed a fresh approach to logics of dependence and independence. Väänänen adopted team semantics as a core notion for his *dependence logic*. Dependence logic extends first-order logic by atomic statements such as *the value of variable x is determined by the value of y*. Such a statement is not meaningful under a single assignment, however, when evaluated over a team, such a statement corresponds precisely to functional dependence of database theory when the team is interpreted as a database table.

* The second and the last author acknowledges support from Jenny and Antti Wihuri Foundation. The last author is also supported by the grant 292767 of the Academy of Finland. The third author is supported by the DFG grant ME 4279/1-1. We thank the anonymous referees for their comments.

Besides functional dependence, there are many other important dependency notions used in fields like statistics and database theory, which give rise to interesting logics based on team semantics. The two most widely studied of these new logics are *independence logic* of Grädel and Väänänen [10], and *inclusion logic* of Galliani [5]. Inclusion logic extends first-order logic by atomic statements of the form $x \subseteq y$, which is satisfied in a team X if any value that appears as a value for x in X also appears as a value of y in X. Dependence and independence logics are equi-expressive with existential second-order logic and thus capture the complexity class NP [24, 10]. Surprisingly, inclusion logic has the same expressive power as *positive greatest fixed point logic* GFP^+ [7]. Since on finite structures, GFP^+ coincides with *least fixed point logic* LFP, it follows from the Immermann-Vardi-Theorem that inclusion logic captures the complexity class P on finite ordered structures. Interestingly under a semantical variant of inclusion logic called *strict semantics* the expressive power of inclusion logic rises to existential second-order logic [6]. Moreover, the fragment of inclusion logic (under strict semantics) in which only k universally quantified variables may occur captures the complexity class $\mathsf{NTIME}_{\mathsf{RAM}}(n^k)$ (i.e., structures that can be recognised by a nondeterministic random access machine in time $\mathcal{O}(n^k)$) [11]. The above characterisations exemplify that, indeed, inclusion logic and its fragments have very compelling descriptive complexity-theoretic properties.

In this paper, we study propositional and modal inclusion logic under both the standard semantics (i.e., *lax semantics*) and strict semantics. The research around propositional and modal logics with team semantics has concentrated on classifying the complexity and definability of the related logics. Due to very active research efforts, the complexity and definability landscape of these logics is understood rather well; see the survey of Durand et al. [4] and the references therein for an overview of the current state of the research. In the context of propositional logic (modal logic, resp.) a team is a set of propositional assignments with a common domain of variables (a subset of the domain a Kripke structure, resp.). *Extended propositional inclusion logic* (*extended modal inclusion logic*, resp.) extends propositional logic (modal logic, resp.) with *propositional inclusion atoms* $\varphi \subseteq \psi$, where φ and ψ are formulae of propositional logic (modal logic, resp.). Inclusion logics have fascinating properties also in the propositional setting. The following definability results hold for the standard lax semantics. A class of team pointed Kripke models is definable in extended modal inclusion logic iff $(\mathfrak{M}, \emptyset)$ is in the class for every model $\mathfrak{M}$, the class is closed under taking unions, and the class is closed under the so-called *team k-bisimulation*, for some finite k [16]. From this, a corresponding characterisation for extended propositional inclusion logic directly follows: a class of propositional teams is definable in extended propositional inclusion logic iff the empty team is in the class, and the class is closed under taking unions. In [21, 22] (global) model definability and frame definability of team based modal logics are studied. It is shown that surprisingly, in both cases, (extended) modal inclusion logic collapses to modal logic.

This paper investigates the complexity of the model checking and the validity problem for propositional and modal inclusion logic. The complexity of the satisfiability problem of modal inclusion logic was studied by Hella et al. [15]. The study on the validity problem of propositional inclusion logic was initiated by Hannula et al. [12], where the focus was on more expressive logics in the propositional setting. Consequently, the current paper directly extends the research effort initiated in these papers. It is important to note that since the logics studied in this paper are not closed under taking negations, the connection between the satisfiability problem and the validity problem fails. In [12] it was shown that, under lax semantics, the validity problem for propositional inclusion logic is coNP-complete. Here we obtain an identical result for the strict semantics. However, surprisingly, for model checking

the picture looks quite different. We establish that whereas the model checking problem for propositional inclusion logic is P-complete under lax semantics, the problem becomes NP-complete for the strict variant. Also surprisingly, for model checking in the modal setting, we obtain the identical results (as in the propositional setting): modal inclusion logic is P-complete under lax semantics and NP-complete under strict semantics. Nevertheless, for the validity problem, the modal variants are much more complex than the propositional ones; we establish coNEXP-hardness for both strict and lax semantics.

2 Propositional logics with team semantics

Let D be a finite, possibly empty set of proposition symbols. A function $s\colon D \to \{0,1\}$ is called an *assignment*. A set X of assignments $s\colon D \to \{0,1\}$ is called a *team*. The set D is the *domain* of X. We denote by 2^D the set of *all assignments* $s\colon D \to \{0,1\}$. If $\vec{p} = (p_1, \ldots, p_n)$ is a tuple of propositions and s is an assignment, we write $s(\vec{p})$ for $(s(p_1), \ldots, s(p_n))$.

Let Φ be a set of proposition symbols. The syntax of propositional logic $\mathsf{PL}(\Phi)$ is given by the following grammar: $\varphi ::= p \mid \neg p \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi)$, where $p \in \Phi$.

We denote by $\models_{\mathsf{PL}}$ the ordinary satisfaction relation of propositional logic defined via assignments in the standard way. Next we give team semantics for propositional logic.

▶ **Definition 1** (Lax team semantics)**.** Let Φ be a set of atomic propositions and let X be a team. The satisfaction relation $X \models \varphi$ is defined as follows.

$$\begin{aligned} X \models p &\quad\Leftrightarrow\quad \forall s \in X : s(p) = 1, \\ X \models \neg p &\quad\Leftrightarrow\quad \forall s \in X : s(p) = 0. \\ X \models (\varphi \wedge \psi) &\quad\Leftrightarrow\quad X \models \varphi \text{ and } X \models \psi. \\ X \models (\varphi \vee \psi) &\quad\Leftrightarrow\quad Y \models \varphi \text{ and } Z \models \psi, \text{ for some } Y, Z \text{ such that } Y \cup Z = X. \end{aligned}$$

The lax team semantics is considered to be the standard semantics for team-based logics. In this paper, we also consider a variant of team semantics called the *strict team semantics*. In strict team semantics, the above clause for disjunction is redefined as follows:

$$X \models_{\mathsf{str}} (\varphi \vee \psi) \;\Leftrightarrow\; Y \models \varphi \text{ and } Z \models \psi, \text{ for some } Y, Z \text{ such that } Y \cap Z = \emptyset \text{ and } Y \cup Z = X.$$

When L denotes a team-based propositional logic, we let $\mathsf{L}_{\mathsf{str}}$ denote the variant of the logic with strict semantics. Moreover, in order to improve readability, for strict semantics we use $\models_{\mathsf{str}}$ instead of $\models$. As a result lax semantics is used unless otherwise specified. The next proposition shows that the team semantics and the ordinary semantics for propositional logic defined via assignments (denoted by $\models_{\mathsf{PL}}$) coincide.

▶ **Proposition 2** ([24])**.** *Let φ be a formula of propositional logic and let X be a propositional team. Then $X \models \varphi$ iff $\forall s \in X : s \models_{\mathsf{PL}} \varphi$.*

The syntax of *propositional inclusion logic* $\mathsf{PInc}(\Phi)$ is obtained by extending the syntax of $\mathsf{PL}(\Phi)$ by the grammar rule $\varphi ::= \vec{p} \subseteq \vec{q}$, where $\vec{p}$ and $\vec{q}$ are finite tuples of proposition variables with the same length. The semantics for propositional inclusion atoms is defined as follows:

$$X \models \vec{p} \subseteq \vec{q} \text{ iff } \forall s \in X \, \exists t \in X : s(\vec{p}) = t(\vec{q}).$$

▶ Remark. *Extended propositional inclusion logic* is the variant of PInc in which inclusion atoms of the form $\vec{\varphi} \subseteq \vec{\psi}$, where $\vec{\varphi}$ and $\vec{\psi}$ are tuples of PL-formulae, are allowed. Observe that this extension does not increase the complexity of the logic and on that account, in this paper, we only consider the non-extended variant.

	p	q	r
s_1	1	0	0
s_2	1	1	1
s_3	0	1	0

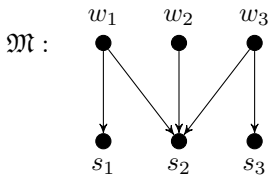


Figure 1 Assignments for teams in Example 4 and the Kripke model for Example 19.

Table 1 Complexity of the satisfiability, validity and model checking problems for propositional logics under both systems of semantics. The shown complexity classes refer to completeness results. † In [15] NEXP-completeness is claimed. However there is a mistake in the proof and the authors of [15] now have a proof for EXP-completeness.

	Satisfiability		Validity		Model checking	
	strict	lax	strict	lax	strict	lax
PL	— NP [3, 19] —		—— coNP [3, 19] ——		——— NC[1] [1] ———	
PInc	EXP†	EXP [15]	coNP [Th. 6]	coNP [12]	NP [Th. 14]	P [Th. 10]

Note that PInc is not a downward closed logic[1]. However, analogously to FO-inclusion-logic [5], satisfaction of PInc-formulas is closed under taking unions.

▶ **Proposition 3** (Closure under unions). *Let $\varphi \in \mathsf{PInc}$ and let X_i, for $i \in I$, be teams. Suppose that $X_i \models \varphi$ for each $i \in I$. Then $\bigcup_{i\in I} X_i \models \varphi$.*

Similarly as in first-order team semantics [5], also for propositional logic the strict and the lax semantics coincide; meaning that $X \models \varphi$ iff $X \models_{\mathsf{str}} \varphi$ for all X and φ. However this does not hold for propositional inclusion logic, for the following example shows that $\mathsf{PInc}_{\mathsf{str}}$ is not union closed. Moreover, we will show that the two different semantics lead to different complexities for the related model checking problems.

▶ **Example 4.** Let s_1, s_2, and s_3 be as in Figure 1 and define $\varphi := \big(p \wedge (p \subseteq r)\big) \vee \big(q \wedge (q \subseteq r)\big)$. Note that $\{s_1, s_2\} \models_{\mathsf{str}} \varphi$ and $\{s_2, s_3\} \models_{\mathsf{str}} \varphi$, but $\{s_1, s_2, s_3\} \not\models_{\mathsf{str}} \varphi$.

However, $\mathsf{PInc}_{\mathsf{str}}$ satisfies a useful weaker form of union closure: it is straightforward to prove by an induction on the formula structure that it is closed under unions of *singleton teams*.

▶ **Lemma 5.** *Let X be a team and $\varphi \in \mathsf{PInc}_{\mathsf{str}}$. If $\{s\} \models_{\mathsf{str}} \varphi$ for every $s \in X$, then $X \models_{\mathsf{str}} \varphi$.*

3 Complexity of propositional inclusion logic

We now define the model checking, satisfiability, and validity problems in the context of team semantics. Let L be a propositional logic with team semantics. A formula $\varphi \in \mathsf{L}$ is *satisfiable*, if there exists a non-empty team X such that $X \models \varphi$. A formula $\varphi \in \mathsf{L}$ is *valid* if $X \models \varphi$ holds for all teams X such that the propositions in φ are in the domain of X. The

[1] A logic L is downward closed if "$X \models \varphi$ and $Y \subseteq X$ implies $Y \models \varphi$" holds for every formula $\varphi \in \mathsf{L}$ and teams X and Y.

satisfiability problem SAT(L) and the validity problem VAL(L) are defined in an obvious way: Given a formula $\varphi \in \mathsf{L}$, decide whether the formula is satisfiable (valid, respectively). For the model checking problem MC(L) we consider combined complexity: Given a formula $\varphi \in \mathsf{L}$ and a team X, decide whether $X \models \varphi$. See Table 1 for known complexity results for PL and PInc, together with partial results of this paper.

It was shown by Hannula et al. [12] that the validity problem of PInc is coNP-complete. Here we establish that the corresponding problem for $\mathsf{PInc}_{\mathsf{str}}$ is also coNP-complete. Our proof is similar to theirs [12], except that instead of union closure we use Lemma 5.

▶ **Theorem 6.** *The validity problem for* $\mathsf{PInc}_{\mathsf{str}}$ *is* coNP*-complete w.r.t.* $\leq_m^{\log}$.

Proof Sketch. The coNP-hardness follows from the fact that PL is a sublogic of $\mathsf{PInc}_{\mathsf{str}}$ and since the validity problem of PL is coNP-hard. On the other hand, by Lemma 5, a formula $\varphi \in \mathsf{PInc}_{\mathsf{str}}$ is valid iff it is satisfied by all singleton teams $\{s\}$. It is easy to see that, over a singleton team $\{s\}$, any inclusion atom is equivalent to a short PL-formula. Consequently, there is a short PL-formula φ^* which is valid iff φ is valid. Since VAL(PL) is in coNP, the same holds for VAL($\mathsf{PInc}_{\mathsf{str}}$). ◀

3.1 Model checking in lax semantics is P-complete

In this section we construct a reduction from the monotone circuit value problem to the model checking problem of PInc. For a deep introduction to circuits see Vollmer [25].

▶ **Definition 7.** A *monotone Boolean circuit* with n input gates and one output gate is a 3-tuple $C = (V, E, \alpha)$, where (V, E) is a finite, simple, directed, acyclic graph, and $\alpha\colon V \to \{\vee, \wedge, x_1, \ldots, x_n\}$ is a function such that the following conditions hold:

1. Every $v \in V$ has in-degree 0 or 2.
2. There exists exactly one $w \in V$ with out-degree 0. We call this node w the *output gate* of C and denote it by g_{out}.
3. If $v \in V$ is a node with in-degree 0, then $\alpha(v) \in \{x_1, \ldots, x_n\}$.
4. If $v \in V$ has in-degree 2, then $\alpha(v) \in \{\vee, \wedge\}$.
5. For each $1 \leq i \leq n$, there exists exactly one $v \in V$ with $\alpha(v) = x_i$.

Let $C = (V, E, \alpha)$ be a monotone Boolean circuit with n input gates and one output gate. Any sequence $b_1, \ldots, b_n \in \{0, 1\}$ of bits of length n is called an *input* to the circuit C. A function $\beta\colon V \to \{0, 1\}$ defined such that

$$\beta(v) := \begin{cases} b_i & \text{if } \alpha(v) = x_i \\ \min\big(\beta(v_1), \beta(v_2)\big) & \text{if } \alpha(v) = \wedge, \text{ where } v_1 \neq v_2 \text{ and } (v_1, v), (v_2, v) \in E, \\ \max\big(\beta(v_1), \beta(v_2)\big) & \text{if } \alpha(v) = \vee, \text{ where } v_1 \neq v_2 \text{ and } (v_1, v), (v_2, v) \in E. \end{cases}$$

is called the *valuation of the circuit* C *under the input* $b_1, \ldots, b_n$. The *output of the circuit* C is then defined to be $\beta(g_{\text{out}})$.

The *monotone circuit value problem* (MCVP) is the following decision problem: Given a monotone circuit C and an input $b_1, \ldots, b_n \in \{0, 1\}$, is the output of the circuit 1?

▶ **Proposition 8** ([9])**.** MCVP *is* P*-complete w.r.t.* $\leq_m^{\log}$ *reductions.*

▶ **Lemma 9.** MC(PInc) *under lax semantics is* P*-hard w.r.t.* $\leq_m^{\log}$.

Proof. We will establish a $\leq_m^{\log}$-reduction from MCVP to the model checking problem of PInc under lax semantics. Since MCVP is P-complete, the claim follows. More precisely, we will show how to construct, for each monotone Boolean circuit C with n input gates and for each input $\vec{b}$ for C, a team $X_{C,\vec{b}}$ and a PInc-formula φ_C such that $X_{C,\vec{b}} \models \varphi_C$ iff the output of the circuit C with the input $\vec{b}$ is 1.

We use teams to encode valuations of the circuit. For each gate v_i of a given circuit, we identify an assignment s_i. The crude idea is that if s_i is in the team under consideration, the value of the gate v_i with respect to the given input is 1. The formula φ_C is used to quantify a truth value for each Boolean gate of the circuit, and then for checking that the truth values of the gates propagate correctly. We next define the construction formally.

Let $C = (V, E, \alpha)$ be a monotone Boolean circuit with n input gates and one output gate and let $\vec{b} = (b_1 \dots b_n) \in \{0,1\}^n$ be an input to the circuit C. We define that $V = \{v_0, \dots, v_m\}$ and that v_0 is the output gate of C. Define

$$\tau_C := \{p_0, \dots, p_m, p_\top, p_\bot\} \cup \{p_{k=i\vee j} \mid i < j, \alpha(v_k) = \vee, \text{ and } (v_i, v_k), (v_j, v_k) \in E\}.$$

For each $i \leq m$, we define the assignment $s_i \colon \tau_C \to \{0,1\}$ as follows:

$$s_i(p) := \begin{cases} 1 & \text{if } p = p_i \text{ or } p = p_\top, \\ 1 & \text{if } p = p_{k=i\vee j} \text{ or } p = p_{k=j\vee i} \text{ for some } j,k \leq m, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, we define $s_\bot(p) = 1$ iff $p = p_\bot$ or $p = p_\top$. We note that the assignment $s_\bot$ will be the only assignment that maps $p_\bot$ to 1. We make use of the fact that for each gate v_i of C, it holds that $s_\bot(p_i) = 0$. We define

$$X_{C,\vec{b}} := \big\{s_i \mid \alpha(v_i) \in \{\wedge, \vee\}\big\} \cup \big\{s_i \mid \alpha(v_i) \in \{x_i \mid b_i = 1\}\big\} \cup \{s_\bot\},$$

that is, $X_{C,\vec{b}}$ consists of assignments for each of the Boolean gates, assignments for those input gates that are given 1 as an input, and of the auxiliary assignment $s_\bot$.

Let X be any nonempty subteam of $X_{C,\vec{b}}$ such that $s_\bot \in X$. We have

$$\begin{aligned} &X \models p_\top \subseteq p_0 && \text{iff } s_0 \in X \\ &X \models p_i \subseteq p_j && \text{iff } (s_i \in X \text{ implies } s_j \in X) \\ &X \models p_k \subseteq p_{k=i\vee j} && \text{iff } (i < j, (v_i, v_k), (v_j, v_k) \in E, \alpha(v_k) = \vee \\ & && \quad \text{and } s_k \in X \text{ implies that } s_i \in X \text{ or } s_j \in X) \end{aligned} \tag{1}$$

Recall the intuition that $s_i \in X$ should hold iff the value of the gate v_i is 1. Define

$$\begin{aligned} \psi_{\text{out}=1} &:= p_\top \subseteq p_0, \\ \psi_\wedge &:= \bigwedge\{p_i \subseteq p_j \mid (v_j, v_i) \in E \text{ and } \alpha(p_i) = \wedge\}, \\ \psi_\vee &:= \bigwedge\{p_k \subseteq p_{k=i\vee j} \mid i < j, (v_i, v_k) \in E, (v_j, v_k) \in E, \text{ and } \alpha(v_k) = \vee\}, \\ \varphi_C &:= \neg p_\bot \vee (\psi_{\text{out}=1} \wedge \psi_\wedge \wedge \psi_\vee). \end{aligned}$$

Now observe that $X_{C,\vec{b}} \models \varphi_C$ iff the output of C with the input $\vec{b}$ is 1.

The idea of the reduction is the following: The disjunction in ϕ_C is used to guess a team Y for the right disjunct that encodes the valuation β of the circuit C. The right disjunct is then evaluated with respect to the team Y with the intended meaning that $\beta(v_i) = 1$

whenever $s_i \in Y$. Note that Y is always as required in (1). The formula $\psi_{\text{out}=1}$ is used to state that $\beta(v_0) = 1$, whereas the formulae $\psi_\wedge$ and $\psi_\vee$ are used to propagate the truth value 1 down the circuit. The assignment $s_\bot$ and the proposition $p_\bot$ are used as an auxiliary to make sure that Y is nonempty and to deal with the propagation of the value 0 by the subformulae of the form $p_i \subseteq p_j$. Finally, it is easy to check that the reduction can be computed in logspace. ◀

For the proof of the above lemma it is not important that lax semantics is considered; the same proof works also for the strict semantics. However, as we will show in the next section, we can show a stronger result for the model checking problem of $\mathsf{PInc}_{\mathsf{str}}$; namely that it is NP-hard. In Section 5.1 we will show that the model checking problem for modal inclusion logic with lax semantics is in P (Lemma 21). Since PInc is essentially a fragment of this logic, by combining Lemmas 9 and 21, we obtain the following theorem.

▶ **Theorem 10.** MC(PInc) *under lax semantics is* P*-complete w.r.t.* $\leq_m^{\log}$.

3.2 Model checking in strict semantics is NP-complete

In this section we reduce the set splitting problem, a well-known NP-complete problem, to the model checking problem of $\mathsf{PInc}_{\mathsf{str}}$.

▶ **Definition 11.** The *set splitting* problem is the following decision problem:
Input: A family $\mathcal{F}$ of subsets of a finite set S.
Problem: Do there exist subsets S_1 and S_2 of S such that
1. S_1 and S_2 are a partition of S (i.e., $S_1 \cap S_2 = \emptyset$ and $S_1 \cup S_2 = S$),
2. for each $A \in \mathcal{F}$, there exist $a_1, a_2 \in A$ such that $a_1 \in S_1$ and $a_2 \in S_2$?

▶ **Proposition 12** ([8])**.** *The set splitting problem is* NP*-complete w.r.t.* $\leq_m^{\log}$.

The following proof relies on the fact that strict semantics is considered. It cannot hold for lax semantics unless P = NP.

▶ **Lemma 13.** MC($\mathsf{PInc}_{\mathsf{str}}$) *is* NP*-hard with respect to* $\leq_m^{\log}$.

Proof. We give a reduction from the set splitting problem to the model checking problem of PInc under strict semantics.

Let $\mathcal{F}$ be an instance of the set splitting problem. We stipulate that $\mathcal{F} = \{B_1, \ldots, B_n\}$ and that $\bigcup \mathcal{F} = \{a_1, \ldots, a_k\}$, where $n, k \in \mathbb{N}$. We will introduce fresh propositions p_i and q_j for each point $a_i \in \bigcup \mathcal{F}$ and set $B_j \in \mathcal{F}$. We will then encode the family of sets $\mathcal{F}$ by assignments over these propositions; each assignment s_i will correspond to a unique point a_i. Formally, let $\tau_{\mathcal{F}}$ denote the set $\{p_1, \ldots, p_k, q_1, \ldots, q_n, p_\top, p_c, p_d\}$ of propositions. For each $i \in \{1, \ldots, k, c, d\}$, we define the assignment $s_i \colon \tau_{\mathcal{F}} \to \{0, 1\}$ as follows:

$$s_i(p) := \begin{cases} 1 & \text{if } p = p_i \text{ or } p = p_\top, \\ 1 & \text{if, for some } j,\ p = q_j \text{ and } a_i \in B_j, \\ 0 & \text{otherwise.} \end{cases}$$

Define $X_{\mathcal{F}} := \{s_1, \ldots, s_k, s_c, s_d\}$, that is, $X_{\mathcal{F}}$ consists of assignments s_i corresponding to each of the points $a_i \in \bigcup \mathcal{F}$ and of two auxiliary assignments s_c and s_d. Note that the only assignment in $X_{\mathcal{F}}$ that maps p_c (p_d, resp.) to 1 is s_c (s_d, resp.) and that every assignment

maps $p_\top$ to 1. Moreover, note that for $1 \leq i \leq k$ and $1 \leq j \leq n$, $s_i(q_j) = 1$ iff $a_i \in B_j$. Now define

$$\varphi_{\mathcal{F}} := \big(\neg p_c \wedge \bigwedge_{i \leq n} p_\top \subseteq q_i\big) \vee \big(\neg p_d \wedge \bigwedge_{i \leq n} p_\top \subseteq q_i\big).$$

We claim that $X_{\mathcal{F}} \models_{\mathsf{str}} \varphi_{\mathcal{F}}$ iff the output of the set splitting problem with input $\mathcal{F}$ is "yes". ◀

In Section 5.1 we establish that the model checking problem of modal inclusion logic with strict semantics is in NP (Theorem 24). Since $\mathsf{PInc}_{\mathsf{str}}$ is essentially a fragment of this logic, together with Lemma 13, we obtain the following theorem.

▶ **Theorem 14.** $\mathrm{MC}(\mathsf{PInc}_{\mathsf{str}})$ *is* NP*-complete with respect to* $\leq_m^{\log}$.

4 Modal logics with team semantics

Let Φ be a set of proposition symbols. The syntax of modal logic $\mathsf{ML}(\Phi)$ is generated by the following grammar: $\varphi ::= p \mid \neg p \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \Diamond\varphi \mid \Box\varphi$, where $p \in \Phi$. By $\varphi^\perp$ we denote the formula that is obtained from $\neg\varphi$ by pushing all negation symbols to the atomic level using the standard duality between $\wedge$ ($\Box$) and $\vee$ ($\Diamond$). A (Kripke) Φ-*model* is a tuple $\mathfrak{M} = (W, R, V)$, where W, called the *domain* of $\mathfrak{M}$, is a non-empty set, $R \subseteq W \times W$ is a binary relation, and $V\colon \Phi \to \mathcal{P}(W)$ is a valuation of the proposition symbols. By $\models_{\mathsf{ML}}$ we denote the *satisfaction relation* of modal logic that is defined via pointed Φ-*models* in the standard way. Any subset T of the domain of a Kripke model $\mathfrak{M}$ is called *a team of* $\mathfrak{M}$. Before we define *team semantics* for ML, we introduce some auxiliary notation.

▶ **Definition 15.** Let $\mathfrak{M} = (W, R, V)$ be a model and T and S teams of $\mathfrak{M}$. Define that

$$R[T] := \{w \in W \mid \exists v \in T \text{ s.t. } vRw\} \text{ and } R^{-1}[T] := \{w \in W \mid \exists v \in T \text{ s.t. } wRv\}.$$

For teams T and S of $\mathfrak{M}$, we write $T[R]S$ if $S \subseteq R[T]$ and $T \subseteq R^{-1}[S]$.

Accordingly, $T[R]S$ holds if and only if for every $w \in T$, there exists some $v \in S$ such that wRv, and for every $v \in S$, there exists some $w \in T$ such that wRv. We are now ready to define team semantics for ML.

▶ **Definition 16** (Lax team semantics)**.** Let $\mathfrak{M}$ be a Kripke model and T a team of $\mathfrak{M}$. The satisfaction relation $\mathfrak{M}, T \models \varphi$ for $\mathsf{ML}(\Phi)$ is defined as follows.

$$\begin{aligned}
\mathfrak{M}, T \models p &\;\Leftrightarrow\; w \in V(p) \text{ for every } w \in T.\\
\mathfrak{M}, T \models \neg p &\;\Leftrightarrow\; w \notin V(p) \text{ for every } w \in T.\\
\mathfrak{M}, T \models (\varphi \wedge \psi) &\;\Leftrightarrow\; \mathfrak{M}, T \models \varphi \text{ and } \mathfrak{M}, T \models \psi.\\
\mathfrak{M}, T \models (\varphi \vee \psi) &\;\Leftrightarrow\; \mathfrak{M}, T_1 \models \varphi \text{ and } \mathfrak{M}, T_2 \models \psi \text{ for some } T_1 \text{ and } T_2 \text{ s.t. } T_1 \cup T_2 = T.\\
\mathfrak{M}, T \models \Diamond\varphi &\;\Leftrightarrow\; \mathfrak{M}, T' \models \varphi \text{ for some } T' \text{ s.t. } T[R]T'.\\
\mathfrak{M}, T \models \Box\varphi &\;\Leftrightarrow\; \mathfrak{M}, T' \models \varphi, \text{ where } T' = R[T].
\end{aligned}$$

Analogously to the propositional case, we also consider the *strict* variant of team semantics for modal logic. In the *strict* team semantics, we have the following alternative semantic definitions for the disjunction and diamond (where W denotes the domain of $\mathfrak{M}$).

$$\begin{aligned}
\mathfrak{M}, T \models_{\mathsf{str}} (\varphi \vee \psi) &\;\Leftrightarrow\; \mathfrak{M}, T_1 \models \varphi \text{ and } \mathfrak{M}, T_2 \models \psi\\
&\qquad \text{for some } T_1 \text{ and } T_2 \text{ such that } T_1 \cup T_2 = T \text{ and } T_1 \cap T_2 = \emptyset.\\
\mathfrak{M}, T \models_{\mathsf{str}} \Diamond\varphi &\;\Leftrightarrow\; \mathfrak{M}, f(T) \models \varphi \text{ for some } f\colon T \to W \text{ s.t. } \forall w \in T : wRf(w).
\end{aligned}$$

Table 2 Complexity of satisfiability, validity and model checking for modal logics under both strict and lax semantics. The given complexity classes refer to completeness results and "-h." denotes hardness. The complexities for Minc and EMinc coincide, see Theorems 23, 24, and 26.
† In [15] NEXP-completeness is claimed. However there is a mistake in the proof and the authors of [15] now have a proof for EXP-completeness.

	Satisfiability		Validity		Model checking	
	strict	lax	strict	lax	strict	lax
ML	- PSPACE [18] -		PSPACE [18]		P [2, 23]	
Minc	EXP†	EXP [15]	coNEXP-h. [Th. 25]	coNEXP-h. [Th. 25]	NP [Th. 24]	P [Th. 23]

When L is a team-based modal logic, we let $\mathsf{L}_{\mathsf{str}}$ to denote its variant with strict semantics. As in the propositional case, for strict semantics we use $\models_{\mathsf{str}}$ instead of $\models$. The formulae of ML have the following flatness property.

▶ **Proposition 17** (Flatness, see, e.g., [4]). *Let $\mathfrak{M}$ be a Kripke model and T be a team of $\mathfrak{M}$. Then, for every formula φ of $\mathsf{ML}(\Phi)$: $\mathfrak{M}, T \models \varphi \Leftrightarrow \forall w \in T : \mathfrak{M}, w \models_{\mathsf{ML}} \varphi$.*

The syntax of *modal inclusion logic* $\mathsf{Minc}(\Phi)$ and *extended modal inclusion logic* $\mathsf{EMinc}(\Phi)$ is obtained by extending the syntax of $\mathsf{ML}(\Phi)$ by the following grammar rule for each $n \in \mathbb{N}$:

$$\varphi ::= \varphi_1, \ldots, \varphi_n \subseteq \psi_1, \ldots, \psi_n,$$

where $\varphi_1, \psi_1, \ldots, \varphi_n, \psi_n \in \mathsf{ML}(\Phi)$. Additionally, for $\mathsf{Minc}(\Phi)$, we require that φ_1, ψ_1, …, φ_n, ψ_n are proposition symbols. The semantics for these inclusion atoms is defined as follows:

$$\mathfrak{M}, T \models \varphi_1, \ldots, \varphi_n \subseteq \psi_1, \ldots, \psi_n \Leftrightarrow \forall w \in T \exists v \in T : \bigwedge_{1 \leq i \leq n} (\mathfrak{M}, \{w\} \models \varphi_i \Leftrightarrow \mathfrak{M}, \{v\} \models \psi_i).$$

The following proposition is proven in the same way as the analogous results for first-order inclusion logic [5]. A modal logic L is union closed if $\mathfrak{M}, T \models \varphi$ and $\mathfrak{M}, S \models \varphi$ implies that $\mathfrak{M}, T \cup S \models \varphi$, for every $\varphi \in \mathsf{L}$.

▶ **Proposition 18** (Union Closure). *The logics* ML, Minc, EMinc *are union closed.*

Analogously to the propositional case, it is easy to establish that for ML the strict and the lax semantics coincide (for a proof in the first-order setting see [5]). Again, as in the propositional case, this does not hold for Minc or EMinc. Note that since $\mathsf{PInc}_{\mathsf{str}}$ is not union closed, neither is $\mathsf{Minc}_{\mathsf{str}}$, nor $\mathsf{EMinc}_{\mathsf{str}}$.

In contrary to the propositional case, Lemma 5 fails in the modal case as the following example illustrates.

▶ **Example 19.** Let $\mathfrak{M}$ be as depicted in the table of Figure 1 and let φ denote the $\mathsf{PInc}_{\mathsf{str}}$-formula of Example 4. Now $\mathfrak{M}, \{w_i\} \models_{\mathsf{str}} \Box\varphi$, for $i \in \{1, 2, 3\}$, but $\mathfrak{M}, \{w_1, w_2, w_3\} \not\models_{\mathsf{str}} \Box\varphi$.

5 Model checking and validity in modal team semantics

The model checking, satisfiability, and validity problems in the context of team semantics of modal logic are defined analogously to the propositional case. Let $\mathsf{L}(\Phi)$ be a modal logic with team semantics. A formula $\varphi \in \mathsf{L}(\Phi)$ is *satisfiable*, if there exists a Kripke Φ-model

$\mathfrak{M}$ and a non-empty team T of $\mathfrak{M}$ such that $\mathfrak{M}, T \models \varphi$. A formula $\varphi \in \mathsf{L}(\Phi)$ is *valid*, if $\mathfrak{M}, T \models \varphi$ holds for every Φ-model $\mathfrak{M}$ and every team T of $\mathfrak{M}$. The satisfiability problem SAT(L) and the validity problem VAL(L) are defined in the obvious way: Given a formula $\varphi \in \mathsf{L}$, decide whether the formula is satisfiable (valid, respectively). For model checking MC(L) we consider combined complexity: Given a formula $\varphi \in \mathsf{L}$, a Kripke model $\mathfrak{M}$, and a team T of $\mathfrak{M}$, decide whether $\mathfrak{M}, T \models \varphi$. See Table 2 for known complexity results on ML and Minc, together with partial results of this paper.

5.1 Complexity of model checking

Let $\mathfrak{M}$ be a Kripke model, T be a team of $\mathfrak{M}$, and φ be a formula of Minc. By $\mathrm{maxsub}(T, \varphi)$, we denote the maximum subteam T' of T such that $\mathfrak{M}, T' \models \varphi$. Since Minc is union closed (cf. Proposition 18), such a maximum subteam always exists.

For a proof of the following lemma, see the full version [14] of this article.

▶ **Lemma 20.** *If φ is a proposition symbol, its negation, or an inclusion atom, then $\mathrm{maxsub}(T, \varphi)$ can be computed in polynomial time with respect to $|T| + |\varphi|$.*

For the following lemma it is crucial that lax semantics is considered. The lemma cannot hold for strict semantics unless $\mathsf{P} = \mathsf{NP}$.

▶ **Lemma 21.** MC(Minc) *under lax semantics is in* P.

Proof. We will present a labelling algorithm for model checking $\mathfrak{M}, T \models \varphi$. Let $\mathrm{subOcc}(\varphi)$ denote the set of all *occurrences* of subformulae of φ. Below we denote occurrences as if they were formulae, but we actually refer to some particular occurrence of the formula.

A function $f \colon \mathrm{subOcc}(\varphi) \to \mathcal{P}(W)$ is called a labelling function of φ in $\mathfrak{M}$. We will next give an algorithm for computing a sequence $f_0, f_1, f_2, \ldots$, of such labelling functions.

- Define $f_0(\psi) = W$ for each $\psi \in \mathrm{subOcc}(\varphi)$.
- For odd $i \in \mathbb{N}$, define $f_i(\psi)$ bottom up as follows:
  1. For literal ψ, define $f_i(\psi) := \mathrm{maxsub}(f_{i-1}(\psi), \psi)$.
  2. $f_i(\psi \wedge \theta) := f_i(\psi) \cap f_i(\theta)$.
  3. $f_i(\psi \vee \theta) := f_i(\psi) \cup f_i(\theta)$.
  4. $f_i(\Diamond\psi) := \{w \in f_{i-1}(\Diamond\psi) \mid R[w] \cap f_i(\psi) \neq \emptyset\}$.
  5. $f_i(\Box\psi) := \{w \in f_{i-1}(\Box\psi) \mid R[w] \subseteq f_i(\psi)\}$.
- For even $i \in \mathbb{N}$ larger than 0, define $f_i(\psi)$ top to bottom as follows:
  1. Define $f_i(\varphi) := f_{i-1}(\varphi) \cap T$.
  2. If $\psi = \theta \wedge \gamma$, define $f_i(\theta) := f_i(\gamma) := f_i(\theta \wedge \gamma)$.
  3. If $\psi = \theta \vee \gamma$, define $f_i(\theta) := f_{i-1}(\theta) \cap f_i(\theta \vee \gamma)$ and $f_i(\gamma) := f_{i-1}(\gamma) \cap f_i(\theta \vee \gamma)$.
  4. If $\psi = \Diamond\theta$, define $f_i(\theta) := f_{i-1}(\theta) \cap R[f_i(\Diamond\theta)]$.
  5. If $\psi = \Box\theta$, define $f_i(\theta) := f_{i-1}(\theta) \cap R[f_i(\Box\theta)]$.

By a straightforward induction on i, we can prove that $f_{i+1}(\psi) \subseteq f_i(\psi)$ holds for every $\psi \in \mathrm{subOcc}(\varphi)$. The only nontrivial induction step is that for $f_{i+1}(\theta)$ and $f_{i+1}(\gamma)$, when $i+1$ is even and $\psi = \theta \wedge \gamma$. To deal with this step, observe that, by the definition of f_{i+1} and f_i, we have $f_{i+1}(\theta) = f_{i+1}(\gamma) = f_{i+1}(\psi)$ and $f_i(\psi) \subseteq f_i(\theta), f_i(\gamma)$, and by the induction hypothesis on ψ, we have $f_{i+1}(\psi) \subseteq f_i(\psi)$.

It follows that there is an integer $j \leq 2 \cdot |W| \cdot |\varphi|$ such that $f_{j+2} = f_{j+1} = f_j$. We denote this fixed point f_j of the sequence $f_0, f_1, f_2, \ldots$ by f_∞. By Lemma 20 the outcome of $\mathrm{maxsub}(\cdot, \cdot)$ is computable in polynomial time with respect to its input. That being, clearly f_{i+1} can be computed from f_i in polynomial time with respect to $|W| + |\varphi|$. On that account f_∞ is also computable in polynomial time with respect to $|W| + |\varphi|$.

We will next prove by induction on $\psi \in \mathrm{subOcc}(\varphi)$ that $\mathfrak{M}, f_\infty(\psi) \models \psi$. Note first that there is an odd integer i and an even integer j such that $f_\infty = f_i = f_j$.

1. If ψ is a literal, the claim is true since $f_\infty = f_i$ and $f_i(\psi) = \mathrm{maxsub}(f_{i-1}(\psi), \psi)$.
2. Assume next that $\psi = \theta \wedge \gamma$, and the claim holds for θ and γ. Since $f_\infty = f_j$, we have $f_\infty(\psi) = f_\infty(\theta) = f_\infty(\gamma)$, as a result, by induction hypothesis, $\mathfrak{M}, f_\infty(\psi) \models \theta \wedge \gamma$.
3. In the case $\psi = \theta \vee \gamma$, we obtain the claim $\mathfrak{M}, f_\infty(\psi) \models \psi$ by using the induction hypothesis, and the observation that $f_\infty(\psi) = f_i(\psi) = f_i(\theta) \cup f_i(\gamma) = f_\infty(\theta) \cup f_\infty(\gamma)$.
4. Assume then that $\psi = \Diamond\theta$. Since $f_\infty = f_i$, we have $f_\infty(\psi) = \{w \in f_{i-1}(\psi) \mid R[w] \cap f_\infty(\theta) \neq \emptyset\}$, as a consequence $f_\infty(\psi) \subseteq R^{-1}[f_\infty(\theta)]$. On the other hand, since $f_\infty = f_j$, we have $f_\infty(\theta) = f_{j-1}(\theta) \cap R[f_\infty(\psi)]$, for this reason $f_\infty(\theta) \subseteq R[f_\infty(\psi)]$. Thus $f_\infty(\psi)[R]f_\infty(\theta)$, and using the induction hypothesis, we see that $\mathfrak{M}, f_\infty(\psi) \models \psi$.
5. Assume finally that $\psi = \Box\theta$. Since $f_\infty = f_i$, we have $R[f_\infty(\psi)] \subseteq f_\infty(\theta)$. On the other hand, since $f_\infty = f_j$, we have $f_\infty(\theta) \subseteq R[f_\infty(\psi)]$. This shows that $f_\infty(\theta) = R[f_\infty(\psi)]$, that being the case by the induction hypothesis, $\mathfrak{M}, f_\infty(\psi) \models \psi$.

In particular, if $f_\infty(\varphi) = T$, then $\mathfrak{M}, T \models \varphi$. Consequently, to complete the proof of the lemma, it suffices to prove that the converse implication is true, as well. To prove this, assume that $\mathfrak{M}, T \models \varphi$. Then for each $\psi \in \mathrm{subOcc}(\varphi)$, there is a team T_ψ such that

1. $T_\varphi = T$.
2. If $\psi = \theta \wedge \gamma$, then $T_\psi = T_\theta = T_\gamma$.
3. If $\psi = \theta \vee \gamma$, then $T_\psi = T_\theta \cup T_\gamma$.
4. If $\psi = \Diamond\theta$, then $T_\psi[R]T_\theta$.
5. If $\psi = \Box\theta$, then $T_\theta = R[T_\psi]$.
6. If ψ is a literal, then $\mathfrak{M}, T_\psi \models \psi$.

We prove by induction on i that $T_\psi \subseteq f_i(\psi)$ for all $\psi \in \mathrm{subOcc}(\varphi)$. For $i = 0$, this is obvious, since $f_0(\psi) = W$ for all ψ. Assume next that $i+1$ is odd and the claim is true for i. We prove the claim $T_\psi \subseteq f_i(\psi)$ by induction on ψ.

1. If ψ is a literal, then $f_{i+1}(\psi) = \mathrm{maxsub}(f_i(\psi), \psi)$. Since $\mathfrak{M}, T_\psi \models \psi$, and by induction hypothesis, $T_\psi \subseteq f_i(\psi)$, the claim $T_\psi \subseteq f_{i+1}(\psi)$ is true.
2. Assume that $\psi = \theta \wedge \gamma$. By induction hypothesis on θ and γ, we have $T_\psi = T_\theta \subseteq f_{i+1}(\theta)$ and $T_\psi = T_\gamma \subseteq f_{i+1}(\gamma)$. For this reason, we get $T_\psi \subseteq f_{i+1}(\theta) \cap f_{i+1}(\gamma) = f_{i+1}(\psi)$.
3. The case $\psi = \theta \vee \gamma$ is similar to the previous one; we omit the details.
4. If $\psi = \Diamond\theta$, then $f_{i+1}(\psi) = \{w \in f_i(\psi) \mid R[w] \cap f_{i+1}(\theta) \neq \emptyset\}$. By the two induction hypotheses on i and θ, we have $\{w \in T_\psi \mid R[w] \cap T_\theta \neq \emptyset\} \subseteq f_{i+1}(\psi)$. The claim follows from this, since the condition $R[w] \cap T_\theta \neq \emptyset$ holds for all $w \in T_\psi$.
5. The case $\psi = \Box\theta$ is again similar to the previous one, so we omit the details.

Assume then that $i+1$ is even and the claim is true for i. This time we prove the claim $T_\psi \subseteq f_i(\psi)$ by top to bottom induction on ψ.

1. By assumption, $T_\varphi = T$, whence by induction hypothesis, $T_\varphi \subseteq f_i(\varphi) \cap T = f_{i+1}(\varphi)$.
2. Assume that $\psi = \theta \wedge \gamma$. By induction hypothesis on ψ, we have $T_\psi \subseteq f_{i+1}(\psi)$. Since $T_\psi = T_\theta = T_\gamma$ and $f_{i+1}(\psi) = f_{i+1}(\theta) = f_{i+1}(\gamma)$, this implies that $T_\theta \subseteq f_{i+1}(\theta)$ and $T_\gamma \subseteq f_{i+1}(\gamma)$.
3. Assume that $\psi = \theta \vee \gamma$. Using the fact that $T_\theta \subseteq T_\psi$, and the two induction hypotheses on i and ψ, we see that $T_\theta \subseteq f_i(\theta) \cap T_\psi \subseteq f_i(\theta) \cap f_{i+1}(\psi) = f_{i+1}(\theta)$. Similarly, we see that $T_\gamma \subseteq f_{i+1}(\gamma)$.

4. Assume that $\psi = \Diamond\theta$. By the induction hypothesis on i, we have $T_\theta \subseteq f_i(\theta)$, and by the induction hypothesis on ψ, we have $T_\theta \subseteq R[T_\psi] \subseteq R[f_{i+1}(\psi)]$. Accordingly, we see that $T_\theta \subseteq f_i(\theta) \cap R[f_{i+1}(\psi)] = f_{i+1}(\theta)$.
5. The case $\psi = \Box\theta$ is similar to the previous one; we omit the details.

It follows now that $T = T_\varphi \subseteq f_\infty(\varphi)$. Since $f_\infty(\varphi) \subseteq f_2(\varphi) \subseteq T$, we conclude that $f_\infty(\varphi) = T$. This completes the proof of the implication $\mathfrak{M}, T \models \varphi \;\Rightarrow\; f_\infty(\varphi) = T$. ◀

The following lemma then follows, since in the context of model checking, we may replace modal formulae that appear as parameters in inclusion atoms by fresh proposition symbols with the same extension.

▶ **Lemma 22.** MC(EMinc) *under lax semantics is in* P.

By combining Lemmas 9, 21, and 22, we obtain the following theorem.

▶ **Theorem 23.** MC(Minc) *and* MC(EMinc) *are* P*-complete w.r.t.* $\leq_m^{\log}$.

▶ **Theorem 24.** MC($\mathsf{Minc}_{\mathsf{str}}$) *and* MC($\mathsf{EMinc}_{\mathsf{str}}$) *are* NP*-complete w.r.t.* $\leq_m^{\log}$.

Proof. The NP-hardness follows from the propositional case, i.e., from Lemma 13.

In order to establish inclusion, we note that the obvious brute force algorithm for model checking for EMinc works in NP: For disjunctions and diamonds, we use nondeterminism to guess the correct partitions or successor teams, respectively. Conjunctions are dealt with sequentially and for boxes the unique successor team can be computed by brute force in quadratic time. Checking whether a team satisfies an inclusion atom or a (negated) proposition symbol can be computed by brute force in polynomial time (this also follows directly from Lemma 20). ◀

5.2 Complexity of validity

The following result involves a reduction from a complement problem of the validity problem of dependency quantified Boolean formulas [20]. The details can be found in the full version [14] of this article.

▶ **Theorem 25.** VAL(Minc) *and* VAL($\mathsf{Minc}_{\mathsf{str}}$) *are* coNEXP*-hard w.r.t.* $\leq_m^{\log}$.

While the exact complexities of the problems VAL(Minc) and VAL(EMinc) remain open, it is straightforward to establish that the complexities coincide. In the proof of the theorem below, we introduce fresh proposition symbols for each modal formula that appears as a parameter for an inclusion atom. We then replace these formulas by the fresh proposition symbols and separately force, by using ML, that the extensions of the proposition symbols and modal formulae are the same, respectively. See [14] for a detailed proof.

▶ **Theorem 26.** *Let* C *be a complexity class that is closed under polynomial time reductions. Then* VAL(Minc) *under lax (strict) semantics in complete for* C *if and only if* VAL(EMinc) *under lax (strict) semantics in complete for* C.

6 Conclusion

In this paper, we investigated the computational complexity of model checking and validity for propositional and modal inclusion logic to complete the complexity landscape of these problems in the mentioned logics. In particular, we gave emphasis to the subtle influence of

which semantics is considered: strict or lax. The model checking problem for these logics under strict semantics was shown to be NP-complete and under lax semantics P-complete. The validity problem was shown to be coNP-complete for the propositional strict semantics case, for the lax semantics coNP-completeness was established earlier by Hannula et al. [12]. For the modal case, we obtained a coNEXP lower bound under lax as well as strict semantics. The upper bound is left open for further research. We however established that, if closed under polynomial time reductions, the complexities of VAL(Minc) and VAL(EMinc), and VAL($\mathsf{Minc}_{\mathsf{str}}$) and VAL($\mathsf{EMinc}_{\mathsf{str}}$) coincide, respectively.

We conclude with an open problem. Let ML($\sim$) denote the extension of modal logic by the contradictory negation $\sim$ with the following semantics: $\mathfrak{M}, T \models \sim\varphi$ iff $\mathfrak{M}, T \not\models \varphi$. What is the complexity of VAL(ML($\sim$))? It is known that VAL(PL($\sim$)) is complete for *alternating exponential time with polynomially many alternations* [13]; this is the best known lower bound for VAL(ML($\sim$)). Decidability with non-elementary upper bound can be obtained, e.g., by using team-bisimulation and Hintikka-types; no better upper bound is known.

References

1. S. R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proc. 19th STOC*, pages 123–131, 1987.
2. E. Clarke, E. A. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM ToPLS*, 8(2):244–263, 1986.
3. S. A. Cook. The complexity of theorem proving procedures. In *Proc. 3rd STOC*, pages 151–158, 1971.
4. A. Durand, J. Kontinen, and H. Vollmer. Expressivity and complexity of dependence logic. In S. Abramsky, J. Kontinen, J. Väänänen, and H. Vollmer, editors, *Dependence Logic: Theory and Applications*, pages 5–32. Springer, 2016.
5. P. 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.
6. P. Galliani, M. Hannula, and J. Kontinen. Hierarchies in independence logic. In *Proc. 22nd CSL*, volume 23 of *LIPIcs*, pages 263–280, 2013.
7. P. Galliani and L. Hella. Inclusion logic and fixed point logic. In *Proc. 22nd CSL*, LIPIcs, pages 281–295, 2013. doi:10.4230/LIPIcs.CSL.2013.281.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.
9. L. M. Goldschlager. The monotone and planar circuit value problems are log-space complete for P. *SIGACT News*, 9:25–29, 1977.
10. E. Grädel and J. Väänänen. Dependence and independence. *Studia Logica*, 101(2):399–410, 2013.
11. M. Hannula and J. Kontinen. Hierarchies in independence and inclusion logic with strict semantics. *J. Log. Comput.*, 25(3):879–897, 2015. doi:10.1093/logcom/exu057.
12. M. Hannula, J. Kontinen, J. Virtema, and H. Vollmer. Complexity of propositional independence and inclusion logic. In *Proc. 40th MFCS*, pages 269–280, 2015. doi:10.1007/978-3-662-48057-1_21.
13. M. Hannula, J. Kontinen, J. Virtema, and H. Vollmer. Complexity of propositional logics in team semantics. *CoRR, extended version of [12]*, abs/1504.06135, 2015.
14. L. Hella, A. Kuusisto, A. Meier, and J. Virtema. Model checking and validity in propositional and modal inclusion logics. *CoRR*, abs/1609.06951, 2016.

15 L. Hella, A. Kuusisto, A. Meier, and H. Vollmer. Modal inclusion logic: Being lax is simpler than being strict. In *Proc. 40th MFCS*, pages 281–292, 2015. doi:10.1007/978-3-662-48057-1_22.

16 L. Hella and J. Stumpf. The expressive power of modal logic with inclusion atoms. In *Proc. 6th GandALF*, pages 129–143, 2015. doi:10.4204/EPTCS.193.10.

17 W. Hodges. Compositional semantics for a language of imperfect information. *Logic Journal of the IGPL*, 5(4):539–563, 1997.

18 R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3):467–480, 1977.

19 L. A. Levin. Universal sorting problems. *Problems of Inform. Transm.*, 9:265–266, 1973.

20 G. Peterson, J. Reif, and S. Azhar. Lower bounds for multiplayer noncooperative games of incomplete information. *Computers & Math. with Applications*, 41(7-8):957–992, 2001. doi:10.1016/S0898-1221(00)00333-3.

21 K. Sano and J. Virtema. Characterizing frame definability in team semantics via the universal modality. In *Proc. of WoLLIC 2015*, pages 140–155, 2015.

22 K. Sano and J. Virtema. Characterizing relative frame definability in team semantics via the universal modality. In *Proc. of WoLLIC 2016*, pages 392–409, 2016.

23 P. Schnoebelen. The complexity of temporal logic model checking. In *Proc. 4th AiML*, pages 393–436, 2002.

24 J. Väänänen. *Dependence Logic*. Cambridge University Press, 2007.

25 H. Vollmer. *Introduction to Circuit Complexity – A Uniform Approach*. Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 1999.

Emptiness Problems for Integer Circuits

Dominik Barth[1], Moritz Beck[2], Titus Dose[3], Christian Glaßer[4], Larissa Michler[5], and Marc Technau[6]

1 **Institute of Computer Science, University of Würzburg, Germany**
2 **Institute of Computer Science, University of Würzburg, Germany**
3 **Institute of Computer Science, University of Würzburg, Germany**
4 **Institute of Computer Science, University of Würzburg, Germany**
5 **Institute of Computer Science, University of Würzburg, Germany**
6 **Institute of Computer Science, University of Würzburg, Germany**

Abstract

We study the computational complexity of emptiness problems for circuits over sets of natural numbers with the operations union, intersection, complement, addition, and multiplication. For most settings of allowed operations we precisely characterize the complexity in terms of completeness for classes like NL, NP, and PSPACE. The case where intersection, addition, and multiplication is allowed turns out to be equivalent to the complement of polynomial identity testing (PIT).

Our results imply the following improvements and insights on problems studied in earlier papers. We improve the bounds for the membership problem $\mathrm{MC}(\cup, \cap, {}^{-}, +, \times)$ studied by McKenzie and Wagner 2007 and for the equivalence problem $\mathrm{EQ}(\cup, \cap, {}^{-}, +, \times)$ studied by Glaßer et al. 2010. Moreover, it turns out that the following problems are equivalent to PIT, which shows that the challenge to improve their bounds is just a reformulation of a major open problem in algebraic computing complexity:

- membership problem $\mathrm{MC}(\cap, +, \times)$ studied by McKenzie and Wagner 2007
- integer membership problems $\mathrm{MC}_{\mathbb{Z}}(+, \times)$ and $\mathrm{MC}_{\mathbb{Z}}(\cap, +, \times)$ studied by Travers 2006
- equivalence problem $\mathrm{EQ}(+, \times)$ studied by Glaßer et al. 2010

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases computational complexity, integer expressions, integer circuits, polynomial identity testing

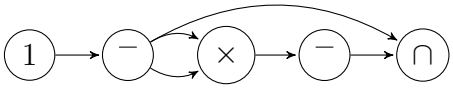
Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.33

1 Introduction

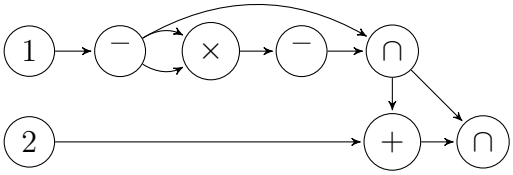
Stockmeyer and Meyer [31] investigated membership and equivalence problems for *integer expressions*, which are built up from single natural numbers using set operations $({}^{-}, \cup, \cap)$ and pairwise addition $(+)$. They also suggested to study expressions involving pairwise multiplication $(\times)$. For example, the expression $\overline{\overline{1} \times \overline{1} \cap 1}$ describes the set of primes $\mathbb{P}$.

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

Wagner [33] introduced *circuits over sets of natural numbers*. These circuits describe expressions in a succinct way. The input gates of such a circuit are labeled with natural numbers, the inner gates compute set operations $({}^{-}, \cup, \cap)$ and arithmetic operations $(+, \times)$. The following circuit has only 4 inner gates and describes the set of primes $\overline{\overline{1} \times \overline{1} \cap 1}$.



A slightly larger circuit describes the set $\{n \in \mathbb{P} \mid n-2 \in \mathbb{P}\}$, i.e., the set of those twin primes p for which $p-2$ is also prime. Hence the set described by this circuit is infinite if and only if the twin prime conjecture holds.



Wagner [33], Yang [34], and McKenzie and Wagner [22] studied the complexity of membership problems for circuits over natural numbers (MC): Here, for a given circuit C with numbers assigned to the input gates, one has to decide whether a given number b belongs to the set described by the circuit. Travers [32] and Breunig [6] considered membership problems for circuits over integers ($\mathrm{MC}_{\mathbb{Z}}$) and positive integers ($\mathrm{MC}_{\mathbb{N}^+}$), respectively. Glaßer et al [11] investigated *equivalence problems for circuits over sets of natural numbers* (EQ), i.e., the problem of deciding whether two given circuits compute the same set.

Satisfiability problems for circuits over sets of natural numbers, studied by Glaßer et al [13], are a generalization of the membership problems investigated by McKenzie and Wagner [22]: Here the circuits can have *unassigned input gates*. The question is, given a circuit C with gates from $\mathcal{O} \subseteq \{\cup, \cap, \bar{}, +, \times\}$, and given a natural number b, does there exist an assignment of natural numbers to the unassigned input gates such that b is contained in the set described by the circuit?

Apart from the mentioned research on circuit problems there has been work on related variants like functions computed by circuits [24] and constraint satisfaction problems over natural numbers [12, 8].

In the present paper, we study *emptiness problems for circuits over sets of natural numbers*. In contrast to membership and satisfiability problems, here the question is whether a given circuit C with gates from $\mathcal{O} \subseteq \{\cup, \cap, \bar{}, +, \times\}$ computes the empty set. We denote this problem with $\mathrm{EC}(\mathcal{O})$. In extension of that, we also consider circuits with unassigned input gates. For these we consider the problem $\Sigma_1\text{-EC}(\mathcal{O})$ (resp., $\Pi_1\text{-EC}(\mathcal{O})$), which asks whether the circuit computes the empty set for at least one assignment (resp., for all assignments).

Our contribution to emptiness problems. For most of the emptiness problems we precisely characterize the complexity in terms of completeness for classes like NL, P, NP, PSPACE, and coNEXP. In the remaining cases we obtain lower and upper bounds that do not match. Our results are summarized in Table 1 in Section 6.

The case of $\mathrm{EC}(\cap, +, \times)$ is particularly interesting. We show that it is logspace many-one equivalent to the complement of the polynomial identity testing (PIT), which asks whether a polynomial (given as a circuit) is identically zero. The problems are similar, still the proof of $\overline{\mathrm{PIT}} \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\cap, +, \times)$ has to address two essential differences: First, PIT contains a universal quantifier (for all assignments the polynomial has to be zero), while $\mathrm{EC}(\cap, +, \times)$ does not. Second, PIT is defined over $\mathbb{Z}$, while $\mathrm{EC}(\cap, +, \times)$ is defined over $\mathbb{N}$.

In several cases we obtain upper bounds for $\Sigma_1\text{-EC}(\mathcal{O})$ and $\Pi_1\text{-EC}(\mathcal{O})$ by observing that if *some* assignment makes a circuit (non-)empty, then there exists a *small* such assignment.

Depending on $\mathcal{O}$ we obtain this observation by one of the following techniques: The first technique (e.g., used for Π_1-EC$(\cap,+) \in$ coNP in Theorem 6) uses specific systems of linear equations that consist of a large number of short equations. Such systems of equations have small solutions by the theory of integer programming. The second technique (e.g., used for EC$(\cap,+,\times) \equiv_{\mathrm{m}}^{\log} \Sigma_1$-EC$(\cap,+,\times)$ in Corollary 21) exploits the fact that the test of whether a multivariate polynomial is identically zero is possible by evaluating this polynomial for a fixed argument. The third technique (e.g., used for Σ_1-EC$(\cup,\cap,^-,+) \in$ 2EXPSPACE and Σ_1-EC$(\cup,\cap,^-,\times) \in$ 3EXPSPACE in Theorem 8) applies the decidability of Presburger and Skolem arithmetic.

Regarding the most general case EC$(\cup,\cap,^-,+,\times)$ we show that this problem is logspace many-one equivalent to MC$(\cup,\cap,^-,+,\times)$ and EQ$(\cup,\cap,^-,+,\times)$, belongs to $\mathcal{R}_{\mathrm{tt}}(\Sigma_1)$, and is $\leq_{\mathrm{m}}^{\log}$-hard for $\mathrm{L}^{\mathrm{NEXP}}$. We leave open whether EC$(\cup,\cap,^-,+,\times)$ is decidable and give evidence for the difficulty of finding a decision algorithm.

Our contribution to questions from previous work. By the equivalence mentioned above, our bounds for EC$(\cup,\cap,^-,+,\times)$ improve the bounds for the problems MC$(\cup,\cap,^-,+,\times)$ [22] and EQ$(\cup,\cap,^-,+,\times)$ [11] as follows. The lower bound is raised from NEXP to $\mathrm{L}^{\mathrm{NEXP}}$ and the upper bound is slightly reduced from $\mathcal{R}_{\mathrm{T}}(\Sigma_1)$ to $\mathcal{R}_{\mathrm{tt}}(\Sigma_1)$.

We prove that PIT is logspace many-one equivalent to MC$(\cap,+,\times)$ studied in [22], MC$_{\mathbb{Z}}(+,\times)$, MC$_{\mathbb{Z}}(\cap,+,\times)$ studied in [32], and EQ$(+,\times)$ studied in [11]. This characterizes the complexity of these problems and shows that the question for improved bounds is equivalent to a well-studied, open problem in algebraic computing complexity.

Finally we show that EQ$(\cap,+,\times)$ is $\leq_{\mathrm{m}}^{\log}$-complete for the complement of the second level of the Boolean hierarchy over PIT. This characterizes the complexity of this equivalence problem and explains the difficulty of improving the known upper bound [11].

The intention of this article is to summarize results and to develop a feeling for the proofs. The emphasis is on sketching several ideas at the expense of details. A comprehensive presentation is provided in the technical report [4].

2 Preliminaries

Basic Notations. Let $\mathbb{N}$ (resp., $\mathbb{Z}$) denote the set of natural numbers (resp., integers). $\mathbb{N}^+$ is the set of positive integers. For $x \in \mathbb{Z}$ the absolute value of x is denoted by $\mathrm{abs}(x)$, and for a matrix of integers $A = (a_{i,j}) \in \mathbb{Z}^{m\times n}$ for positive natural numbers m and n we define $||A||_\infty = \max\{\mathrm{abs}(a_{i,j}) \mid 1 \leq i \leq m \text{ and } 1 \leq j \leq n\}$.

L, NL, P, RP, BPP, NP, PSPACE, and NEXP denote standard complexity classes [23]. For a nondeterministic machine M, let $\mathrm{acc}_M(x)$ be the number of accepting paths of M on input x. The class #L consists of all functions acc_M, where M is a nondeterministic logarithmic-space-bounded machine. C$_=$L is the class of problems A for which there exist $f, g \in$ #L such that for all inputs x it holds that $x \in A \Leftrightarrow f(x) = g(x)$. Further information on counting classes can be found in [2].

Let Σ_i and Π_i denote the levels of the arithmetical hierarchy. 2EXPSPACE, i.e., the class of problems decidable by a deterministic algorithm in double exponential space $2^{2^{n^k}}$ for some $k \in \mathbb{N}$, and 3EXPSPACE, which consists of the problems decidable in triple exponential space. For complexity classes $\mathcal{C}$ let co$\mathcal{C} = \{\overline{A} \mid A \in \mathcal{C}\}$. We denote by K the Σ_1-complete halting problem (for some fixed Gödelization).

Addition and multiplication are extended to sets of integers: Let $A, B \subseteq \mathbb{Z}$. Then $A + B = \{a + b \mid a \in A, b \in B\}$ and $A \times B = \{a \cdot b \mid a \in A, b \in B\}$.

An oracle Turing machine is nonadaptive, if its queries are independent of the oracle (i.e., for all x and all oracles B and B', the computations $M^B(x)$ and $M^{B'}(x)$ have the same sequence of queries). For sets A and B we say that A is Turing reducible to B ($A \leq_{\mathrm{T}} B$), if there exists an oracle Turing machine M that accepts A with B as its oracle. If M is nonadaptive, then A is truth-table reducible to B ($A \leq_{\mathrm{tt}} B$). A is logspace Turing reducible to B ($A \leq_{\mathrm{T}}^{\log} B$), if there exists a logarithmic-space-bounded oracle Turing machine M (with one oracle tape) that accepts A with B as its oracle. If M's queries are nonadaptive (i.e., independent of the oracle), then A is logspace truth-table reducible to B ($A \leq_{\mathrm{tt}}^{\log} B$). A is logspace disjunctive-truth-table reducible to B ($A \leq_{\mathrm{dtt}}^{\log} B$), if there exists a logspace computable function f such that for all x, $f(x) = (y_1, y_2, \ldots, y_n)$ for some $n \geq 1$ and $\chi_A(x) = \max\{\chi_B(y_1), \chi_B(y_2), \ldots, \chi_B(y_n)\}$, where χ_S for a set S is the characteristic function of S. A set A is logspace (resp., polynomial time) many-one reducible to B, in notation $A \leq_{\mathrm{m}}^{\log} B$ (resp., $A \leq_{\mathrm{m}}^{\mathrm{p}} B$), if there exists a logarithmic-space-computable (resp., polynomial-time-computable) function f such that $\chi_A(x) = \chi_B(f(x))$. For a complexity class $\mathcal{C}$ we define $\mathcal{R}_{\mathrm{tt}}(\mathcal{C}) = \{A \mid \text{there is a } C \in \mathcal{C} \text{ with } A \leq_{\mathrm{tt}} C\}$.

Definition of circuits. A *circuit* $C = (V, E, g_C)$ is a finite, directed, acyclic graph with vertex set $V \subseteq \mathbb{N}$ and a designated vertex $g_C \in V$. Here, graphs are allowed to have multi-edges and are not required to be connected. We require that C is topologically ordered, that is, if $v, v' \in V$ are vertices with $v < v'$, then there is no edge from v' to v.

Let $\mathcal{O} \subseteq \{\cup, \cap, \overline{\ }, +, \times\}$. A *partially assigned $\mathcal{O}$-circuit* ($\mathcal{O}$-*circuit* for short) $C = (V, E, g_C, \alpha)$ is a circuit (V, E, g_C) whose nodes are labeled by the *labeling function* $\alpha : V \to \mathcal{O} \cup \mathbb{N} \cup \{\square\}$ such that each node has indegree ≤ 2, nodes with indegree 0 have labels from $\mathbb{N} \cup \{\square\}$, nodes with indegree 1 have label $\overline{\ }$, and nodes with indegree 2 have labels from $\mathcal{O} \setminus \{\overline{\ }\}$. In the context of circuits, nodes are also called *gates*. *Input gates* (i.e., gates with indegree 0) with labels from $\mathbb{N}$ are called *assigned* input gates. Input gates with label $\square$ are called *unassigned*. An $\mathcal{O}$-circuit whose input gates are all assigned is called *completely assigned $\mathcal{O}$-circuit*. We use the term *integer circuit* for both partially assigned $\mathcal{O}$-circuits and completely assigned $\mathcal{O}$-circuits.

There exists a deterministic logarithmic-space-bounded algorithm which on input of a graph decides whether the input is a partially assigned $\mathcal{O}$-circuit.

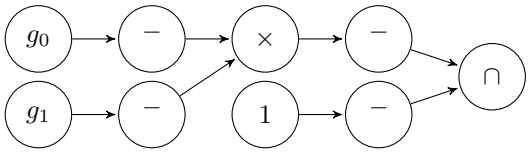
The set computed by a circuit. For an $\mathcal{O}$-circuit C with unassigned input gates $g_1 < \cdots < g_n$ and $x_1, \ldots, x_n \in \mathbb{N}$, let $C(x_1, \ldots, x_n)$ be the completely assigned $\mathcal{O}$-circuit that is obtained from C by modifying the labeling function α such that $\alpha(g_i) = x_i$ for $i = 1, \ldots, n$.

For a completely assigned $\mathcal{O}$-circuit $C = (V, E, g_C, \alpha)$ we inductively define the *set $I(g; C)$ computed by a gate $g \in V$* by

$$I(g;C) = \begin{cases} \{\alpha(g)\} \subseteq \mathbb{N} & \text{if } g \text{ has indegree } 0 \quad (g \text{ is an input gate}), \\ \mathbb{N} \setminus I(g';C) & \text{if } g = \overline{g'} \quad (g \text{ is a complement gate}), \\ I(g';C) \otimes I(g'';C) & \text{if } g = g' \otimes g'' \quad (g \text{ is a gate of type } \otimes \in \{\cup, \cap, +, \times\}). \end{cases}$$

The *set computed by C* is defined as $I(C) = I(g_C; C)$.

Example. For unassigned inputs g_0 and g_1, consider the circuit C:



We write $C = \overline{\overline{g_0} \times \overline{g_1}} \cap \overline{1}$ as an abbreviation. $C(1,1)$ computes the set of all primes, $C(x,y)$ for $x = y \in \mathbb{P} \cup \{0\}$ computes the set $\{x\}$, and $C(x,y)$ for all other (x,y) computes the empty set.

▶ **Definition 1.** Let $\mathcal{O} \subseteq \{\cup, \cap, \neg, +, \times\}$. We define *membership*, *emptiness*, *equivalence*, and *satisfiability* problems for circuits.

$$\begin{aligned}
\mathrm{MC}(\mathcal{O}) &\stackrel{df}{=} \{(C,b) \mid C \text{ is a completely assigned } \mathcal{O}\text{-circuit and } b \in I(C)\}\\
\Sigma_1\text{-MC}(\mathcal{O}) &\stackrel{df}{=} \{(C,b) \mid C \text{ is a partially assigned } \mathcal{O}\text{-circuit with } n \text{ unassigned inputs,}\\
&\qquad \text{there exist } x_1,\ldots,x_n \in \mathbb{N} \text{ s.t. } b \in I(C(x_1,\ldots,x_n))\}\\
\mathrm{EQ}(\mathcal{O}) &\stackrel{df}{=} \{(C_1,C_2) \mid C_1, C_2 \text{ are completely assigned } \mathcal{O}\text{-circuits, } I(C_1) = I(C_2)\}^1\\
\mathrm{EC}(\mathcal{O}) &\stackrel{df}{=} \{C \mid C \text{ is a completely assigned } \mathcal{O}\text{-circuit and } I(C) = \emptyset\}\\
\Sigma_1\text{-EC}(\mathcal{O}) &\stackrel{df}{=} \{C \mid C \text{ is a partially assigned } \mathcal{O}\text{-circuit with } n \text{ unassigned inputs,}\\
&\qquad \text{there exist } x_1,\ldots,x_n \in \mathbb{N} \text{ s.t. } I(C(x_1,\ldots,x_n)) = \emptyset\}\\
\Pi_1\text{-EC}(\mathcal{O}) &\stackrel{df}{=} \{C \mid C \text{ is a partially assigned } \mathcal{O}\text{-circuit with } n \text{ unassigned inputs,}\\
&\qquad \text{for all } x_1,\ldots,x_n \in \mathbb{N} \text{ it holds } I(C(x_1,\ldots,x_n)) = \emptyset\}\\
\Sigma_1\text{-NEC}(\mathcal{O}) &\stackrel{df}{=} \overline{\Pi_1\text{-EC}(\mathcal{O})}
\end{aligned}$$

The integer variants $\mathrm{MC}_{\mathbb{Z}}(\mathcal{O})$, $\mathrm{EC}_{\mathbb{Z}}(\mathcal{O})$, and $\Sigma_1\text{-EC}_{\mathbb{Z}}(\mathcal{O})$ are defined analogously (assigned and unassigned inputs are from $\mathbb{Z}$, the complement is defined with respect to $\mathbb{Z}$). A systematic study of the membership problems $\mathrm{MC}_{\mathbb{Z}}(\mathcal{O})$ was done by Travers [32].

We use the following abbreviations: we write n for the singleton $\{n\}$; we write C for $I(C)$, where C is a circuit; we write $\mathrm{MC}(\cup, \cap, \neg, +, \times)$ for $\mathrm{MC}(\{\cup, \cap, \neg, +, \times\})$ and the like.

3 Basic Results

We start with easy reductions between $\mathrm{EC}(\mathcal{O})$, $\Sigma_1\text{-EC}(\mathcal{O})$, $\Pi_1\text{-EC}(\mathcal{O})$, and $\mathrm{MC}(\mathcal{O})$.

▶ **Lemma 2.** *Let $\mathcal{O} \subseteq \{\cup, \cap, \neg, +, \times\}$ and $\mathcal{E} \in \{\mathrm{EC}, \Sigma_1\text{-EC}, \Pi_1\text{-EC}\}$.*

1. *If $\cap \in \mathcal{O}$, then $\mathrm{MC}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \overline{\mathrm{EC}(\mathcal{O})}$ and $\Sigma_1\text{-MC}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \Sigma_1\text{-NEC}(\mathcal{O})$.*
2. *If $\times \in \mathcal{O}$, then $\overline{\mathrm{EC}(\mathcal{O})} \leq_{\mathrm{m}}^{\log} \mathrm{MC}(\mathcal{O})$ and $\Sigma_1\text{-NEC}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \Sigma_1\text{-MC}(\mathcal{O})$.*
3. *If $\mathcal{O} \subseteq \{\cup, +, \times\}$ or $\mathcal{O} \subseteq \{\neg\}$, then $\mathrm{EC}(\mathcal{O}) = \Sigma_1\text{-EC}(\mathcal{O}) = \Pi_1\text{-EC}(\mathcal{O}) = \emptyset$.*
4. *$\mathcal{E}(\{\cup, \neg\} \cup \mathcal{O}) \equiv_{\mathrm{m}}^{\log} \mathcal{E}(\{\cap, \neg\} \cup \mathcal{O}) \equiv_{\mathrm{m}}^{\log} \mathcal{E}(\{\cup, \cap, \neg\} \cup \mathcal{O})$ for $\mathcal{O} \subseteq \{+, \times\}$.*
5. *$\mathcal{E}(\mathcal{O}') \leq_{\mathrm{m}}^{\log} \mathcal{E}(\mathcal{O})$ for $\mathcal{O}' \subseteq \mathcal{O}$.*
6. *$\mathrm{EC}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \Sigma_1\text{-EC}(\mathcal{O})$ and $\mathrm{EC}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \Pi_1\text{-EC}(\mathcal{O})$.*

The following bounds are obtained with minor effort from known results and Lemma 2.

▶ **Theorem 3** ([22, 11, 13])**.**

1. *$\mathrm{EC}(\cup, \cap, +, \times)$ is $\leq_{\mathrm{m}}^{\log}$-complete for* coNEXP*.*
2. *$\mathrm{EC}(\cup, \cap, \neg, +), \mathrm{EC}(\cup, \cap, \neg, \times) \in$ PSPACE.*

[1] In [11], equivalence problems for circuits are denoted by $\mathrm{EC}(\mathcal{O})$, which is in conflict with our notation for emptiness problems. Therefore, we use the notation $\mathrm{EQ}(\mathcal{O})$ for equivalence problems.

3. $\mathrm{EC}(\cap,+)$ *and* $\mathrm{EC}(\cap,\times)$ *are* $\leq_{\mathrm{m}}^{\log}$*-hard for* $\mathrm{coC_{=}L}$.
4. $\Pi_1\text{-EC}(\cap,\times)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* coNP.

Circuits with solely set operations can express graph accessibility as well as evaluation and satisfiability of Boolean circuits. This leads to the following results.

▶ **Proposition 4.**
1. $\mathrm{EC}(\cap)$, $\Sigma_1\text{-EC}(\cap)$, *and* $\Pi_1\text{-EC}(\cap)$ *are* $\leq_{\mathrm{m}}^{\log}$*-complete for* NL.
2. $\mathrm{EC}(\cup,\cap,^{-})$, $\mathrm{EC}(\cup,\cap)$, $\Sigma_1\text{-EC}(\cup,\cap)$, *and* $\Pi_1\text{-EC}(\cup,\cap)$ *are* $\leq_{\mathrm{m}}^{\log}$*-complete for* P.
3. $\Sigma_1\text{-EC}(\cup,\cap,^{-})$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* NP *and* $\Pi_1\text{-EC}(\cup,\cap,^{-})$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* coNP.

4 Circuits with One Arithmetic Operation

4.1 Circuits without Complement

Here only those problems are relevant that admit intersection, since otherwise the circuits compute non-empty sets.

By an induction on the structure of the circuit we obtain: $C \in \Sigma_1\text{-EC}(\cap,+)$ if and only if at least one of the circuits $C(0,\ldots,0)$, $C(1,0,\ldots,0)$, $C(0,1,\ldots,0)$, $\ldots$, $C(0,0,\ldots,1)$ belongs to $\mathrm{EC}(\cap,+)$. Hence $\Sigma_1\text{-EC}(\cap,+) \leq_{\mathrm{dtt}}^{\log} \mathrm{EC}(\cap,+)$. It is known that $\mathrm{EC}(\cap,+) \in \mathrm{coC_{=}L}$ [11] and $\mathrm{coC_{=}L}$ is closed under $\leq_{\mathrm{dtt}}^{\log}$ [3]. This yields:

▶ **Theorem 5.** $\Sigma_1\text{-EC}(\cap,+) \in \mathrm{coC_{=}L}$.

We obtain several results that rely on an estimation by Schrijver [28] saying that systems of linear equations consisting of arbitrarily many equations have solutions whose greatest component is at most $(32k)^{12n^4}$, where k is the greatest constant in the system and n the number of variables. So there are "small solutions for huge systems of small equations".

It is straightforward to see that the question of whether an $\{\cap,+\}$-circuit C is in $\mathcal{E}(\cap,+)$ for $\mathcal{E} \in \{\mathrm{EC}, \Sigma_1\text{-EC}, \Pi_1\text{-EC}\}$ can be answered by solving a system of linear equations in which each unassigned input gate is represented by one variable and constants have polynomial length in the size of the circuit. We extend this idea such that also emptiness problems that allow unions can be reduced to similar problems regarding (sets of) equation systems. This leads to the following results.

▶ **Theorem 6.**
1. $\Pi_1\text{-EC}(\cap,+)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* coNP.
2. $\mathrm{EC}(\cup,\cap,+)$ *and* $\mathrm{EC}(\cup,\cap,\times)$ *are* $\leq_{\mathrm{m}}^{\log}$*-hard for* PSPACE.
3. $\Sigma_1\text{-EC}(\cup,\cap,+), \Pi_1\text{-EC}(\cup,\cap,+), \Pi_1\text{-EC}(\cup,\cap,\times) \in \mathrm{PSPACE}$.

The problems Σ_1-EC and EC for the sets of operations $\{\cap,\times\}$ and $\{\cup,\cap,\times\}$ will be solved by a general tool given in Theorems 19, 20, and Corollary 21.

4.2 Circuits with Complement

4.2.1 Upper Bounds

$\mathrm{Th}(\mathbb{N};+,=)$ denotes the Presburger arithmetic, i.e., the first-order theory of $\mathbb{N}$ with addition. $\mathrm{Th}(\mathbb{N};\times,=,\mathbb{P}\cup\{0,1\})$ denotes the Skolem arithmetic with constants, i.e., the first-order theory of $\mathbb{N}$ with multiplication and constants for 0, 1, and all primes.

▶ **Theorem 7** ([9, 10, 14, 5])**.**
1. $\mathrm{Th}(\mathbb{N};+,=) \in \mathrm{2EXPSPACE}$.
2. $\mathrm{Th}(\mathbb{N};\times,=,\mathbb{P}\cup\{0,1\}) \in \mathrm{3EXPSPACE}$.

The sets computed in the nodes of circuits over $\{\cup, \cap, \overline{\ }, +\}$ and $\{\cup, \cap, \overline{\ }, \times\}$ can be expressed by Presburger and Skolem formulas. These formulas can be constructed in polynomial time, which allows to transfer the upper bounds of Presburger and Skolem arithmetic.

▶ **Theorem 8.**
1. Σ_1-EC$(\cup, \cap, \overline{\ }, +), \Pi_1$-EC$(\cup, \cap, \overline{\ }, +) \in$ 2EXPSPACE.
2. Σ_1-EC$(\cup, \cap, \overline{\ }, \times), \Pi_1$-EC$(\cup, \cap, \overline{\ }, \times) \in$ 3EXPSPACE.

4.2.2 Lower Bounds

All emptiness problems covered by Section 4.2 – in particular EC$(\overline{\ }, +)$ and EC$(\overline{\ }, \times)$ – are $\leq_{\mathrm{m}}^{\log}$-hard for PSPACE. We show the same for MC$(\overline{\ }, +)$ and MC$(\overline{\ }, \times)$ improving unpublished results by Reinhardt, which were announced by McKenzie and Wagner [22] and which state that these problems are $\leq_{\mathrm{m}}^{\mathrm{p}}$-hard for PSPACE. This section mainly sketches our proof for the $\leq_{\mathrm{m}}^{\log}$-hardness of MC$(\overline{\ }, +)$ for PSPACE. For that we define a further problem.

▶ **Definition 9.** A $\{\overline{\ }, +\}$-circuit over $\mathbb{N}^k$ is a completely assigned $\{\overline{\ }, +\}$-circuit $C = ((V, E), g_C, \alpha)$ where all constants are elements of $\mathbb{N}^k$. The set $I(g; C) \subseteq \mathbb{N}^k$ computed by a node g is defined analogously to the one-dimensional case. Further let $I(C) = I(g_C; C)$ and

$$\mathrm{MC}^+(\overline{\ }, +) = \{(C, b) \mid C \text{ is a completely assigned } \{\overline{\ }, +\}\text{-circuit over } \mathbb{N}^k,$$
$$||c||_\infty \leq 1 \text{ for every input } c, \ ||b||_\infty \leq 3, \text{ and } b \in I(C)\}.$$

The PSPACE-hardness of this problem is obtained by reducing CNF-QBF, which is the problem of whether a given quantified Boolean formula F in conjunctive normal form is true. It is known that CNF-QBF is $\leq_{\mathrm{m}}^{\log}$-complete for PSPACE.

The proof of the following lemma is based on an unpublished proof by Reinhardt [25] showing the $\leq_{\mathrm{m}}^{\mathrm{p}}$-hardness of MC$(\overline{\ }, \times)$ for PSPACE.

▶ **Lemma 10.** MC$^+(\overline{\ }, +)$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* PSPACE.

From now on our proof differs from Reinhardt's proof. Instead of showing directly $\mathrm{MC}^+(\overline{\ }, +) \leq_{\mathrm{m}}^{\mathrm{p}} \mathrm{MC}(\overline{\ }, \times)$ via a simple reduction, we first prove $\mathrm{MC}^+(\overline{\ }, +) \leq_{\mathrm{m}}^{\log} \mathrm{MC}(\overline{\ }, +)$ and then show $\overline{\mathrm{MC}(\overline{\ }, +)} \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\overline{\ }, +) \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\overline{\ }, \times) \leq_{\mathrm{m}}^{\log} \overline{\mathrm{MC}(\overline{\ }, \times)}$.

To show $\mathrm{MC}^+(\overline{\ }, +) \leq_{\mathrm{m}}^{\log} \mathrm{MC}(\overline{\ }, +)$ it is convenient to represent a vector of natural numbers $a = (a_0, \ldots, a_k)$ as a natural number $\mathrm{ad}_n(a) = \sum_{i=0}^{k} a_{k-i} n^i$ for $n \geq 2$. We denote the function mapping (C, b) onto $(C', \mathrm{ad}_n(b))$ by ad_{n}, where C' is the circuit obtained from C by replacing each input x with $\mathrm{ad}_n(x)$.

Indeed, ad_n for sufficiently large n works as reduction if for example union and intersection are allowed instead of complement. However, in our case we have $(0, 1, 0) \notin \overline{(0, 0, 0)} + (0, 0, 1)$, but $\mathrm{ad}_8(0, 1, 0) = \mathrm{ad}_8(0, 0, 7) + \mathrm{ad}_8(0, 0, 1) \in \overline{\mathrm{ad}_8(0, 0, 0)} + \mathrm{ad}_8(0, 0, 1)$. Such "overflows" are the reason why ad_n is not a reduction $\mathrm{MC}^+(\overline{\ }, +) \leq_{\mathrm{m}}^{\log} \mathrm{MC}(\overline{\ }, +)$ for any n.

To address this problem we implement an operation similar to the bitwise 'or' for characteristic sequences: for two finite sets A and B (note that for problems of the form "$(C, b) \in \mathrm{MC}^+(\overline{\ }, +)$?" it suffices to consider the first $b + 1$ bits of characteristic sequences of sets) let $m = \max(A \cup B)$ and consider $M = \overline{\overline{A + \{m-1\}} + 1 + \overline{B + \{m-1\}}} + 1 = ((A + \{m\}) \cup \{0\}) + ((B + \{m\}) \cup \{0\})$. Observe that $M \cap \{m, \ldots, 2m\} = \{m + x \mid x \in A \cup B\}$, which equals the union of A and B shifted by the offset m.

Now a circuit over $\mathbb{N}^k$ can be simulated by a circuit over $\mathbb{N}$: Roughly speaking, we use ad_8 and after computing the operation of some node g, the numbers x with $||\mathrm{ad}_8^{-1}(x)|| > 3$ can be deleted from $I(g)$ by adding them into $\overline{I(g)}$ via the 'or'-operation mentioned above. Every 'or'-operation yields an offset which has to be taken into account. This leads to:

▶ **Theorem 11.** $\mathrm{MC}(^{-},+)$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* PSPACE.

The following theorem is essentially due to Sigmund [30]. He provided the proof of the second reduction and the main idea of the proof of the first one.

▶ **Theorem 12.** $\overline{\mathrm{MC}(^{-},+)} \leq_{\mathrm{m}}^{\log} \mathrm{EC}(^{-},+)$ *and* $\mathrm{EC}(^{-},+) \leq_{\mathrm{m}}^{\log} \mathrm{EC}(^{-},\times)$.

▶ **Corollary 13.** $\mathrm{EC}(^{-},+)$, $\mathrm{EC}(^{-},\times)$, *and* $\mathrm{MC}(^{-},\times)$ *are* $\leq_{\mathrm{m}}^{\log}$*-hard for* PSPACE.

5 Circuits with both Arithmetic Operations

Besides proving bounds for emptiness problems with $+$ and $\times$, we improve the known lower and upper bounds for $\mathrm{MC}(\cup,\cap,^{-},+,\times)$ [22] and $\mathrm{EQ}(\cup,\cap,^{-},+,\times)$ [11]. Then we provide arguments that suggest the difficulty of proving the decidability of $\mathrm{EC}(^{-},+,\times)$ and $\mathrm{EC}(\cup,\cap,^{-},+,\times,)$. Finally we draw connections to polynomial identity testing (PIT) and show that the open questions for the complexities of $\mathrm{MC}(\cap,+,\times)$ [22], $\mathrm{MC}_{\mathbb{Z}}(\cap,+,\times)$, $\mathrm{MC}_{\mathbb{Z}}(+,\times)$ [32], and $\mathrm{EQ}(+,\times)$ [11] are equivalent to the well-studied, open question for the complexity of PIT.

5.1 Upper and Lower Bounds for Possibly Undecidable Problems

We obtain upper bounds by improving a known decision algorithm [11] and lower bounds via the Matiyasevich-Robinson-Davis-Putnam theorem [21, 7].

▶ **Theorem 14.**
1. $\mathrm{EC}(\cup,\cap,^{-},+,\times) \in \mathcal{R}_{\mathrm{tt}}(\Sigma_1)$.
2. $\Sigma_1\text{-}\mathrm{EC}(\cup,\cap,^{-},+,\times) \in \Sigma_2$ *and* $\Pi_1\text{-}\mathrm{EC}(\cup,\cap,^{-},+,\times) \in \Pi_2$.
3. $\Pi_1\text{-}\mathrm{EC}(\cap,+,\times)$ *and* $\Pi_1\text{-}\mathrm{EC}(\cup,\cap,+,\times)$ *are* $\leq_{\mathrm{m}}^{\log}$*-complete for* Π_1.
4. $\Sigma_1\text{-}\mathrm{MC}(^{-},+,\times)$ *and* $\Sigma_1\text{-}\mathrm{EC}(^{-},+,\times)$ *are* $\leq_{\mathrm{m}}^{\log}$*-hard for* Σ_1.
5. $\Pi_1\text{-}\mathrm{EC}(^{-},+,\times)$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* Π_1.

5.2 Connecting Emptiness with Membership and Equivalence Problems

We show that with the operations $^{-}$, $+$, and $\times$ one can express a Boolean combination of emptiness problems as a single emptiness problem. Therefore, truth-table reductions to certain emptiness problems can be transformed into many-one reductions. This allows us to show certain emptiness problems to be many-one equivalent to membership problems and equivalence problems. As a byproduct we improve the known lower and upper bounds of $\mathrm{MC}(\cup,\cap,^{-},+,\times)$ [22] and $\mathrm{EQ}(\cup,\cap,^{-},+,\times)$ [11].

▶ **Proposition 15.** *The following holds if* $\{^{-},+,\times\} \subseteq \mathcal{O}$.
1. *If* $A \leq_{\mathrm{tt}}^{\log} \mathrm{EC}(\mathcal{O})$, *then* $A \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\mathcal{O})$. *If* $A \leq_{\mathrm{tt}} \mathrm{EC}(\mathcal{O})$, *then* $A \leq_{\mathrm{m}} \mathrm{EC}(\mathcal{O})$.
2. *If* $\mathrm{EC}(\mathcal{O})$ *is* $\leq_{\mathrm{m}}$*-hard for* Σ_1, *then it is* $\leq_{\mathrm{m}}$*-hard for* $\mathcal{R}_{\mathrm{tt}}(\Sigma_1)$.
3. *If* $\mathrm{EC}(\mathcal{O}) \in \Sigma_1 \cup \Pi_1$, *then* $\mathrm{EC}(\mathcal{O}) \in \Sigma_0$.

▶ **Corollary 16.**
1. $\mathrm{MC}(\cup,\cap,^{-},+,\times) \equiv_{\mathrm{m}}^{\log} \mathrm{EQ}(\cup,\cap,^{-},+,\times) \equiv_{\mathrm{m}}^{\log} \mathrm{EC}(\cup,\cap,^{-},+,\times) \equiv_{\mathrm{m}}^{\log} \overline{\mathrm{EC}(\cup,\cap,^{-},+,\times)}$.
2. $\Sigma_1\text{-}\mathrm{MC}(\cup,\cap,^{-},+,\times) \equiv_{\mathrm{m}}^{\log} \Sigma_1\text{-}\mathrm{EC}(\cup,\cap,^{-},+,\times) \equiv_{\mathrm{m}}^{\log} \Sigma_1\text{-}\mathrm{NEC}(\cup,\cap,^{-},+,\times)$.
3. $\mathrm{EC}(\cup,\cap,^{-},+,\times), \mathrm{MC}(\cup,\cap,^{-},+,\times), \mathrm{EQ}(\cup,\cap,^{-},+,\times) \in \mathcal{R}_{\mathrm{tt}}(\Sigma_1)$ *and these problems are* $\leq_{\mathrm{m}}^{\log}$*-hard for* $\mathcal{R}_{\mathrm{T}}^{\log}(\mathrm{NEXP}) = \mathrm{L}^{\mathrm{NEXP}}$.
4. $\mathrm{EC}(^{-},+,\times)$ *is* $\leq_{\mathrm{m}}$*-hard for* Σ_1 *if and only if it is* $\leq_{\mathrm{m}}$*-complete for* $\mathcal{R}_{\mathrm{tt}}(\Sigma_1)$.
5. $\mathrm{EC}(\cup,\cap,^{-},+,\times)$ *is* $\leq_{\mathrm{m}}$*-hard for* Σ_1 *if and only if it is* $\leq_{\mathrm{m}}$*-complete for* $\mathcal{R}_{\mathrm{tt}}(\Sigma_1)$.

For $\{^{-}, +, \times\}$-circuits there are further equivalences between membership and emptiness problems.

▶ **Proposition 17.**
1. $\mathrm{MC}(^{-}, +, \times) \equiv_{\mathrm{m}}^{\log} \mathrm{EC}(^{-}, +, \times) \equiv_{\mathrm{m}}^{\log} \overline{\mathrm{EC}(^{-}, +, \times)}$.
2. $\Sigma_1\text{-MC}(^{-}, +, \times) \equiv_{\mathrm{m}}^{\log} \Sigma_1\text{-EC}(^{-}, +, \times) \equiv_{\mathrm{m}}^{\log} \Sigma_1\text{-NEC}(^{-}, +, \times)$.

5.3 The Difficulty of $\mathrm{EC}(^{-}, +, \times)$ and $\mathrm{EC}(\cup, \cap, ^{-}, +, \times)$

In the Corollaries 13 and 16 we showed that $\mathrm{EC}(^{-}, +, \times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for PSPACE and $\mathrm{EC}(\cup, \cap, ^{-}, +, \times,)$ is $\leq_{\mathrm{m}}^{\log}$-hard for $\mathrm{L}^{\mathrm{NEXP}}$. By Theorem 14, both problems belong to $\mathcal{R}_{\mathrm{tt}}(\Sigma_1)$. It is an open question whether these problems are decidable. This subsection explains the difficulty of finding decision algorithms for these problems.

Goldbach conjectured that every even integer greater than 2 is the sum of two primes. At the time the conjecture was made 1 was considered to be prime, but later the opposite view became accepted. Let $\mathbb{P}_1 = \mathbb{P} \cup \{1\}$. Below we formulate both variants, Goldbach's original conjecture (GC_1) and the one that nowadays is called *Goldbach's conjecture* (GC).

$$\begin{aligned} \mathrm{GC}_1 &= \forall n \geq 1\ \exists p, q \in \mathbb{P}_1\ \ [2n = p + q] \\ \mathrm{GC} &= \forall n \geq 2\ \exists p, q \in \mathbb{P}\ \ [2n = p + q] \end{aligned}$$

We define circuits that express the truth of these conjectures, where $\mathbb{P}_1$ stands for $\overline{\overline{1} \times \overline{1}}$, $\mathbb{P}$ for $\overline{\overline{1} \times \overline{1}} \cap \overline{1}$, and $\{0, 1\}$ for $\overline{\overline{0} + 1}$.

$$\begin{aligned} C_1 &= \overline{((\mathbb{P}_1 + \mathbb{P}_1) \times \{0, 1\}) + \{0, 1\}} \\ C &= \overline{\mathbb{P} + \mathbb{P}} \cap (2 \times \overline{\{0, 1\}}) \end{aligned}$$

GC_1 is true if any only if $C_1 \in \mathrm{EC}(^{-}, +, \times)$. GC is true if and only if $C \in \mathrm{EC}(\cup, \cap, ^{-}, +, \times,)$. This tells us: If one finds a decision algorithm for $\mathrm{EC}(^{-}, +, \times)$ or $\mathrm{EC}(\cup, \cap, ^{-}, +, \times)$ and proves its correctness, then in a sense this solves Goldbach's conjecture, since the computation of this algorithm is a proof or refutation. In particular, this would imply that Goldbach's conjecture is provable or refutable, which is unknown (cf. [17]). This underlines the difficulty of finding decision algorithms for $\mathrm{EC}(^{-}, +, \times)$ and $\mathrm{EC}(\cup, \cap, ^{-}, +, \times)$.

5.4 Connection between Emptiness and Σ_1-Emptiness

We show that several emptiness problems are equivalent to their Σ_1-emptiness variants. The proof exploits the fact that the test of whether a multivariate polynomial with coefficients bounded by some constant K is identically zero is possible by evaluating this polynomial for one fixed argument only dependent on K and the total degree of the polynomial. The following lemma shows that the test of whether multivariate polynomials are identically zero reduces to the univariate case.

▶ **Lemma 18** ([20]). *Given a polynomial $f(x_1, \ldots, x_n)$ over $\mathbb{R}$ with total degree at most d, the substitution $x_i \mapsto x^{(d+1)^{i-1}}$ has the property that f is identically zero on $\mathbb{R}^n$ if and only if the obtained univariate polynomial is identically zero on $\mathbb{R}$.*

The lemma allows a reduction from $\Sigma_1\text{-EC}(\cap, +, \times)$ to $\mathrm{EC}(\cap, +, \times)$: Consider a circuit $C \in \Sigma_1\text{-EC}(\cap, +, \times)$ with unassigned inputs $u_1, \ldots, u_n$ and let $z_1, \ldots, z_n \in \mathbb{N}$ such that $C(z_1, \ldots, z_n) = \emptyset$. So under this assignment there exists a $\cap$-gate g connected to the output and computing $\emptyset$ such that no ancestor of g computes $\emptyset$. The unique number computed in the left\right predecessor $g_l \backslash g_r$ of g (note that due to the absence of $^{-}$ and $\cup$ each gate

computes a set containing at most one element) can be written as a multivariate polynomial $p_l \backslash p_r$ with variables $u_1, \ldots, u_n$. It holds that $p_l \neq p_r$, since g computes $\emptyset$. By Lemma 18, the same holds for the univariate polynomials $p'_l \backslash p'_r$ obtained by the substitution. Note that $p'_l(x) \neq p'_r(x)$ for every large enough x. Moreover, there is a circuit computable in logarithmic space that generates such an x. So the substitution rule provides the assignment $x^{(d+1)^0}, x^{(d+1)^1}, \ldots, x^{(d+1)^{n-1}}$ under which g and hence also C computes $\emptyset$. This yields the following theorem.

▶ **Theorem 19.**
1. $\mathrm{EC}(\cap,+,\times) \equiv_{\mathrm{m}}^{\log} \Sigma_1\text{-}\mathrm{EC}(\cap,+,\times)$.
2. $\mathrm{EC}_{\mathbb{Z}}(\cap,+,\times) \equiv_{\mathrm{m}}^{\log} \Sigma_1\text{-}\mathrm{EC}_{\mathbb{Z}}(\cap,+,\times)$.
3. $\mathrm{EC}(\cap,\times) \equiv_{\mathrm{m}}^{\log} \Sigma_1\text{-}\mathrm{EC}(\cap,\times)$.

We generalize this argument to $\{\cup,\cap,+,\times\}$-circuits by unfolding such circuits C to a tree D, which exponentially increases the size, but not the depth. Then we observe that $C(z_1,\ldots,z_n) = \emptyset$ if and only if for all possibilities to prune D to some D' such that each $\cup$-gate has exactly one predecessor it holds that $D'(z_1,\ldots,z_n) = \emptyset$. Since a $\cup$-gate with exactly one predecessor acts like a wire, the D' are $\{\cap,+,\times\}$-circuits. Hence $C \in \Sigma_1\text{-}\mathrm{EC}(\cup,\cap,+,\times)$ if and only if for all D' it holds that $D' \in \Sigma_1\text{-}\mathrm{EC}(\cap,+,\times)$. So we reached a situation similar to Theorem 19.1 with the difference that D' has exponential size and polynomial depth, which translates to polynomials with an exponential number of monomials and polynomially bounded degrees. Since the argument for Theorem 19 depends only on the polynomial's degree, but not on the number of monomials we obtain:

▶ **Theorem 20.**
1. $\mathrm{EC}(\cup,\cap,+,\times) \equiv_{\mathrm{m}}^{\log} \Sigma_1\text{-}\mathrm{EC}(\cup,\cap,+,\times)$.
2. $\mathrm{EC}(\cup,\cap,\times) \equiv_{\mathrm{m}}^{\log} \Sigma_1\text{-}\mathrm{EC}(\cup,\cap,\times)$.

From known results on $\mathrm{MC}(\cap,+,\times)$ and $\mathrm{MC}(\cap,\times)$ [22] and Theorem 6 we obtain:

▶ **Corollary 21.**
1. $\mathrm{EC}(\cap,+,\times) \equiv_{\mathrm{m}}^{\log} \Sigma_1\text{-}\mathrm{EC}(\cap,+,\times) \equiv_{\mathrm{m}}^{\log} \overline{\mathrm{MC}(\cap,+,\times)} \equiv_{\mathrm{m}}^{\log} \overline{\mathrm{EQ}(+,\times)}$.
2. $\Sigma_1\text{-}\mathrm{EC}(\cap,+,\times) \in \mathrm{RP}$.
3. $\Sigma_1\text{-}\mathrm{EC}(\cup,\cap,+,\times)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* coNEXP.
4. $\Sigma_1\text{-}\mathrm{EC}(\cap,\times) \in \mathrm{P}$.
5. $\mathrm{EC}(\cap,\times) \equiv_{\mathrm{m}}^{\log} \Sigma_1\text{-}\mathrm{EC}(\cap,\times) \equiv_{\mathrm{m}}^{\log} \overline{\mathrm{MC}(\cap,\times)}$.
6. $\Sigma_1\text{-}\mathrm{EC}(\cup,\cap,\times)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* PSPACE.

The 5th statement shows that improving the non-matching bounds for $\mathrm{EC}(\cap,\times)$ is as difficult as improving the bounds for $\mathrm{MC}(\cap,\times)$, which is an open problem from [22].

5.5 Connection to Polynomial Identity Testing

We extend the equivalence in statement 1 of Corollary 21 by $\mathrm{EC}_{\mathbb{Z}}(\cap,+,\times)$, $\Sigma_1\text{-}\mathrm{EC}_{\mathbb{Z}}(\cap,+,\times)$, $\overline{\mathrm{MC}_{\mathbb{Z}}(\cap,+,\times)}$, $\overline{\mathrm{MC}_{\mathbb{Z}}(+,\times)}$, and $\overline{\mathrm{PIT}}$. The connection to PIT is interesting as it explains the difficulty of several open questions, namely the non-matching lower and upper bounds of $\mathrm{MC}(\cap,+,\times)$ in [22], $\mathrm{MC}_{\mathbb{Z}}(\cap,+,\times)$ and $\mathrm{MC}_{\mathbb{Z}}(+,\times)$ in [32], and $\mathrm{EQ}(+,\times)$ in [11]. In addition, it settles the question for the complexity of $\mathrm{EC}(\cap,+,\times)$ and $\Sigma_1\text{-}\mathrm{EC}(\cap,+,\times)$.

PIT (polynomial identity testing) is the following problem: For a given integer circuit consisting of input gates associated with variables/constants from $\mathbb{Z}$ and internal gates for addition/multiplication over $\mathbb{Z}$, one has to decide whether the polynomial described by the

circuit is identically zero or not. The term *identically zero* means that the polynomial must be formally zero, i.e., if we write it as a linear combination of monomials with coefficients from $\mathbb{Z}$, then all coefficients are zero. For the ring $\mathbb{Z}$ this is equivalent to requiring that the polynomial is zero on $\mathbb{Z}^n$, where n is the number of unassigned input gates. (For other rings this makes a difference: for example over $\mathbb{F}_2$, the polynomial $x^2 + x$ is not identically zero, although it is zero on $\mathbb{F}_2$.) Formally, we can define PIT as the following problem concerning $\{+, \times\}$-circuits over $\mathbb{Z}$:

$$\text{PIT} \stackrel{df}{=} \{\, C \mid C \text{ is a } \{+,\times\}\text{-circuit with unassigned inputs } u_1 < \cdots < u_n \text{ such that the assigned inputs have labels from } \mathbb{Z} \text{ and for all } x_1,\ldots,x_n \in \mathbb{Z} \text{ it holds that } I(C(x_1,\ldots,x_n)) = \{0\} \,\}.$$

It is known that PIT $\in$ coRP [15], but proving the exact complexity of PIT is considered as one of the greatest challenges in algebraic computing complexity [26] and theoretical computer science in general [29]. This fundamental problem has many applications, e.g., a deterministic primality test [1]. For further background on PIT we refer to the articles [26, 29, 27, 19]. Let $\mathcal{PIT}$ denote the class of problems that are $\leq_{\mathrm{m}}^{\log}$-reducible to PIT.

▶ **Theorem 22.** $\mathrm{EC}(\cap,+,\times) \equiv_{\mathrm{m}}^{\log} \overline{\mathrm{MC}(\cap,+,\times)} \equiv_{\mathrm{m}}^{\log} \overline{\mathrm{EQ}(+,\times)} \equiv_{\mathrm{m}}^{\log} \overline{\mathrm{PIT}} \equiv_{\mathrm{m}}^{\log} \mathrm{EC}_{\mathbb{Z}}(\cap,+,\times)$.

We sketch the proof: By Corollary 21, $\mathrm{EC}(\cap,+,\times) \equiv_{\mathrm{m}}^{\log} \overline{\mathrm{MC}(\cap,+,\times)} \equiv_{\mathrm{m}}^{\log} \overline{\mathrm{EQ}(+,\times)}$. Theorem 19 implies $\overline{\mathrm{EQ}(+,\times)} \leq_{\mathrm{m}}^{\log} \overline{\mathrm{PIT}} \leq_{\mathrm{m}}^{\log} \Sigma_1\text{-}\mathrm{EC}_{\mathbb{Z}}(\cap,+,\times) \leq_{\mathrm{m}}^{\log} \mathrm{EC}_{\mathbb{Z}}(\cap,+,\times)$. It remains to show $\mathrm{EC}_{\mathbb{Z}}(\cap,+,\times) \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\cap,+,\times)$.

We simulate a $\{\cap,+,\times\}$-circuit C over $\mathbb{Z}$ by a $\{\cap,+,\times\}$-circuit C' over $\mathbb{N}$ such that the value $v \in \mathbb{Z}$ computed in gate i of C is represented in C' by two positive numbers $\widetilde{i} + v$ and $\widetilde{i} - v$, where $\widetilde{i} = 2^{3^i}$. This shift by $\widetilde{i}$ allows $\{\cap,+,\times\}$-circuits over $\mathbb{N}$ to represent numbers from $\mathbb{Z}$. A technical elaboration shows that the circuits can also *process* numbers represented in this way, i.e., there are subcircuits that simulate the operations $\cap$, $+$, and $\times$.

Together with the Theorems 19 and 22 we obtain:

▶ **Corollary 23.** *The following problems are* $\leq_{\mathrm{m}}^{\log}$*-equivalent to* $\overline{\mathrm{PIT}}$*:*
$\mathrm{EC}(\cap,+,\times)$, $\Sigma_1\text{-}\mathrm{EC}(\cap,+,\times)$, $\mathrm{EC}_{\mathbb{Z}}(\cap,+,\times)$, $\Sigma_1\text{-}\mathrm{EC}_{\mathbb{Z}}(\cap,+,\times)$, $\overline{\mathrm{MC}_{\mathbb{Z}}(\cap,+,\times)}$, $\overline{\mathrm{MC}_{\mathbb{Z}}(+,\times)}$.

The equivalence to $\overline{\mathrm{PIT}}$ shows the difficulty of understanding the complexity of the problems $\mathrm{EC}(\cap,+,\times)$ and $\Sigma_1\text{-}\mathrm{EC}(\cap,+,\times)$ as well as the open problems from [22, 32, 11] mentioned above. Kabanets and Impagliazzo [16] substantiate the hardness of obtaining subexponential algorithms for PIT by showing that it implies that NEXP $\not\subset$ P/poly or the permanent is not computable by polynomial size arithmetic circuits over $\mathbb{Q}$ with divisions. Both statements are expected to be difficult to prove.

In view of Theorem 22 it seems unlikely that $\mathrm{EQ}(\cap,+,\times)$ is equivalent to PIT: A straightforward proof shows that $\mathrm{EQ}(\cap,+,\times)$ is $\leq_{\mathrm{m}}^{\log}$-complete for $\mathcal{PIT} \vee \mathrm{co}\mathcal{PIT} = \{L \cup L' \mid L \in \mathcal{PIT}, L' \in \mathrm{co}\mathcal{PIT}\}$, which is the complement of the second level of the Boolean hierarchy [18] over $\mathcal{PIT}$. If $\mathrm{EQ}(\cap,+,\times) \equiv_{\mathrm{m}}^{\log} \mathrm{PIT}$, then $\mathcal{PIT} = \mathcal{PIT} \vee \mathrm{co}\mathcal{PIT}$ and hence $\mathrm{PIT} \equiv_{\mathrm{m}}^{\log} \overline{\mathrm{PIT}} \in \mathrm{RP} \subseteq \mathrm{NP}$. Kabanets and Impagliazzo [16] show that PIT $\in$ NP is unlikely, since it implies NEXP$\cap$coNEXP $\not\subset$ P/poly or the permanent is not computable by polynomial-size arithmetic circuits over $\mathbb{Q}$ with divisions. This also explains the difficulty of improving the upper bound for $\mathrm{EQ}(\cap,+,\times)$ from BPP [11] to coRP, since this implies $\overline{\mathrm{PIT}} \leq_{\mathrm{m}}^{\log} \mathrm{EQ}(\cap,+,\times) \in \mathrm{coRP}$.

6 Conclusions and Open Questions

The results of this paper are summarized in Table 1. For most of the emptiness problems it was possible to precisely characterize their complexity.

Table 1 Upper bounds mean membership in the class, lower bounds stand for $\leq_{\mathrm{m}}^{\log}$-hardness for the class. Numbers refer to results in this paper. Gray cells do not contain references, since by statement 5 of Lemma 2 these results are obtained from white cells. Subsets $\mathcal{O}$ that are missing in the first column either correspond to trivial problems (statement 3 of Lemma 2) or can be transformed by De Morgan's law to an equivalent subset (statement 4 of Lemma 2). $\mathcal{PIT}$ is the class of problems that are logspace many-one reducible to polynomial identity testing, which is a well-studied problem in algebraic computing complexity. It is known that $\mathrm{P} \subseteq \mathcal{PIT} \subseteq \mathrm{coRP}$ and it is an open problem to improve these bounds.

$\mathcal{O}$	EC l.b.	EC u.b.	Σ_1-EC l.b.	Σ_1-EC u.b.	Π_1-EC l.b.	Π_1-EC u.b.
$\cap$	NL, 4	NL	NL	NL, 4	NL	NL, 4
$\cup\,\cap$	P, 4	P	P	P, 4	P	P, 4
$\cap\,+$	$\mathrm{coC_{=}L}$, 3	$\mathrm{coC_{=}L}$	$\mathrm{coC_{=}L}$	$\mathrm{coC_{=}L}$, 5	coNP, 6	coNP, 6
$\cap\,\times$	$\mathrm{coC_{=}L}$, 3	P	$\mathrm{coC_{=}L}$	P, 21	coNP, 3	coNP, 3
$^{-}\,+$	PSPACE, 13	PSPACE	PSPACE	2EXPSPACE	PSPACE	2EXPSPACE
$^{-}\,\times$	PSPACE, 13	PSPACE	PSPACE	3EXPSPACE	PSPACE	3EXPSPACE
$\cup\,\cap\,^{-}$	P	P, 4	NP, 4	NP, 4	coNP, 4	coNP, 4
$\cup\,\cap\,+$	PSPACE, 6	PSPACE	PSPACE	PSPACE, 6	PSPACE	PSPACE, 6
$\cup\,\cap\,\times$	PSPACE, 6	PSPACE	PSPACE	PSPACE, 21	PSPACE	PSPACE, 6
$\cap\,+\,\times$	co$\mathcal{PIT}$, 22	co$\mathcal{PIT}$	co$\mathcal{PIT}$	co$\mathcal{PIT}$, 23	Π_1, 14	Π_1
$^{-}\,+\,\times$	PSPACE	$\mathcal{R}_{\mathrm{tt}}(\Sigma_1)$	Σ_1, 14	Σ_2	Π_1, 14	Π_2
$\cup\,\cap\,^{-}+$	PSPACE	PSPACE, 3	PSPACE	2EXPSPACE, 8	PSPACE	2EXPSPACE, 8
$\cup\,\cap\,^{-}\times$	PSPACE	PSPACE, 3	PSPACE	3EXPSPACE, 8	PSPACE	3EXPSPACE, 8
$\cup\,\cap\,+\times$	coNEXP, 3	coNEXP	coNEXP	coNEXP, 21	Π_1	Π_1, 14
$\cup\,\cap\,^{-}+\times$	$\mathrm{L}^{\mathrm{NEXP}}$, 16	$\mathcal{R}_{\mathrm{tt}}(\Sigma_1)$, 14	Σ_1	Σ_2, 14	Π_1	Π_2, 14

The results provide insights and improved complexity bounds for the following problems: $\mathrm{MC}(\cup,\cap,^{-},+,\times)$, $\mathrm{MC}(\cap,+,\times)$ studied in [22], $\mathrm{MC}_{\mathbb{Z}}(+,\times)$, $\mathrm{MC}_{\mathbb{Z}}(\cap,+,\times)$ studied in [32], and $\mathrm{EQ}(\cup,\cap,^{-},+,\times)$, $\mathrm{EQ}(+,\times)$, $\mathrm{EQ}(\cap,+,\times)$ studied in [11].

A challenging open problem is to improve the bounds for the problems $\mathrm{EC}(^{-},+,\times)$ and $\mathrm{EC}(\cup,\cap,^{-},+,\times)$. Here the state of knowledge is as follows (cf. Propositions 15, 17, and Corollary 16):

1. Both problems are equivalent to problems investigated in [22, 11]: $\mathrm{EC}(^{-},+,\times) \equiv_{\mathrm{m}}^{\log} \mathrm{MC}(^{-},+,\times)$ and $\mathrm{EC}(\cup,\cap,^{-},+,\times) \equiv_{\mathrm{m}}^{\log} \mathrm{MC}(\cup,\cap,^{-},+,\times) \equiv_{\mathrm{m}}^{\log} \mathrm{EQ}(\cup,\cap,^{-},+,\times)$.
2. Finding a decision algorithm and proving its correctness is at least as difficult as showing that Goldbach's conjecture is provable or refutable, which is an open problem.
3. The problems are either decidable or outside $\Sigma_1 \cup \Pi_1$.
4. The problems are $\leq_{\mathrm{m}}$-hard for Σ_1 if and only if they are $\leq_{\mathrm{m}}$-complete for $\mathcal{R}_{\mathrm{tt}}(\Sigma_1)$.

Another open problem is to improve the complexity bounds whenever we have one of the classes 2EXPSPACE and 3EXPSPACE as upper bound. The latter are consequences of the decidability of the Presburger and Skolem arithmetic. It is possible that more specific proof techniques can improve these bounds. By Lemma 2, Π_1-$\mathrm{EC}(\cup,\cap,^{-},\times)$ is equivalent to the complement of Σ_1-$\mathrm{MC}(\cup,\cap,^{-},\times)$, which has already been investigated in [13, 12].

A third open problem is to improve the bounds for $\mathrm{EC}(\cap,\times)$ and Σ_1-$\mathrm{EC}(\cap,\times)$. Both problems are equivalent to $\mathrm{MC}(\cap,\times)$, which has already been studied in [22].

Acknowledgements. We thank Klaus Reinhardt for sharing an unpublished proof for the $\leq_{\mathrm{m}}^{\mathrm{p}}$-hardness of $\mathrm{MC}(^{-},\times)$ for PSPACE, which we slightly adapted to show the $\leq_{\mathrm{m}}^{\log}$-hardness of $\mathrm{MC}^{+}(^{-},+)$ for PSPACE (Lemma 10), and for helpful discussions on the $\leq_{\mathrm{m}}^{\log}$-hardness

of MC($^-$, +) for PSPACE (Theorem 11). Moreover, we thank Jakob Sigmund for helpful discussions and contributions to the proof of Theorem 12.

References

1 M. Agrawal, N. Kayal, and N. Saxena. Primes is in P. *Annals of Mathematics*, 160:781–793, 2004.

2 E. Allender. Making computation count: Arithmetic circuits in the nineties. *SIGACT NEWS*, 28(4):2–15, 1997.

3 E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO Theoretical Informatics and Applications*, 30:1–21, 1996.

4 D. Barth, M. Beck, T. Dose, C. Glaßer, L. Michler, and M. Technau. Emptiness problems for integer circuits. Technical Report 17-012, Electronic Colloquium on Computational Complexity (ECCC), 2017.

5 A. Bès. A survey of arithmetical definability. *Soc. Math. Belgique*, pages 1–54, 2002.

6 H. Breunig. The complexity of membership problems for circuits over sets of positive numbers. In *International Symposium on Fundamentals of Computation Theory*, volume 4639 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2007.

7 M. Davis, H. Putnam, and J. Robinson. The decision problem for exponential Diophantine equations. *Annals of Mathematics*, 74(2):425–436, 1961.

8 T. Dose. Complexity of constraint satisfaction problems over finite subsets of natural numbers. In *41st International Symposium on Mathematical Foundations of Computer Science*, volume 58 of *Leibniz International Proceedings in Informatics*, pages 32:1–32:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

9 J. Ferrante and C. Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM J. Comput.*, 4:69–76, 1975.

10 J. Ferrante and C. W. Rackoff. *The Computational Complexity of Logical Theories*, volume 718 of *Lecture Notes in Mathematics*. Springer Verlag, 1979.

11 C. Glaßer, K. Herr, C. Reitwießner, S. D. Travers, and M. Waldherr. Equivalence problems for circuits over sets of natural numbers. *Theory of Computing Systems*, 46(1):80–103, 2010.

12 C. Glaßer, P. Jonsson, and B. Martin. Circuit satisfiability and constraint satisfaction around skolem arithmetic. In *12th Conference on Computability in Europe*, volume 9709 of *Lecture Notes in Computer Science*, pages 323–332. Springer, 2016.

13 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.

14 E. Grädel. Dominoes and the complexity of subclasses of logical theories. *Annals of Pure and Applied Logic*, 43(1):1–30, 1989.

15 O. Ibarra and S. Moran. Probabilistic algorithms for deciding equivalence of straight-line programs. *Journal of the ACM*, 30(1):217–228, 1983.

16 V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1):1–46, 2004.

17 D. E. Knuth. All questions answered. *Notices of the AMS*, 49(3):318–324, 2002.

18 J. Köbler, U. Schöning, and K. W. Wagner. The difference and the truth-table hierarchies for NP. *RAIRO Inform. Théor.*, 21:419–435, 1987.

19 D. König and M. Lohrey. Parallel identity testing for skew circuits with big powers and applications. *CoRR*, abs/1502.04545, 2015.

20 R. J. Lipton and N. K. Vishnoi. Deterministic identity testing for multivariate polynomials. In *Proceedings of the 14th Symposium on Discrete Algorithms*, pages 756–760. ACM/SIAM, 2003.

21 Y. V. Matiyasevich. Enumerable sets are diophantine. *Doklady Akad. Nauk SSSR*, 191:279–282, 1970. Translation in Soviet Math. Doklady, 11:354–357, 1970.

22 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.

23 C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

24 I. Pratt-Hartmann and I. Düntsch. Functions definable by arithmetic circuits. In *Conference on Mathematical Theory and Computational Practice*, volume 5635 of *Lecture Notes in Computer Science*, pages 409–418. Springer, 2009.

25 K. Reinhardt, 2016. Personal communication.

26 N. Saxena. Progress on polynomial identity testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:101, 2009.

27 N. Saxena. Progress on polynomial identity testing - II. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:186, 2013.

28 A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

29 A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.

30 J. Sigmund, 2016. Personal communication.

31 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC'73, pages 1–9, New York, NY, USA, 1973. ACM.

32 S. D. Travers. The complexity of membership problems for circuits over sets of integers. *Theoretical Computer Science*, 369(1-3):211–229, 2006.

33 K. Wagner. The complexity of problems concerning graphs with regularities (extended abstract). In *Proceedings of the Mathematical Foundations of Computer Science 1984*, pages 544–552, London, UK, UK, 1984. Springer-Verlag.

34 K. Yang. Integer circuit evaluation is PSPACE-complete. *Journal of Computer and System Sciences*, 63(2):288–303, 2001. An extended abstract of appeared at CCC 2000.

Another Characterization of the Higher K-Trivials*

Paul-Elliot Angles d'Auriac[1] **and Benoit Monin**[2]

1 **UPEC LACL, Creteil, France**
panglesd@lacl.fr
2 **UPEC LACL, Creteil, France**
benoit.monin@computability.fr

Abstract

In algorithmic randomness, the class of K-trivial sets has proved itself to be remarkable, due to its numerous different characterizations. We pursue in this paper some work already initiated on K-trivials in the context of higher randomness. In particular we give here another characterization of the non hyperarithmetic higher K-trivial sets.

1998 ACM Subject Classification Computability theory

Keywords and phrases Algorithmic randomness, higher computability, K-triviality, effective descriptive set theory, Kolmogorov complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.34

1 Introduction

Algorithmic randomness defines what it is for an infinite 0-1 valued sequence to be random. It takes its roots deeply in computability : lots of definition and techniques from pure computability are used in algorithmic randomness, as hierarchies, reducibilites and forcing constructions. The research in this field led to the identification of many different randomness notions, the most known being perhaps Martin-Löf randomness: a sequence is Martin-Löf random if it is in no Π^0_2 set $\bigcap_n \mathcal{U}_n$ where the Lebesgue measure of each $\mathcal{U}_n$ is smaller than 2^{-n}. The reader can refer to [21] and [8] for more details on algorithmic randomness. One of the main research area is to study how the different classes of random sequences relate. For a given such class of randoms, another important research area is to study the sets relative to which this class does not change. This is called *lowness for randomness*. For example, the class of K-trivials are exactly the low for Martin-Löf randomness. This class is defined as the set of infinite sequences having minimal (up to a constant) Kolmogorov complexity on their prefixes, that is the Kolmogorov complexity of a prefix should not be bigger than its length [1]. The class of K-trivials proved itself to be remarkable, due to its numerous very different characterizations [20], [12], [6], [1], [9].

Another field has a lot of interactions with computability theory : descriptive set theory. This field can be studied completely independently from recursion theory as in [14]. However, the study of descriptive set theory in close relation with computability appeared to be a fruitful approach. The mix of these two fields is called effective descriptive set theory and can be used to prove lots of results from the classical version of descriptive set theory. This is done mainly in [19]. Effective descriptive set theory also gave rise to higher computability.

* This work was partially supported by TARMAC.

[1] Here it is important that we use the so called *prefix-free Kolmogorov complexity*, as it is the case in general with algorithmic randomness.

The notion of computation for this field comes from a very logical point of view, far from any implementation. Nonetheless it is possible to give an intuition of higher computation which is closer to what computer scientists are used to: one can view a higher computation as a regular computation (by a Turing machine, say) where the steps of computation are carried through the computable ordinals. This new way of computing has several things in common with the classical one, and of course some differences (as the lack of continuity in a computation) that may cause trouble when trying to lift some computability theorems from the classical setting to the higher setting. The reader can refer to [22], [5] and [18] for more details on higher recursion theory. The reader can also see [11] for more information about what could be a Turing machine which keeps running over ordinal times of computation.

Algorithmic randomness naturally arises from mixing probability theory and computability. Following the same ideas, researchers defined notions of higher randomness, obtained analogously, but by considering higher computability instead of computability. After the founder paper of the field [13], a lots of advances were made by several researchers ([4], [3], [2], [10]). The reader can also refer to [5] and [18] for more details on higher randomness. One of the notion which has previously been studied and which is the core subject of the paper, is the notion of higher K-triviality, the direct higher analogue of K-triviality. In particular, we give in this paper a characterization of the non-Δ^1_1 higher K-trivials, by proving that they are exactly the sets that shrink the class $\mathrm{W}\Pi^1_1\mathrm{R}$ to the class Π^1_1-$\mathrm{ML}\langle\mathcal{O}\rangle$ when relativizing continuously. This characterization is specific to the higher setting: the randomness notions that are equivalent to $\mathrm{W}\Pi^1_1\mathrm{R}$ and Π^1_1-$\mathrm{ML}\langle\mathcal{O}\rangle$ in the lower setting, coincide.

2 Preliminaries

2.1 Notations

In this paper, we work in the space of infinite sequences of 0's and 1's, called the Cantor space, denoted by 2^ω. We call *strings* finite sequences of 0's and 1's and *sequences* or *sets* elements of the Cantor space. For a sequence A we write $A\upharpoonright_n$ to denote the string equal to the n first bits of A. The space of strings is denoted by $2^{<\omega}$ and the space of strings of length smaller than n is denoted by $2^{<n}$. For a string σ, we denote the set of sequences extending σ by $[\sigma]$.

The topology on Cantor space is generated by the basic intervals $[\sigma] = \{X \in 2^\omega \mid X \succ \sigma\}$ for any string σ. For a set of strings $W \subseteq 2^{<\omega}$, we let $[W] = \bigcup_{\sigma\in W}[\sigma]$. A set of string W is said to be prefix-free if no string in W is a prefix of another string in W.

For $\mathcal{A} \subseteq 2^\omega$ Lebesgue-measurable, $\lambda(\mathcal{A})$ denotes the Lebesgue measure of $\mathcal{A}$, which is the unique Borel measure such that $\lambda([\sigma]) = 2^{-|\sigma|}$ for all strings σ.

We assume that the reader is familiar with basic notions of computability. For $A, B \in 2^\omega$ we write $A \leq_T B$ if A is Turing reducible to B. We denote by $\emptyset'$ the halting problem. We also assume that the reader is familiar with the basics of effective descriptive set theory, in particular with the notations $\Sigma^0_1, \Pi^0_1, \Sigma^0_2, \Pi^0_2, etc...$

We finally also assume that the reader is familiar with the notion of Kolmogorov complexity. In this paper, we will only consider a prefix-free version of the Kolmogorov complexity (used in the definition of K-triviality): using compressors $M : 2^{<\omega} \to 2^{<\omega}$ such that the domain of M is prefix-free. It is an easy exercise to show that an optimal prefix-free compressor exists (optimal up to a constant of course).

2.2 Background on algorithmic randomness

In 1966, Martin-Löf gave in [16] a definition capturing elements of the Cantor space that can be considered 'random'. Many nice properties of the Martin-Löf random sequences make this notion of randomness one of the most interesting and one of the most studied.

Intuitively a random sequence should not have any atypical property. A property is here considered atypical if the set of sequences sharing this property is of measure 0. It also makes sense to consider only properties which can be described in some effective way (because any X has the property of being in the set $\{X\}$ and thus nothing would be random).

▶ **Definition 1.** An intersection of measurable sets $\bigcap_n \mathcal{A}_n$ is said to be *effectively of measure* 0 if the function which to n associates the measure of $\mathcal{A}_n$ is bounded by 2^{-n}. A *Martin-Löf test*, or an ML-test is a Π^0_2 set $\bigcap_n \mathcal{U}_n$ effectively of measure 0. We say that $X \in 2^\omega$ is *Martin-Löf random* if it is in no Martin-Löf test. The class of Martin-Löf randoms is also referred to as the class MLR.

The requirement for a Martin-Löf test to be effectively of measure 0 is important and leads to very nice properties. In particular there exists a universal Martin-Löf test, i.e. a test containing all the others (see [16]). This is not the case anymore if we drop the 'effectively of measure 0' condition. Instead we get a notion known as weak-2-randomness.

▶ **Definition 2.** A Π^0_2 nullset is called a *weak-2 test* or a W2 test. We say that $X \in 2^\omega$ is *weakly-2-random* if it is in no weak-2 test. The class of weakly-2-randoms is also referred to as the class W2R.

As a randomness notion, weak-2-randomness is a strictly stronger than 1-randomness: tests can capture more elements and thus there are fewer randoms. For any given randomness notion, it makes sense to relativize it to any oracle:

▶ **Definition 3.** Let $A \in 2^\omega$. An ML^A test is a $\Pi^0_2(A)$ set $\bigcap_n O_n$ effectively of measure 0. We say that $X \in 2^\omega$ is MLR^A if it is in no ML^A test. Similarly a $\mathrm{W2}^A$ test is a $\Pi^0_2(A)$ nullset, and we say that X is $\mathrm{W2R}^A$ if it is in no $\mathrm{W2}^A$ test.

A nice characterization of W2R has been given from restricting the relativization $\mathrm{MLR}^{\emptyset'}$: we can only use $\emptyset'$ to find the indices of the open sets in a test.

▶ **Definition 4.** Let $(W_e)_{e\in\omega}$ be an effective enumeration of the c.e. sets of strings. A $\mathrm{ML}\langle\emptyset'\rangle$ test is a set $\bigcap_n [W_{f(n)}]$ with $\lambda([W_{f(n)}]) \leq 2^{-n}$ were $f : \omega \to \omega$ is computable from $\emptyset'$. A set is $\mathrm{MLR}\langle\emptyset'\rangle$ if it is in no $\mathrm{ML}\langle\emptyset'\rangle$ test.

Note that with the full relativization of an ML test to A, the oracle A itself is not needed to find the index of the n-th $\Sigma^0_1(A)$ open set of the test: the use of A for that can be swallowed in the process of enumerating each $\Sigma^0_1(A)$ component of the test.

Going back to the previous definition, we have the following easy theorem:

▶ **Theorem 5** ([2], section 7)**.** $\mathrm{W2R} = \mathrm{ML}\langle\emptyset'\rangle$.

Proof. Let's start with $\mathrm{W2R} \subseteq \mathrm{ML}\langle\emptyset'\rangle$, given a $\mathrm{ML}\langle\emptyset'\rangle$ test $\bigcap_n \mathcal{U}_{f(n)}$, we will show that it is included in a W2R test. We define $V_{\langle m,t\rangle} = \bigcup_{s\geq t} \mathcal{U}_{f_s(m)}$. As $\bigcap_n V_n = \bigcap_n \mathcal{U}_{f(n)}$, we have that $\lambda(\bigcap_n \mathcal{V}_n) = 0$, so this is a W2R test.

Now, let $\bigcap_n \mathcal{U}_n$ be a Π^0_2 nullset, one can use $\emptyset'$ to find uniformly in n the first $m = f(n)$ such that $\lambda(\mathcal{U}_m) \leq 2^{-n}$. ◀

The sets relative to which $\mathrm{MLR}^A = \mathrm{MLR}$ have been extensively studied, and have been identified as the class of K-trivial sets.

▶ **Definition 6.** A set $A \in 2^\omega$ is K-trivial if for any n, the prefix-free Kolmogorov complexity of $A\restriction_n$ is smaller than the prefix-free Kolmogorov complexity of n (up to a constant).

The reader can refer to [21] for more details on the K-trivials. They are also the sets relative to which $\mathrm{W2R}^A = \mathrm{W2R}$:

▶ **Theorem 7** ([20], [15], [7])**.** *The following are equivalent for a set A:*
1. *A is K-trivial*
2. $\mathrm{W2R}^A = \mathrm{W2R}$
3. $\mathrm{MLR}^A = \mathrm{MLR}$

As we will see, this characterization fails in the higher setting, but it fails in a way that will help us provide another characterization of the higher K-trivials.

2.3 Background on higher computability

We assume that the reader is familiar with the concepts of Δ^1_1, Π^1_1 and Σ^1_1 subsets of ω and of 2^ω. A known result is that an open set $\mathcal{U}$ is Π^1_1 if and only if there exists a Π^1_1 set of strings W such that $\mathcal{U} = [W]^\prec$.

There is a strong analogy between classical concepts in computability (referred to as the *lower setting*) and their analogue in higher computability (referred to as the *higher setting*). For instance, Δ^1_1 can be seen as a higher analogue of computable, and Π^1_1 can be seen as a higher analogue of computably enumerable, with the difference that the times at which elements are enumerated are now computable ordinals.

We refer to the set of codes for computable ordinals (using whichever equivalent coding) as Kleene's $\mathcal{O}$. As usual, the smallest non-computable ordinal is denoted by ω_1^{CK}.

We recall here a few definitions about continuous higher Turing reductions. In [2] (Definition 1.1) higher Turing reductions are defined to compute elements of 2^ω. In [10] (section 3.2) this definition is extended in a straightforward way, to compute elements of $(\omega_1^{CK})^\omega$. We also extend this definition here in a straightforward way, to compute elements of $(\omega_1^{CK})^{\omega_1^{CK}}$.

An absolutely formal definition of computations of functions from ω_1^{CK} to ω_1^{CK} should either use the language of set theory and deals with actual ordinals, or use a unique notation system for computable ordinals. There exists such a Π^1_1 notation system $\mathcal{O}_1 \subseteq \omega$ (see [22] or [18], 3.6.1) and up to this notation system, one can view a function from ω_1^{CK} to ω_1^{CK} as a function from $\mathcal{O}_1$ to $\mathcal{O}_1$, and thus simply defined on integers.

▶ **Definition 8** ([10] [2])**.** We say that A higher Turing computes (or higher computes) $f : \omega_1^{CK} \mapsto \omega_1^{CK}$ (respectively $g : \omega \mapsto \omega_1^{CK}$) if there exists a Π^1_1 set $C \subseteq 2^{<\omega} \times \omega_1^{CK} \times \omega_1^{CK}$ (respectively $C \subseteq 2^{<\omega} \times \omega \times \omega_1^{CK}$) such that $f(o_1) = o_2$ iff $\exists \sigma \prec A\ (\sigma, o_1, o_2) \in C$ (respectively $g(n) = o$ iff $\exists \sigma \prec A\ (\sigma, n, o) \in C$). We say that A higher Turing computes $B \in 2^\omega$ if A higher Turing computes the characteristic function of B.

In [2] it is shown that Kleene's $\mathcal{O}$ higher Turing computes a set $A \in 2^\omega$ iff $\mathcal{O}$ Turing computes A.

2.4 Background on higher randomness

Higher randomness goes back to Martin-Löf who promoted the notion of Δ^1_1-randomness (already defined by Sacks [22]), defending the idea that it was the appropriate mathematical concept of randomness [17]. Even if his first definition undoubtedly became the most

successful over the years, this other definition recently got a second wind on the initiative of Hjorth and Nies who started to study the analogy between the usual notions of randomness and their higher counterparts. In order to do so they created in [13] a higher analogue of Martin-Löf randomness.

▶ **Definition 9** (Hjorth, Nies)**.** A Π^1_1-Martin-Löf test, or a Π^1_1-ML test, is given by an effectively null intersection of open sets $\bigcap_n \mathcal{U}_n$, each $\mathcal{U}_n$ being Π^1_1 uniformly in n. A sequence X is Π^1_1-Martin-Löf random if it is in no Π^1_1-Martin-Löf test. The class of Π^1_1-Martin-Löf randoms is also referred to as the class Π^1_1-MLR.

The higher analogue of weak-2-randomness has also been studied (see [4] [2]):

▶ **Definition 10.** We say that X is *weakly-Π^1_1-random* if it belongs to no $\bigcap_n \mathcal{U}_n$ with each $\mathcal{U}_n$ open set Π^1_1 uniformly in n and with $\lambda(\bigcap_n \mathcal{U}_n) = 0$. The class of weakly-$\Pi^1_1$-randoms is also referred to as the class $\mathrm{W}\Pi^1_1\mathrm{R}$.

It is also possible to define an analogue of $\mathrm{MLR}\langle\emptyset'\rangle$ in the higher setting, using Kleene's $\mathcal{O}$ in place of $\emptyset'$.

▶ **Definition 11.** Let $(W_e)_{e\in\omega}$ be an enumeration of the Π^1_1 sets of strings. A Π^1_1-$\mathrm{ML}\langle\mathcal{O}\rangle$ test is a set $\bigcap_n [W_{f(n)}]$ with $\lambda([W_{f(n)}]) \leq 2^{-n}$ were $f : \omega \to \omega$ is Turing computable from Kleene's $\mathcal{O}$. A set is Π^1_1-$\mathrm{MLR}\langle\mathcal{O}\rangle$ if it is in no Π^1_1-$\mathrm{ML}\langle\mathcal{O}\rangle$ test.

Theorem 5 does not lift to the higher setting. The proof in the lower setting uses what has been defined in [2] to be a 'time trick': we use the fact that time and space are the same objects: the natural numbers. In the higher setting, this is not anymore true as the time goes along the ordinals. It is in fact possible to show that the class Π^1_1-$\mathrm{MLR}\langle\mathcal{O}\rangle$ is strictly contained in the class $\mathrm{W}\Pi^1_1\mathrm{R}$. To be more specific, let us introduce maybe the most important notion of higher randomness, first given by Sacks, and made possible by a theorem of Lusin saying that even though Π^1_1 sets are not necessarily Borel, they remain all measurable.

▶ **Definition 12** (Sacks)**.** We say that $X \in 2^\omega$ is Π^1_1-Random if it is in no Π^1_1 nullset.

We have the following:

▶ **Theorem 13** ([2])**.** Π^1_1-$\mathrm{MLR}\langle\mathcal{O}\rangle \subsetneq \Pi^1_1$-Randoms $\subsetneq \mathrm{W}\Pi^1_1\mathrm{R} \subsetneq \Pi^1_1$-MLR

We finally give another characterization of Π^1_1-$\mathrm{ML}\langle\mathcal{O}\rangle$, that has no counterpart in the lower setting (with $\mathrm{ML}\langle\emptyset'\rangle$ in place of Π^1_1-$\mathrm{ML}\langle\mathcal{O}\rangle$), and which will be useful in the paper.

▶ **Property 14** ([2])**.** *The following are equivalent for a sequence $X \in 2^\omega$:*

1. *X is Π^1_1-$\mathrm{ML}\langle\mathcal{O}\rangle$ random*
2. *X does not belong to any test $(\mathcal{U}_s)_{s<\omega_1^{CK}}$ not necessarily nested where each $\mathcal{U}_s$ is a Π^1_1 open set uniformly in s, and such that $\lambda(\bigcap_{s<\omega_1^{CK}} \mathcal{U}_s) = 0$*

2.5 Continuous relativization of higher randomness

It is also possible to define an analogue of K-triviality in the higher setting. The higher K-trivials are defined analogously, but using a version of Kolmogorov complexity with Π^1_1-prefix-free compression machines.

▶ **Definition 15** ([13]). We define:

- The *higher prefix-free Kolmogorov complexity* is given by $K(y) = \min\{|\sigma| : U(\sigma) = y\}$ for U the universal prefix-free Π^1_1-machine given by $U(0^e1\tau) = M_e(\tau)$ and $(M_e)_{e\in\omega}$ a uniform enumeration of the Π^1_1 prefix-free machines.
- A is *higher K-trivial* if $\exists b\ \forall n\ K(A \upharpoonright n) \leq K(n) + b$.

However, for the higher K-trivials to also be low for Π^1_1-MLR, one has to be careful about the way things are relativized to oracles. In higher computability we don't have anymore the continuity aspect of the lower setting : if B is $\Delta^1_1(A)$, it does not mean that a finite quantity of A suffices to know a finite quantity of B. However, we can force this state of things, as done previously with the notion of higher Turing computations. We next define what it means to relativize the notion of Π^1_1 set, continuously to an oracle.

▶ **Definition 16** ([2]). An oracle-continuous Π^1_1 set of integers is given by a set $W \subseteq 2^{<\mathbb{N}} \times \mathbb{N}$. For a string σ we write W^σ to denote the set $\{n : \exists \tau \prec \sigma, (\tau, n) \in W\}$. For a sequence X we write W^X to denote the set $\{n : \exists \tau \prec X, (\tau, n) \in W\}$. The set W^X is then called an X-continuous Π^1_1 set of integers.

An open set $\mathcal{U}$ is X-continuously Π^1_1 if there is an X-continuously Π^1_1 set of strings W such that $\mathcal{U} = [W^X]$.

We are now ready to define continuous relativization of randomness notions :

▶ **Definition 17.** If A is a set, we say that X is $\mathrm{WII}^1_1\mathrm{R}^A$ if it is in no $\mathcal{U} = \bigcap_n \mathcal{U}_n$ where $(\mathcal{U}_n)_{n\in\omega}$ is a uniform family of A-continuous Π^1_1 open sets, such that $\lambda(\mathcal{U}) = 0$. We say that X is $\Pi^1_1\text{-MLR}^A$ if it is in no $\bigcap_n \mathcal{U}_n$ where $(\mathcal{U}_n)_{n\in\omega}$ is a uniform family of A-continuous Π^1_1 open sets, such that $\lambda(\mathcal{U}_n) \leq 2^{-n}$.

We now have the following:

▶ **Theorem 18** ([2]). *The higher K-trivials are exactly the low for* Π^1_1-MLR*, using continuous relativization.*

Unlike in the lower setting, the higher K-trivials are not anymore the low for $\mathrm{WII}^1_1\mathrm{R}$. For A Δ^1_1 (a special case of being higher K-trivial), it is still obviously the case that $\mathrm{WII}^1_1\mathrm{R}^A = \mathrm{WII}^1_1\mathrm{R}$. But if A is K-trivial and not Δ^1_1, we will actually see that $\mathrm{WII}^1_1\mathrm{R}^A = \Pi^1_1\text{-ML}\langle\mathcal{O}\rangle$.

3 Another characterization of the higher K-trivials

3.1 Collapsing approximations

When trying to lift the Δ^0_2 definitions from the lower to the higher setting, some new possibilities appear. In the lower setting, for an approximation of A the set $\{A_t : t < s\}$ is always finite as s ranges over the natural numbers. So in particular it is closed. At the contrary, when s is an ordinal, the set $\{A_t : t < s\}$ may not have this property, which leads us to define a different type of approximation, which depends on the topological properties of $\{A_t : t < s\}$.

▶ **Property/Definition 19** ([2]).

1. *A sequence A is higher Δ^0_2 if it satisfies the following equivalent properties :*
   - **(a)** $A \leq_T \mathcal{O}$
   - **(b)** *There is a higher computable sequence $(A_s)_{s<\omega_1^{CK}}$ with $\lim_{s\to\omega_1^{CK}} A_s = A$*
2. *A computable approximation $(A_s)_{s<\omega_1^{CK}}$ converging to A is said to be collapsing if for every stage s, the set A is not in the closure of $\{A_t : t < s\}$.*

Such approximations are called collapsing, because they can be used to "collapse" ω_1^{CK} to a computable ordinal in a strong way: such approximations can be used to compute an ω-sequence of computable ordinals, with ω_1^{CK} as a supremum:

▶ **Property 20** ([2]). *Every sequence A with a collapsing approximation, higher Turing computes a function $f : \omega \to \omega_1^{CK}$ which is cofinal in ω_1^{CK}.*

In classical computability, given an effectively open set $\mathcal{U}$, it is uniformly possible to obtain a c.e. and prefix-free set W such that $[W] = \mathcal{U}$. However, in the setting of higher computability, it can be proved that this is no more possible: there is a Π^1_1 open set $\mathcal{U}$ such that for every prefix-free Π^1_1 set of strings W, $\mathcal{U} \neq [W]$. But working relative to some sets that have a collapsing approximation allows us to use time tricks, and in a way brings us "closer" to classical computability.

▶ **Property 21** ([2],end of page 20). *If A has a collapsing approximation $(A_s)_{s<\omega_1^{CK}}$, and $\mathcal{U}$ is an oracle-continuous Π^1_1 open set, then there exists an oracle-continuous Π^1_1 set $W \subseteq 2^{<\omega} \times 2^{<\omega}$ such that $\mathcal{U}^A = [W^A]$ and for all B, the set W^B is prefix-free.*

Proof. We define an enumeration of a Π^1_1 oracle-continuous set W. The enumeration of W will compute throughout its stages a collapsing approximation $(A_s)_{s<\omega_1^{CK}}$ of A. At stage s, if A_s is not in the closure of $\{A_t\}_{t<s}$, then let $\tau \prec A_s$ be the smallest such that τ has never been a prefix of some A_t for $t < s$. Then enumerate into W^τ_{s+1} all strings σ of length smaller than or equal to $|\tau|$ such that $[\sigma] \subseteq \mathcal{U}^\tau_s$ but $[\sigma]$ is disjoint from $[W^\tau_s]$.

It is clear that $[W^A] \subseteq \mathcal{U}^A$. Let us argue that $\mathcal{U}^A \subseteq [W^A]$. Suppose $\sigma \in \mathcal{U}^A$. There are sequences $\{\tau_n\}_{n\in\omega}$ and $\{s_n\}_{n\in\omega}$ such that for every n, the ordinal s_n is the first for which we have $A_{s_n}\restriction_{|\tau_n|} = A\restriction_{|\tau_n|} = \tau_n$, and such that $\sup_n s_n = \omega_1^{CK}$.

Let n be the smallest such that $|\tau_n| > |\tau_{n-1}| \geq |\sigma|$ and such that $\sigma \in \mathcal{U}^{\tau_n}_{s_n}$. Then we have by construction that $\sigma \subseteq [W^{\tau_n}_{s_n+1}]$. Therefore $[W^A] = \mathcal{U}^A$. Also by construction W^B is prefix-free for every B. ◀

3.2 Properties of higher K-Trivials

One key property of the higher K-trivial sequences is that they have a collapsing approximation as long as they are not Δ^1_1.

▶ **Property 22** ([2]). *Every higher K-trivial, but not Δ^1_1, sequence has a collapsing approximation.*

▶ **Corollary 23.** *If A is higher K-trivial but not Δ^1_1, then:*

- *A higher Turing computes a function $f : \omega \to \omega_1^{CK}$ whose range is unbounded in ω_1^{CK} ;*
- *if $\mathcal{U}$ is an oracle-continuous Π^1_1 open set, one can uniformly find a Π^1_1 oracle-continuous set of strings W such that $\mathcal{U}^A = [W^A]$ and $\forall B \in 2^\omega$, W^B is prefix-free.*

Proof of the corollary. By property 22, together with property 20 and 21. ◀

The proof that the low for Martin-Löf randoms are exactly the K-trivials requires a big machinery. Using the fact that higher K-trivials have a collapsing approximation, it is possible to transpose this proof and to show that the continuously low for Π^1_1-MLR are the higher K-trivials. The machinery developed in this proof can also be used to show a slightly more general statement, known in the lower setting as the "Main Lemma". One can find a detailed proof and explanation of this result for the higher setting in [2]. We give here a version of the Main Lemma which is closer to our need than the one in [2] (using oracle-continuous open sets in place of oracle-continuous discrete semi-measures) :

▶ **Theorem 24** (Main Lemma). *If A is higher K-trivial, $(A_s)_{s<\omega_1^{CK}}$ is any collapsing approximation of A, and W is an oracle-continuous Π^1_1 set of strings such that there exists $c \in \omega$ such that for all X we have $\sum_{\sigma \in W^X} 2^{-|\sigma|} \leq c$, then there exists a higher computable function $q : \omega_1^{CK} \to \omega_1^{CK}$ such that:*

$$S = \sum_{r<\omega_1^{CK}} \sum_{\sigma \in E_r} 2^{-|\sigma|} \text{ is finite}$$

where

$$E_r = \left\{ \sigma : \begin{array}{l} \sigma \in W^A[q(r)] \text{ with use } u, \text{ and} \\ A[q(r)] \upharpoonright u \neq A[q(r+1)] \upharpoonright u \end{array} \right\}.$$

Intuitively, if A is higher K-trivial, we can slow down its approximations in such a way that not too much measure is added in the open set, with pieces of oracle that were believed at some point to be prefixes of A but in fact are not: the total sum of 'wrong' measure added this way over the times of computation can be made finite.

3.3 A higher K-trivial and not Δ^1_1 implies Π^1_1-ML$\langle\mathcal{O}\rangle = $ WΠ^1_1R^A

▶ **Theorem 25.** *If A is higher K-Trivial and not Δ^1_1, then* WΠ^1_1R$^A \subseteq \Pi^1_1$-MLR$\langle\mathcal{O}\rangle$.

Proof. Fix an A. By contrapositive, we prove that if X is captured by a Π^1_1-ML$\langle\mathcal{O}\rangle$ test, then it is also captured by a WΠ^1_1R^A test. We use the characterization 14 of Π^1_1-ML$\langle\mathcal{O}\rangle$ tests, so let $\mathcal{U} = \bigcap_{s<\omega_1^{CK}} \mathcal{U}_s$ be such a test.

We make use of the corollary 23 that A higher computes a function f with cofinality ω_1^{CK}. Let $g(\langle m, n\rangle)$ be the m-th element of $\mathcal{O}_{\leq f(n)} \subseteq \mathbb{N}$ (where $\mathcal{O}_{\leq \alpha}$ is the set of codes for computable ordinals smaller than α). Then g is also higher computable from A, and its range is all the computable ordinals. Now, we consider $\bigcap_n \mathcal{U}_{g(n)}$. As the range of g is ω_1^{CK}, the intersection is equal to $\mathcal{U}$, so its measure is 0 and as g is higher computable from A, this set is a WΠ^1_1R^A test. ◀

The other inclusion will be a corollary of a more general theorem, whose proof follows the same spirit than the proof in the lower setting that K-trivials are low for W2R.

▶ **Theorem 26.** *Let A be higher K-trivial. Let $G = \bigcap_n \mathcal{U}_n$ where $(\mathcal{U}_n)_{n\in\omega}$ is a uniform family of Π^1_1 open sets, continuously in A. Then there exists a set $S = \bigcap_{s<\omega_1^{CK}} \mathcal{V}_s$ where $(\mathcal{V}_s)_{s<\omega_1^{CK}}$ is a uniformly Π^1_1 family of open sets, such that $\lambda(S) = \lambda(G)$ and $S \supseteq G$.*

We will first prove the result for the simplest G, that is when the family is reduced to a single open set $\mathcal{U}$, and then extend this result to a uniform countable intersection of such open sets.

▶ **Lemma 27.** *Let A be higher K-trivial. Let G be a A-continuously Π^1_1 open set. Then there exists a set $S = \bigcap_{s<\omega_1^{CK}} \mathcal{V}_s$ where $(\mathcal{V}_s)_{s<\omega_1^{CK}}$ is a uniformly Π^1_1 family of open sets, such that $\lambda(S) = \lambda(G)$ and $S \supseteq G$.*

Proof. Using the property 21, there exists an oracle-continuous Π^1_1 set of strings W such that $G = [W^A]$, and such that W^B is prefix-free for all B.

If A is Δ^1_1 we are done. Otherwise, as it is higher K-trivial, it has a collapsing approximation, so we can try to use it to approximate G with Π^1_1 open sets $\mathcal{V}_s$. A first candidate for $\mathcal{V}_s$ could be $\bigcup_{s\leq r<\omega_1^{CK}} W^A[r]$, because every such $\mathcal{V}_s$ would contain G, but this approximation

is "too large", because W and approximations of A can be such that $W^A[s]$ enumerates the empty word for a family of s cofinal in ω_1^{CK}.

The trick to prevent the measure to increase is to restrain the computation of $W^A[s]$ only to some special stages and parts of the oracle, so that the weight of all errors is finite. These stages are given by the Main Lemma : let $q : \omega_1^{CK} \to \omega_1^{CK}$ be the function given by the Main Lemma, applied to W. We then have:

$$\sum_{r<\omega_1^{CK}} \sum_{\sigma\in E_r} 2^{-|\sigma|} \text{ is finite}$$

where

$$E_r = \left\{ \sigma : \begin{array}{l} \sigma \in W^A[q(r)] \text{ with use } u, \text{ and} \\ A[q(r)] \upharpoonright u \neq A[q(r+1)] \upharpoonright u \end{array} \right\}.$$

Now we define $\mathcal{V}_s$ by computing only over the special stages and prefixes, that is

$$\sigma \in \mathcal{V}_s \Leftrightarrow \exists r \geq s \text{ such that } \sigma \in W^A[q(r)].$$

Every $\mathcal{V}_s$ contains G as any string σ enumerated in W^A, with use u, will be in every $W^A[q(r)]$ for $r \geq t$ such that $A[q(t+1)]$ has settled on $A \upharpoonright u$.

Now consider the errors of the $\mathcal{V}_s$, that is the strings σ enumerated in $\mathcal{V}_s$ but such that $[\sigma] \not\subseteq G$. There must exists an $r \geq s$ such that $\sigma \in W^A[q(r)]$ with use u, and such that $A[q(r)] \upharpoonright u \neq A[q(r+1)] \upharpoonright u$. Then $\sigma \in E_r$ for some $r \geq s$. It follows that:

$$\lambda(\mathcal{V}_s \setminus G) \leq \sum_{s\leq r<\omega_1^{CK}} \sum_{\sigma\in E_r} 2^{-|\sigma|}.$$

But as the total sum is finite, the partial sum goes to zero as s increases:

$$\lim_{s\to\omega_1^{CK}} \lambda(\mathcal{V}_s \setminus G) = 0.$$

Finally with $S = \bigcap_{s<\omega_1^{CK}} \mathcal{V}_s$, we have $\lambda(S \setminus G) = 0$ and $S \supseteq G$, which concludes the proof of the lemma. ◀

proof of Theorem 26. It remains to prove using this lemma the more general case when $G = \bigcap_{n\in\omega} \mathcal{U}_n$. We can apply what we just proved to $R = \bigcup_{e\in\omega} 0^e 1[W_e]$ where $(W_e)_{e\in\omega}$ is an effective listing of the A-continuous Π^1_1 sets. We then find $T = \bigcap_{s<\omega_1^{CK}} T_s$ with $T \supseteq R$ and $\lambda(T) = \lambda(R)$.

Let f be a computable function such that $\mathcal{U}_n = [W_{f(n)}]$. Writing $A|w = \{X : wX \in A\}$, we let $S = \bigcap_{n\in\omega}(T \mid 0^{f(n)}1)$. Let us show that S works for our purpose. First S is a Π^1_1-ML$\langle\mathcal{O}\rangle$ test, by the characterization 14 of these tests, as

$$S = \bigcap_{n\in\omega} \left(\left(\bigcap_{s<\omega_1^{CK}} T_s \right) \mid 0^{f(n)}1 \right) = \bigcap_{\omega s+n<\omega_1^{CK}} T_s \mid 0^{f(n)}1.$$

Then $S \supseteq G$ as for every n, we have $T \mid 0^{f(n)}1 \supseteq R \mid 0^{f(n)}1 = [W_{f(n)}] = \mathcal{U}_n \supseteq G$. Finally, we show that $\lambda(S \setminus G) = 0$. We have:

$$S - G = \bigcap_{n\in\omega} S - [W_{f(n)}] \subseteq \bigcap_{n\in\omega} T \mid 0^{f(n)}1 - [W_{f(n)}].$$

But $\lambda(T \mid 0^{f(n)}1 - [W_{f(n)}]) \leq 2^{f(n)+1}\lambda(T - R) = 0$, so finally $\lambda(S - G) = 0$. ◀

▶ **Corollary 28.** *If A is higher K-Trivial and not Δ^1_1, then $\mathrm{W}\Pi^1_1\mathrm{R}^A \supseteq \Pi^1_1\text{-ML}\langle\mathcal{O}\rangle$.*

Proof. We proceed by contrapositive, and show that every $\mathrm{W}\Pi^1_1\mathrm{R}^A$ test G is included in a $\Pi^1_1\text{-ML}\langle\mathcal{O}\rangle$ test. Given a $\mathrm{W}\Pi^1_1\mathrm{R}^A$ test, we just apply the theorem to this test and get $S = \bigcap_{s<\omega_1^{CK}} \mathcal{V}_s$ such that $S \supseteq G$ and $\lambda(S) = \lambda(G) = 0$, that is S is a $\Pi^1_1\text{-ML}\langle\mathcal{O}\rangle$ test containing G. ◀

3.4 $\Pi^1_1\text{-ML}\langle\mathcal{O}\rangle = \mathrm{W}\Pi^1_1\mathrm{R}^A$ implies A higher K-trivial and not Δ^1_1

In this section, we will suppose that A is not higher K-trivial, and we will prove that under this assumption there exists a $\mathrm{W}\Pi^1_1\mathrm{R}^A$ sequence that is not $\Pi^1_1\text{-ML}\langle\mathcal{O}\rangle$ random. To do this we need the existence of a particular set, that will allow us to build a specific sequence by forcing.

This proof follows the lines of the proof of lowness for Π^1_1-randomness [10]: if A is not Δ^1_1 and not higher K-trivial, then there exists a Π^1_1-ML test relative to A, which captures a Π^1_1-random. In [10] the proof has been done using full relativization and not continuous relativization. Full relativization helps in particular to work with tests whose captured sequences are closed under suppression of prefixes. It is not necessarily obvious using continuous relativization that we can work with such tests. In particular, for some oracles A it might be the case that there is no universal Π^1_1-ML test continuously relativized to A. Thus we first need to show the following lemma:

▶ **Lemma 29.** *Let A be any set, and $\mathcal{U}$ a $\Pi^1_1\text{-ML}^A$ test. Then there exists an $\Pi^1_1\text{-ML}^A$ test $\mathcal{V}$ such that if $\sigma X \in \mathcal{U}$ then $X \in \mathcal{V}$.*

Proof. First we establish some notation. For $\mathcal{A} \subseteq 2^\omega$, we write $\mathcal{A}^{-n}$ for $\{X : \exists \sigma \in 2^n, \sigma X \in \mathcal{A}\}$ that is the set of strings of $\mathcal{A}$, for which we remove the first n bits. We remark that $\lambda(\mathcal{A}^{-n}) \leq 2^n\lambda(\mathcal{A})$. Now say $\mathcal{U} = \bigcap_n \mathcal{U}_n$ with $\lambda(\mathcal{U}_m) \leq 2^{-m}$ and $(\mathcal{U}_m)$ is uniformly Π^1_1-open, continuously in A. We define $\mathcal{V} = \bigcap_n \mathcal{V}_n$ by :

$$\mathcal{V}_n = \{X : \exists m > n, \exists \sigma \in 2^{<m}, \sigma X \in \mathcal{U}_{2m}\} = \bigcup_{m>n} \bigcup_{i<m} (\mathcal{U}_{2m})^{-i}.$$

We now only need to verify that this proves the theorem. We need this set to be a $\Pi^1_1\text{-ML}^A$ test. It is easily a uniform intersection of Π^1_1 open sets continuously in A, but we need to check that it is effectively of measure 0. We have

$$\lambda(\mathcal{V}_n) \leq \sum_{m>n} \sum_{i<m} \lambda(\mathcal{U}_{2m}^{-i}) \leq \sum_{m>n} \sum_{i<m} 2^i \lambda(\mathcal{U}_{2m})$$

by the remark after the definition of $\mathcal{A}^{-n}$, and then

$$\lambda(\mathcal{V}_n) \leq \sum_{m>n} \sum_{i<m} 2^i 2^{-2m} \leq \sum_{m>n} 2^m 2^{-2m} \leq \sum_{m>n} 2^{-m} \leq 2^{-n}.$$

So $\mathcal{V}$ is a test. ◀

▶ Remark. We did not proved that the test $\bigcap \mathcal{V}_n$ is closed under deletion of prefixes. It's own closure under deletion of prefixes may need to be bigger, but this state of things will be enough for our needs.

Recall we will suppose in this part that A is not higher K-trivial. The next lemma makes use of this fact to define a set that will be useful in our next construction.

▶ **Lemma 30.** *If A is not higher K-trivial, then there exists a Π^1_1-ML^A test $\bigcap_n \mathcal{U}_n$ such that for every n and every Π^1_1 open set $\mathcal{V}$ with $\lambda(\mathcal{V}) < 1$, we have $\mathcal{U}_n \cap \mathcal{V}^c \neq \emptyset$ (that is $\mathcal{U}_n \not\subseteq \mathcal{V}$).*

Proof. By contrapositive, we will show that if the conclusion of the theorem does not hold, then every Π^1_1-ML^A test is contained in a Π^1_1-ML test. As the sequences which are continuously low for Π^1_1-MLR are exactly the higher K-trivials (Theorem 18), we can conclude that A is higher K-trivial. Following this plan, our hypothesis becomes: "For every Π^1_1-ML^A test $\cap_n \mathcal{U}_n$ there exists n and a Π^1_1 open set $\mathcal{V}$ with $\lambda(\mathcal{V}) < 1$ and such that $\mathcal{U}_n \subseteq \mathcal{V}$."

Our goal is to show that A is low for Π^1_1-MLR. Let $\mathcal{U} = \bigcap \mathcal{U}_n$ be a Π^1_1-ML^A test. By the previous lemma, we find a test $\widetilde{\mathcal{U}} = \bigcap \widetilde{\mathcal{U}_n}$ containing all the suffixes of elements in $\mathcal{U}$. Then by the hypothesis, we find $\mathcal{V} \supseteq \bigcap \widetilde{\mathcal{U}_n} (\supseteq \bigcap \mathcal{U}_n)$ where $\mathcal{V}$ is Π^1_1 open and $\lambda(\mathcal{V}) < 1$. Let W be such that $\mathcal{V} = [W]$ and $\mathrm{wg}(W) = \sum_{\sigma \in W} 2^{-|\sigma|} < 1 - \varepsilon$ for some ε (we make W almost prefix-free , that is $\mathrm{wg}(W) \leq \lambda(\mathcal{V}) + \varepsilon'$ for ε' sufficiently small, as allowed by [18], Lemma 3.7.1). We define:

$$\mathcal{V}^n = [W^n] = [\{\sigma_1 \sigma_2 \cdots \sigma_n : \sigma_i \in W\}].$$

We show that $\bigcap \mathcal{V}^n \supseteq \bigcap \mathcal{U}_n$ and that it is a valid test. Let $X \in \bigcap \mathcal{U}_n - \bigcap \mathcal{V}^n$ toward a contradiction. There exists a n such that $X \in \mathcal{V}^n$ and $X \notin \mathcal{V}^{n+1}$ (we must have $X \in \mathcal{V}^1$ by definition of $\mathcal{V}^1$). As $X \in \mathcal{V}^n$, there exists $\sigma \in W^n$ such that $X = \sigma Y$. But as $X \in \bigcap \mathcal{U}_n$, $Y \in \bigcap \widetilde{\mathcal{U}_n} \subseteq \mathcal{V}$, and there exists $\tau \in W$ such that $Y = \tau Z$. But then, $\sigma\tau \in W^{n+1}$ and $X \in \mathcal{V}^{n+1}$, a contradiction.

It remains to prove that $\bigcap \mathcal{V}^n$ is a test which is the case if it is effectively of measure 0. To do so we can easily prove by induction that $\lambda(\mathcal{V}^n) \leq \mathrm{wg}(W)^n$. Indeed, $\lambda(\mathcal{V}^{n+1}) \leq \sum_{\sigma \in W^n} \sum_{\tau \in W} 2^{-|\sigma\tau|} \leq (\sum_{\sigma \in W^n} 2^{-|\sigma|})(\sum_{\tau \in W} 2^{-|\tau|}) = \mathrm{wg}(W)^n$. Then $\lambda(\mathcal{V}^{n+1}) \leq (1 - \varepsilon)^{n+1}$.

We covered every Π^1_1-ML^A test with a test without oracle, so A is low for Π^1_1-MLR, that is, higher K-trivial. ◀

▶ **Theorem 31.** *Suppose A is not higher K-trivial. Then, there is a Π^1_1-ML$\langle\mathcal{O}\rangle$-random which is not* WII$^1_1$R^A.

Proof. Let us denote by $\mathcal{R}_{\mathcal{O}}$ (respectively $\mathcal{R}_{\mathrm{W}}$) the set of Π^1_1-ML$\langle\mathcal{O}\rangle$ (respectively WII$^1_1$R^A) randoms. We are trying to prove that the set $\mathcal{R}_{\mathcal{O}} \cap \overline{\mathcal{R}_{\mathrm{W}}}$ is not empty. We will build an element inside this intersection by forcing. The main thing needed for the construction is to clarify how we will layer these two sets.

First we have $\mathcal{R}_{\mathcal{O}} = \bigcap_m \bigcup_n \mathcal{F}_{m,n}$ where the $\mathcal{F}_{m,n}$ are Σ^1_1 closed sets, increasing over n. Neither the intersection or the union need to be effective. Each union is in fact effective in Kleene's $\mathcal{O}$ (by definition of a Π^1_1-ML$\langle\mathcal{O}\rangle$ test) and each intersection is effective in the double jump of Kleene's $\mathcal{O}$ (to select the functions Turing computable from Kleene's $\mathcal{O}$ which are totals and which pick the right indices for a Π^1_1-ML$\langle\mathcal{O}\rangle$ test). Note that we can also require without loss of generality that each $\mathcal{F}_{m,n}$ contains only Π^1_1-ML randoms: to do so we simply replace each $\mathcal{F}_{m,n}$ by the uniform union of its intersection with each Σ^1_1 closed component in the complement of a universal Π^1_1-ML test.

Then $\overline{\mathcal{R}_{\mathrm{W}}}$ is the union of all the WII$^1_1$R^A tests. In particular, it contains the test $\bigcap_n \mathcal{U}_n$ given by the lemma 30: as A is not higher K-trivial, every $\mathcal{U}_n$ intersects every Σ^1_1 closed set of positive measure. Furthermore if this closed set contains only Π^1_1-ML randoms, this intersection must be of positive measure (it is a fact that no Π^1_1-ML random can be in a Σ^1_1 closed set of measure 0).

In conclusion, it is sufficient to construct a Z such that $Z \in \mathcal{U}_n$ for every n and $Z \in \bigcup_n \mathcal{F}_{m,n}$ for every m. It is now clear how to do so by forcing with a decreasing

sequence of Σ^1_1 closed sets of positive measure: We start with $\mathcal{F}_{0,0}$ which intersects with positive measure some $[\sigma_0] \subseteq \mathcal{U}_0$.

Suppose now by induction, that for some m we have closed sets $\mathcal{F}_{i,n_i}$ for $i \leq m$ and strings $\sigma_1 \prec \cdots \prec \sigma_m$, such that $\lambda(\bigcap_{i\leq m} \mathcal{F}_{i,n_i} \cap [\sigma_m]) > 0$ and such that $[\sigma_m] \subseteq \bigcap_{i\leq m} \mathcal{U}_i$. Let us find n_{m+1} and $\sigma_{m+1} \succ \sigma_m$ with $[\sigma_{m+1}] \subseteq \bigcap_{i\leq m+1} \mathcal{U}_i$ such that $\lambda(\bigcap_{i\leq m+1} \mathcal{F}_{i,n_i} \cap [\sigma_m]) > 0$.

As $\bigcap_{i\leq m} \mathcal{F}_{i,n_i} \cap [\sigma_m]$ is a Σ^1_1 closed set of positive measure, it intersects with positive measure the set $\mathcal{U}_{m+1}$. Thus there exists $\sigma_{m+1} \succ \sigma_m$ with $\sigma_{m+1} \subseteq \bigcap_{i\leq m+1} \mathcal{U}_i$ such that $\lambda(\bigcap_{i\leq m} \mathcal{F}_{i,n_i} \cap [\sigma_{m+1}]) > 0$. Now as $\lambda(\bigcup_n \mathcal{F}_{m+1,n}) = 1$, there is some n_{m+1} such that $\lambda(\bigcap_{i\leq m+1} \mathcal{F}_{i,n_i} \cap [\sigma_{m+1}]) > 0$.

By construction, the unique sequence $Z \in \bigcap_i [\sigma_i]$ is such that $Z \in \bigcap_m \bigcup_n \mathcal{F}_{m,n}$ and $Z \in \bigcap_n \mathcal{U}_n$ which concludes the proof. ◀

References

1 Laurent Bienvenu, Adam R Day, Noam Greenberg, Antonín Kučera, Joseph S Miller, André Nies, and Dan Turetsky. Computing k-trivial sets by incomplete random sets. *The Bulletin of Symbolic Logic*, 20(01):80–90, 2014.

2 Laurent Bienvenu, Noam Greenberg, and Benoit Monin. Continuous higher randomness.

3 Chi Tat Chong, André Nies, and Liang Yu. Lowness of higher randomness notions. *Israel J. Math.*, 166(1):39–60, 2008.

4 Chi Tat Chong and Liang Yu. Randomness in the higher setting. Submitted.

5 Chi Tat Chong and Liang Yu. *Recursion Theory: Computational Aspects of Definability*, volume 8. Walter de Gruyter GmbH & Co KG, 2015.

6 Adam R. Day and Joseph S. Miller. Cupping with random sets. *Proc. Amer. Math. Soc.*, 142(8):2871–2879, 2014. doi:10.1090/S0002-9939-2014-11997-6.

7 Rod Downey, Andre Nies, Rebecca Weber, and Liang Yu. Lowness and Π^0_2 nullsets. *J. Symbolic Logic*, 71(3):1044–1052, 09 2006. doi:10.2178/jsl/1154698590.

8 Rodney G. Downey and Denis R. Hirschfeldt. *Algorithmic Randomness and Complexity*. Theory and Applications of Computability. Springer, 2010. doi:10.1007/978-0-387-68441-3.

9 N. Greenberg, J. Miller, B. Monin, and D. Turetsky. Two more characterizations of k-triviality. *Notre Dame Journal of Formal Logic*, To appear.

10 Noam Greenberg and Benoit Monin. Higher randomness and genericity.

11 Joel David Hamkins and Andy Lewis. Infinite time turing machines. *The Journal of Symbolic Logic*, 65(02):567–604, 2000.

12 Denis Hirschfeldt, André Nies, and Frank Stephan. Using random sets as oracles. *Journal of the London Mathematical Society*, 75(3):610–622, 2007.

13 Greg Hjorth and André Nies. Randomness via effective descriptive set theory. *Journal of the London Mathematical Society*, 75(2):495–508, 2007.

14 Alexander S. Kechris. *Classical Descriptive Set Theory*. Graduate Texts in Mathematics. Springer New York, 2012.

15 Bjørn Kjos-Hanssen, Joseph S Miller, and Reed Solomon. Lowness notions, measure and domination. *Journal of the London Mathematical Society*, page jdr072, 2012.

16 Per Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.

17 Per Martin-Löf. On the notion of randomness. *Studies in Logic and the Foundations of Mathematics*, 60:73–78, 1970. doi:10.1016/S0049-237X(08)70741-9.

18 Benoit Monin. *Higher computability and randomness*. PhD thesis, Universite Paris Diderot, 2014.

19 Yiannis Moschovakis. *Descriptive Set Theory*. Mathematical surveys and monographs. American Mathematical Society, 2009.

20 André Nies. Lowness properties and randomness. *Advances in Mathematics*, 197(1):274–305, 2005.

21 André Nies. *Computability and Randomness*. Oxford Logic Guides. Oxford University Press, 2009.

22 Gerald E. Sacks. *Higher recursion theory*. Perspectives in mathematical logic. Springer-Verlag, 1990.

The Quantum Monad on Relational Structures*

Samson Abramsky[1], Rui Soares Barbosa[2], Nadish de Silva[3], and Octavio Zapata[4]

1 **Department of Computer Science, University of Oxford, UK**
samson.abramsky@cs.ox.ac.uk
2 **Department of Computer Science, University of Oxford, UK**
rui.soares.barbosa@cs.ox.ac.uk
3 **Department of Computer Science, University College London, UK**
nadish.desilva@utoronto.ca
4 **Department of Computer Science, University College London, UK**
ocbzapata@gmail.com

Abstract

Homomorphisms between relational structures play a central role in finite model theory, constraint satisfaction, and database theory. A central theme in quantum computation is to show how quantum resources can be used to gain advantage in information processing tasks. In particular, non-local games have been used to exhibit quantum advantage in boolean constraint satisfaction, and to obtain quantum versions of graph invariants such as the chromatic number. We show how quantum strategies for homomorphism games between relational structures can be viewed as Kleisli morphisms for a quantum monad on the (classical) category of relational structures and homomorphisms. We use these results to exhibit a wide range of examples of contextuality-powered quantum advantage, and to unify several apparently diverse strands of previous work.

1998 ACM Subject Classification F.1.2 Modes of Computation, F.3.2 Semantics of Programming Languages

Keywords and phrases non-local games, quantum computation, monads

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.35

1 Introduction

Finite relational structures and the homomorphisms between them form a mathematical core common to finite model theory [25], constraint satisfaction [14], and relational database theory [24]. Moreover, much of graph theory can be formulated in terms of the existence of graph homomorphisms, as expounded e.g. in the influential text [17]. Thus, implicitly at least, the mathematical setting for all these works is categories of σ-structures and homomorphisms, for relational vocabularies σ.

What could it mean to quantize these structures? More precisely, with the advent of quantum computing, we can now consider the consequences of using quantum resources for carrying out various information-processing tasks. A major theme of current research is to

* This work was carried out in part while the authors visited the Simons Institute for the Theory of Computing (supported by the Simons Foundation) at the University of California, Berkeley, as participants of the Logical Structures in Computation programme. Support from the following is also gratefully acknowledged: EPSRC – Engineering and Physical Sciences Research Council: EP/N018745/1 (SA, RSB) and EP/N017935/1 (NdS), 'Contextuality as a Resource in Quantum Computation'; CONACyT – Consejo Nacional de Ciencia y Tecnología (OZ); and Cambridge Quantum Computing Ltd. (OZ).

delineate the scope of the *quantum advantage* which can be gained by the use of quantum resources. How can this be related to these fundamental structures?

Our starting point is the notion of *quantum graph homomorphism* introduced in [27] as a generalization of the notion of quantum chromatic number [11]. Consider the following game, played by Alice and Bob cooperating against a Verifier. Their goal is to establish the existence of a homomorphism $G \to H$ for given graphs G and H. Verifier provides vertices $v_1, v_2 \in V(G)$ to Alice and Bob respectively. They produce outputs $w_1, w_2 \in V(H)$ in response. No communication between Alice and Bob is permitted during the game. They win if the following conditions hold: $v_1 = v_2 \;\Rightarrow\; w_1 = w_2$ and $v_1 \sim v_2 \;\Rightarrow\; w_1 \sim w_2$, where we write $\sim$ for the adjacency relation.

If only classical resources are permitted, then the existence of a *perfect strategy* for Alice and Bob — one in which they win with probability 1 — is equivalent to the existence of a graph homomorphism in the standard sense. However, using quantum resources, in the form of an entangled bipartite state where Alice and Bob can each perform measurements on their part, there are perfect strategies in cases where no classical homomorphism exists, thus exhibiting quantum advantage.

Alice–Bob games have also been studied for other tasks, notably for *constraint systems*. Consider the following system of linear equations over $\mathbb{Z}_2$:

$$\begin{array}{lll} A \oplus B \oplus C = 0 & D \oplus E \oplus F = 0 & G \oplus H \oplus I = 0 \\ A \oplus D \oplus G = 0 & B \oplus E \oplus H = 0 & C \oplus F \oplus I = 1 \end{array}$$

Of course, this system is not satisfiable in the standard sense, as we can see by summing over the left- and right-hand sides. Now consider the following Alice–Bob game. The Verifier sends Alice an equation, and Bob a variable. Alice returns an assignment to the variables in the equation, and Bob returns an assignment for his variable. They win if Bob's assignment agrees with Alice's, and moreover Alice's assignment satisfies the given equation. Classically, the existence of a perfect strategy is equivalent to the existence of a satisfying assignment for the whole system. Using quantum resources, there is a perfect strategy for the above system, which corresponds to Mermin's "magic square" construction [29]. This can be generalized to a notion of quantum perfect strategies for a broad class of constraint systems [13, 12], which have strong connections both to the study of contextuality in quantum mechanics, and to a number of challenging mathematical questions [36, 35]. Clearly, these games are analogous to those for graph homomorphisms. What is the precise relationship?

In [27], generalizing results in [11], the existence of a quantum perfect strategy for the homomorphism game from G to H is characterized in terms of the existence of a family $\{E_{vw}\}_{v \in V(G), w \in V(H)}$ of projectors in d-dimensional Hilbert space for some d, subject to certain conditions. Analogous results for constraint systems are proved in [13]. This characterization eliminates the two-person aspect of the game, and the shared state, leaving a "projector-valued relation" as the witness for existence of a quantum perfect strategy. We shall henceforth call these witnesses *quantum graph homomorphisms*. An important further step is taken in [27]. A construction $H \mapsto \mathsf{M}H$ on graphs is introduced, such that the existence of a quantum graph homomorphism from G to H is equivalent to the existence of a *standard* graph homomorphism $G \to \mathsf{M}H$.

Our contribution begins at this point. We describe a general notion of non-local game for witnessing homomorphisms between structures for any relational signature. We show that the use of quantum resources in these games can be characterized by a notion of *quantum homomorphism*. Moreover, quantum homomorphisms can in turn be characterized as the

Kleisli morphisms for a *quantum monad* on the (classical) category of relational structures and homomorphisms. This monad is *graded* [30] by the dimension of the Hilbert space.

Our account refines and generalizes the ideas from both [11, 27] and [13]. We characterize quantum solutions for general constraint satisfaction problems, showing as a special case that these subsume the binary constraint systems of [13]. We also show how quantum witnesses for strong contextuality in the sense of [4] are characterized by quantum homomorphisms.

The precise relationship with the quantum graph homomorphisms of [27] turns out to be more subtle. We show that their notion is characterized by a quantum solution in our sense for a related boolean constraint system. Overall, we show that a wide range of notions of quantum advantage is captured in a uniform way by the quantum monad, applied directly to the standard classical structures.

For reasons of limited space, some background material on linear algebra and quantum mechanics (e.g. the notions of POVM and PVM) and some proofs have been relegated to an Appendix.

2 From quantum perfect strategies to quantum homomorphisms

We write $[p] := \{1, \ldots, p\}$. We fix a finite relational vocabulary $\sigma = \{R_1, \ldots, R_p\}$, where R_a has arity k_a, $a \in [p]$. A σ-structure has the form $\mathcal{A} = (A, R_1^{\mathcal{A}}, \ldots, R_p^{\mathcal{A}})$, where A is a non-empty set, and $R_a^{\mathcal{A}} \subseteq A^{k_a}$, $a \in [p]$. A homomorphism of σ-structures $f : \mathcal{A} \to \mathcal{B}$ is a function $f : A \to B$ such that, for all $a \in [p]$ and $\mathbf{x} \in A^{k_a}$, $\mathbf{x} \in R_a^{\mathcal{A}} \;\Rightarrow\; f(\mathbf{x}) \in R_a^{\mathcal{B}}$. Here we use vector notation: $\mathbf{x} = (x_1, \ldots, x_{k_a})$ and $f(\mathbf{x}) = (f(x_1), \ldots, f(x_{k_a}))$. We denote the category of σ-structures and homomorphisms by $\mathcal{R}(\sigma)$, and the full subcategory of finite structures by $\mathcal{R}_f(\sigma)$.

We now consider the following game, played on finite structures $\mathcal{A}$ and $\mathcal{B}$, in which Alice and Bob cooperate to convince a Verifier that there is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$:

- Alice and Bob are separated, and not allowed to communicate (exchange classical information) while the game is played.
- In a play of the game, the Verifier sends Alice an index a, and a tuple $\mathbf{x} \in R_a^{\mathcal{A}}$; and Bob an element $x \in A$.
- Alice returns a tuple $\mathbf{y} \in B^{k_a}$, and Bob returns an element $y \in B$.
- Alice and Bob win that play if
  - **(i)** $\mathbf{y} \in R_a^{\mathcal{B}}$
  - **(ii)** $x = \mathbf{x}_i \;\Rightarrow\; y = \mathbf{y}_i$, $i \in [k_a]$.

Alice and Bob may use probabilistic strategies. A *perfect strategy* is one in which they win with probability 1.

It is clear that if only classical resources are allowed, the existence of a perfect strategy is equivalent to the existence of a homomorphism from $\mathcal{A}$ to $\mathcal{B}$. The actual strategy played by Alice and Bob may be pure or mixed, in the latter case using some shared randomness.

We now consider the use of quantum resources in the homomorphism game. We shall only consider the case of finite-dimensional resources in this paper. Such resources have the following general form:

- There are finite-dimensional Hilbert spaces H and K, and a pure state ψ on $\mathsf{H} \otimes \mathsf{K}$. This state is shared between Alice and Bob. The separation between Alice and Bob is reflected in the fact that Alice can only perform operations on H, while Bob can only perform operations on K.
- For each $a \in [p]$ and tuple $\mathbf{x} \in R_a^{\mathcal{A}}$, Alice has a POVM $\mathcal{E}_{\mathbf{x}}^a = \{\mathcal{E}_{\mathbf{x},\mathbf{y}}^a\}_{\mathbf{y} \in B^{k_a}}$.
- For each $x \in A$, Bob has a POVM $\mathcal{F}_x = \{\mathcal{F}_{x,y}\}_{y \in B}$

These resources are used as follows:

- Given a and $\mathbf{x}$, Alice measures $\mathcal{E}^a_{\mathbf{x}}$ on her part of ψ.
- Given x, Bob measures $\mathcal{F}_x$ on his part of ψ.
- They obtain the joint outcome $(\mathbf{y}, y)$ with probability $\psi^*(\mathcal{E}^a_{\mathbf{x},\mathbf{y}} \otimes \mathcal{F}_{x,y})\psi$.

If with probability 1 the outcome $(\mathbf{y}, y)$ satisfies the winning conditions, then this is a quantum perfect strategy.

We can write the winning conditions explicitly in terms of the quantum operations:

$$\text{(QS1)} \quad \psi^*(\mathcal{E}^a_{\mathbf{x},\mathbf{y}} \otimes \mathcal{F}_{x,y})\psi = 0 \qquad \text{if } x = \mathbf{x}_i \text{ and } y \neq \mathbf{y}_i$$

$$\text{(QS2)} \quad \psi^*(\mathcal{E}^a_{\mathbf{x},\mathbf{y}} \otimes I)\psi = 0 \qquad \text{if } \mathbf{y} \notin R^{\mathcal{B}}_a.$$

A first remark is that our assumption of the bipartite structure of the state space does not in fact lose any generality. We could have asked simply that Bob's operators commute with those of Alice. However, since we are considering the finite-dimensional case, a result of Tsirelson [37, 35] implies that this is equivalent to the tensor product formulation we have used. Furthermore, using a pure state also does not lose any generality. Indeed, if we had a mixed state $\rho = \sum_i p_i \psi_i \psi_i^*$, with the trace replacing the inner products in (QS1) and (QS2), then the linearity of the trace implies that we could just as well have used any of the pure states ψ_i with the same measurements, and still satisfy the conditions.

We shall now show that in fact a quantum perfect strategy can without loss of generality be assumed to have a very special form, which will lead us to the equivalence with quantum homomorphisms. These results combine ingredients from [13] and [27, 11]. The proofs are closest to those in [34], but are considerably simpler as well as more general.

For notational convenience, we shall focus on the case where the relational signature has a single k-ary relation R. Thus a quantum perfect strategy for the homomorphism game from $\mathcal{A}$ to $\mathcal{B}$ has the form $(\psi, \{\mathcal{E}_{\mathbf{x}}\}_{\mathbf{x}\in R^{\mathcal{A}}}, \{\mathcal{F}_x\}_{x\in A})$, where $\mathcal{E}_{\mathbf{x}} = \{\mathcal{E}_{\mathbf{x},\mathbf{y}}\}_{\mathbf{y}\in B^k}$ and $\mathcal{F}_x = \{\mathcal{F}_{x,y}\}_{y\in B}$ are POVMs satisfying the conditions (QS1) and (QS2).

Our first step is to show that ψ can be taken to have full Schmidt rank.

▶ **Lemma 1.** *Given a quantum perfect strategy $(\psi', \{\mathcal{E}'_{\mathbf{x}}\}_{\mathbf{x}\in R^{\mathcal{A}}}, \{\mathcal{F}'_x\}_{x\in A})$, we can find a strategy $(\psi, \{\mathcal{E}_{\mathbf{x}}\}_{\mathbf{x}\in R^{\mathcal{A}}}, \{\mathcal{F}_x\}_{x\in A})$ where $\psi \in \mathbb{C}^d \otimes \mathbb{C}^d$ has the form $\sum_{i=1}^d \lambda_i e_i \otimes e_i$ with $\lambda_i > 0$ for all i.*

Proof. Our proof will follow closely the first part of the proof of Theorem 6.5.1 in [34], so we omit detailed calculations. We write the Schmidt decomposition of ψ' as $\sum_{i=1}^d \lambda_i \alpha_i \otimes \beta_i \in \mathbb{C}^{d_A} \otimes \mathbb{C}^{d_B}$, where $\{\alpha_i\}$, $\{\beta_i\}$ are orthonormal families of vectors, and $\lambda_i > 0$.

We define $P_A := \sum_{i=1}^d e_i \alpha_i^*$, $P_B := \sum_{i=1}^d e_i \beta_i^*$. Thus $P_A : \mathbb{C}^{d_A} \to \mathbb{C}^d$, and $P_B : \mathbb{C}^{d_B} \to \mathbb{C}^d$. It is straightforward to verify that $P_A P_A^* = I_d = P_B P_B^*$. We have $\psi := (P_A \otimes P_B)\psi'$, and $\psi' = (P_A^* \otimes P_B^*)\psi$.

Similarly, we define $\mathcal{E}_{\mathbf{x},\mathbf{y}} := P_A \mathcal{E}'_{\mathbf{x},\mathbf{y}} P_A^*$, and $\mathcal{F}_{x,y} := P_A \mathcal{F}'_{x,y} P_B^*$. Again, it is straightforward to verify that this yields well-defined POVMs, and moreover that the probabilities are preserved: $\psi^*(\mathcal{E}_{\mathbf{x},\mathbf{y}} \otimes \mathcal{F}_{x,y})\psi = \psi'^*(\mathcal{E}'_{\mathbf{x},\mathbf{y}} \otimes \mathcal{F}'_{x,y})\psi'$. Thus $(\psi, \{\mathcal{E}_{\mathbf{x}}\}_{\mathbf{x}\in R^{\mathcal{A}}}, \{\mathcal{F}_x\}_{x\in A})$ is a quantum perfect strategy. ◀

The following simple general result will be useful.

▶ **Lemma 2.** *Let $\mathcal{A}$ be a $*$-algebra, and a, b, d be self-adjoint elements of $\mathcal{A}$, where d is also invertible. Suppose that $ad = adb = db$. Then a and b are projectors, and they both commute with d^2. If $\mathcal{A}$ is a C^*-algebra, then $a = b$.*

Proof. First, $a^2d = adb = ad$. Since d is invertible, this implies that $a^2 = a$, so a is a projector. Similarly, $b^2 = b$. Moreover, since a, b and d are self-adjoint, $db = ad \iff bd = da$. Hence $ad^2 = dbd = d^2a$.

If $\mathcal{A}$ is a C*-algebra, then it is standard that d commutes with every element which commutes with d^2 [9, 32], so $da = ad = db$, and since d is invertible, this yields $a = b$. ◀

We shall now show that under the assumption of full Schmidt rank, the measurements are already remarkably constrained. We define $\mathcal{E}^i_{\mathbf{x},y} := \sum_{\mathbf{y}_i=y} \mathcal{E}_{\mathbf{x},\mathbf{y}}$.

▶ **Lemma 3.** *Let $(\psi, \{\mathcal{E}_\mathbf{x}\}, \{\mathcal{F}_x\})$ be a quantum perfect strategy in which ψ has full Schmidt rank. Then for all $\mathbf{x}$, i, y, $\mathcal{E}^i_{\mathbf{x},y}$ and $\mathcal{F}_{x,y}$ are projectors, and $\mathcal{E}^i_{\mathbf{x},y} = \mathcal{F}^T_{x,y}$ whenever $x = \mathbf{x}_i$.*

Proof. We write ψ as $\sum_{i=1}^d \lambda_i e_i \otimes e_i$, where $\lambda_i > 0$. The corresponding $d \times d$ diagonal matrix $D = \mathsf{diag}\{\lambda_i\}$ is full rank, and hence invertible, and $D^* = D$. Note that $\psi = \mathsf{vec}(D)$, the vectorization of D. Using the standard equations $(A \otimes B)\mathsf{vec}(D) = \mathsf{vec}(ADB^T)$ and $\mathsf{vec}(A)^*\mathsf{vec}(B) = \mathsf{Tr}(AB)$, we have

$$\psi^*(\mathcal{E}_{\mathbf{x},\mathbf{y}} \otimes \mathcal{F}_{x,y})\psi = 0 \iff \mathsf{Tr}(D\mathcal{E}_{\mathbf{x},\mathbf{y}}D\mathcal{F}^T_{x,y}) = 0 \iff \mathsf{Tr}(\mathcal{E}_{\mathbf{x},\mathbf{y}}D\mathcal{F}^T_{x,y}D) = 0.$$

By Proposition 17, $\mathsf{Tr}(\mathcal{E}_{\mathbf{x},\mathbf{y}}D\mathcal{F}^T_{x,y}D) = 0 \iff \mathcal{E}_{\mathbf{x},\mathbf{y}}D\mathcal{F}^T_{x,y}D = \mathbf{0}$, and since D is invertible, this is equivalent to $\mathcal{E}_{\mathbf{x},\mathbf{y}}D\mathcal{F}^T_{x,y} = \mathbf{0}$. By condition (QS1), $\mathcal{E}_{\mathbf{x},\mathbf{y}}D\mathcal{F}^T_{x,y} = \mathbf{0}$ when $x = \mathbf{x}_i$ and $y \neq \mathbf{y}_i$. This in turn implies that $\mathcal{E}^i_{\mathbf{x},y}D\mathcal{F}^T_{x,y'} = \mathbf{0}$ when $x = \mathbf{x}_i$ and $y \neq y'$.

Now fix $\mathbf{x}$ and $x = \mathbf{x}_i$. Let $A_y := \mathcal{E}^i_{\mathbf{x},y}$, and $B_y := \mathcal{F}^T_{x,y}$. We have $\sum_y A_y = I = \sum_y B_y$, and $A_yDB_{y'} = \mathbf{0}$ when $y \neq y'$. Hence $A_yD = \sum_{y'} A_yDB_{y'} = A_yDB_y = \sum_{y'} A_{y'}DB_y = DB_y$. We can now apply Lemma 2, taking $a = A_y$, $d = D$, and $b = B_y$, to conclude that $\mathcal{E}^i_{\mathbf{x},y}$ and $\mathcal{F}_{x,y}$ are projectors, and moreover that they commute with D^2. We can use the last part of Lemma 2 to conclude that $\mathcal{E}^i_{\mathbf{x},y} = \mathcal{F}^T_{x,y}$ whenever $x = \mathbf{x}_i$. ◀

Finally, we show that the state can be chosen to be maximally entangled.

▶ **Lemma 4.** *Let $(\psi', \{\mathcal{E}_\mathbf{x}\}, \{\mathcal{F}_x\})$ be a quantum perfect strategy where $\psi' = \sum_{i=1}^d \lambda_i e_i \otimes e_i$ with $\lambda_i > 0$ for all i. Then $(\psi, \{\mathcal{E}_\mathbf{x}\}, \{\mathcal{F}_x\})$ is a quantum perfect strategy, where $\psi = 1/\sqrt{d}\sum_{i=1}^d e_i \otimes e_i$ is the maximally entangled state.*

Proof. Let D' be the diagonal matrix associated with ψ'. Using Lemma 3,

$$\psi'^*(\mathcal{E}_{\mathbf{x},\mathbf{y}} \otimes \mathcal{F}_{x,y})\psi' = 0 \iff \mathcal{E}_{\mathbf{x},\mathbf{y}}D'\mathcal{F}^T_{x,y} = \mathbf{0} \iff \mathcal{E}_{\mathbf{x},\mathbf{y}}\mathcal{F}^T_{x,y}D' = \mathbf{0} \iff \mathcal{E}_{\mathbf{x},\mathbf{y}}\mathcal{F}^T_{x,y} = \mathbf{0},$$

since D' commutes with $\mathcal{F}^T_{x,y}$ and is invertible. Similarly,

$$\psi'^*(\mathcal{E}_{\mathbf{x},\mathbf{y}} \otimes I)\psi' = 0 \iff \mathcal{E}_{\mathbf{x},\mathbf{y}}D' = \mathbf{0} \iff \mathcal{E}_{\mathbf{x},\mathbf{y}} = \mathbf{0}.$$

These conditions will be preserved by any state ψ whose diagonal matrix is full rank and commutes with the matrices $\mathcal{F}_{x,y}$. This holds in particular for the maximally entangled state, whose diagonal matrix has the form $\frac{1}{\sqrt{d}}I_d$. ◀

We can now combine Lemmas 1, 3 and 4 to obtain the following result:

▶ **Theorem 5.** *The existence of a quantum perfect strategy implies the existence of a strategy $(\psi, \{\mathcal{E}_\mathbf{x}\}, \{\mathcal{F}_x\})$ with the following properties:*

- *The POVMs $\mathcal{E}^i_\mathbf{x}$ and $\mathcal{F}_x$ are projective.*
- *The state ψ is a maximally entangled state $\psi = 1/\sqrt{d}\sum_{i=1}^d e_i \otimes e_i$.*
- *If $x = \mathbf{x}_i$ then $\mathcal{E}^i_{\mathbf{x},y} = \mathcal{F}^T_{x,y}$.*
- *If $\mathbf{x} \in R^\mathcal{A}$ and $\mathbf{y} \notin R^\mathcal{B}$, then $\mathcal{E}_{\mathbf{x},\mathbf{y}} = \mathbf{0}$.*

It is worth noting that the procedure for obtaining the strategy in this special form has three steps:

- In Step 1, the state and strategies are projected down to the subspace corresponding to the support of the Schmidt decomposition of the state. This step reduces the dimension of the Hilbert space, and preserves the probabilities for the strategy exactly.
- Step 2 does not change the strategy at all, but shows that it must already have strong properties.
- Step 3 changes the state but not the measurements. In general, the probabilities for the strategy will be changed, but the possibilities are preserved exactly.

Thus in passing to the special form, the dimension is reduced; the process by which we obtain projective measurements is not at all akin to dilation.

This theorem shows that all the information determining the strategy is in Alice's operators. Moreover, Alice's operators must be chosen non-contextually, so that $\mathcal{E}^i_{\mathbf{x},y}$ is independent of the context $\mathbf{x}$. This means that we can define projectors $P_{x,y} := \mathcal{E}^i_{\mathbf{x},y}$ whenever $x = \mathbf{x}_i$. If $\mathbf{x}_i = x = \mathbf{x}'_j$, then we have $\mathcal{E}^i_{\mathbf{x},y} = \mathcal{F}^T_{x,y} = \mathcal{E}^j_{\mathbf{x}',y}$, so $P_{x,y}$ is well-defined.

Now, recall the notion of joint measurability: a family of POVMs $\{A^1_y\}_{y\in Y_1}, \ldots, \{A^k_y\}_{y\in Y_k}$ is said to be jointly measurable if there is a POVM $\{B_{\mathbf{y}}\}_{\mathbf{y}\in Y_1\times\cdots\times Y_k}$ such that for all i, $A^i_y = \sum_{\mathbf{y}_i=y} B_{\mathbf{y}}$. The following result is standard [16].

▶ Proposition 6. A family of projective measurements is jointly measurable by a POVM if and only if they pairwise commute, and in this case the POVM is the product of the family, and hence projective.

For each $\mathbf{x} \in R^{\mathcal{A}}$, the projective measurements $P_{\mathbf{x}_i} = \{P_{\mathbf{x}_i,y}\}$ defined above are jointly measured by $\mathcal{E}_{\mathbf{x}}$. Thus $\mathcal{E}_{\mathbf{x}}$ is the projective measurement $P_{\mathbf{x}} = \{P_{\mathbf{x},\mathbf{y}}\}_{\mathbf{y}}$ given by $P_{\mathbf{x},\mathbf{y}} := P_{\mathbf{x}_1,\mathbf{y}_1} \cdots P_{\mathbf{x}_k,\mathbf{y}_k}$.

We shall now introduce the notion of *quantum homomorphism* between relational structures $\mathcal{A}$ and $\mathcal{B}$. A quantum homomorphism is a family of projectors $\{P_{x,y}\}_{x\in A,y\in B}$ in $\mathsf{Proj}(d)$ for some d, satisfying the following conditions:

- (QH1) For all $x \in A$, $\sum_{y\in B} P_{x,y} = I$.
- (QH2) For all $\mathbf{x} \in R^{\mathcal{A}}$, $x = \mathbf{x}_i$, $x' = \mathbf{x}_j$, and $y, y' \in B$, $[P_{x,y}, P_{x',y'}] = \mathbf{0}$. Thus we can define a projective measurement $P_{\mathbf{x}} = \{P_{\mathbf{x},\mathbf{y}}\}_{\mathbf{y}}$, where $P_{\mathbf{x},\mathbf{y}} := P_{\mathbf{x}_1,\mathbf{y}_1} \cdots P_{\mathbf{x}_k,\mathbf{y}_k}$.
- (QH3) If $\mathbf{x} \in R^{\mathcal{A}}$ and $\mathbf{y} \notin R^{\mathcal{B}}$, then $P_{\mathbf{x},\mathbf{y}} = \mathbf{0}$.

Note that (QH1) implies that for any x, $P_{x,y}P_{x,y'} = \mathbf{0}$ whenever $y \neq y'$.

We write $\mathcal{A} \xrightarrow{q} \mathcal{B}$ for the existence of a quantum homomorphism from $\mathcal{A}$ to $\mathcal{B}$.

▶ **Theorem 7.** *For finite structures $\mathcal{A}$, $\mathcal{B}$, the following are equivalent:*

1. *There is a quantum perfect strategy for the homomorphism game from $\mathcal{A}$ to $\mathcal{B}$.*
2. *$\mathcal{A} \xrightarrow{q} \mathcal{B}$.*

Proof. The implication from (1) to (2) follows directly from Theorem 5 and the subsequent discussion. For the converse, given a quantum homomorphism $\{P_{x,y}\}_{x\in A,y\in B}$, we can define $\mathcal{E}_{\mathbf{x},\mathbf{y}} := P_{\mathbf{x},\mathbf{y}}$, $\mathcal{F}_{x,y} := P^T_{x,y}$, and use the maximally entangled state to obtain a quantum perfect strategy. It is straightforward to verify that the homomorphism conditions (QH1)–(QH3) imply the strategy conditions (QS1) and (QS2). ◀

As a final remark, although we have focussed on a single relation to simplify the notation, our results go through for arbitrary relational signatures. Note that the general form of condition (QH2) is that $P_{x,y}$ and $P_{x',y'}$ must commute whenever x and x' are adjacent in the Gaifman graph of $\mathcal{A}$ – that is, they both occur in some tuple of some relation.

3 From quantum homomorphisms to the quantum monad

We now show how to characterize quantum homomorphisms as the Kleisli morphisms of a monad $\mathcal{Q}_d$ on the category of relational structures. This monad is *graded* [30] by the dimension of the Hilbert space used as the quantum resource.

The monad will be defined on the category $\mathcal{R}(\sigma)$ of all σ-structures, since $\mathcal{Q}_d\mathcal{A}$ will always be infinite, even if $\mathcal{A}$ is finite. For the underlying universes of the structures, the construction can be seen as a quantum variant of the discrete distribution monad [20], widely used in coalgebra and semantics. It is well known that the distribution monad can be defined over any commutative semiring, with the non-negative reals being used for the standard case of probabilities [20, 4]. Here we shall use the projectors $\mathsf{Proj}(d)$ with d ranging over the positive integers. For each d, $\mathsf{Proj}(d)$ is a partial commutative semiring, since we can only add projectors if they are orthogonal, and only multiply them if they commute. We also have the graded multiplication given by the tensor product: if $P \in \mathsf{Proj}(d)$ and $Q \in \mathsf{Proj}(d')$, then $P \otimes Q \in \mathsf{Proj}(dd')$.

We fix a relational signature σ. For each positive integer d and σ-structure $\mathcal{A}$, we define a σ-structure $\mathcal{Q}_d\mathcal{A}$. The universe of this structure $\mathcal{Q}_dA$ is the set of all functions $p : A \to \mathsf{Proj}(d)$ satisfying the normalization condition: $\sum_{x\in A} p(x) = I$. Note that normalization implies that the projectors $\{p(x)\}_{x\in A}$ are pairwise orthogonal. Since we are in finite dimension d, this in turn implies that p has finite support: $p(x) = \mathbf{0}$ for all but finitely many x. We can think of $\mathcal{Q}_dA$ as the projector-valued distributions on A in dimension d. For each relation R of arity k in σ, we define $R^{\mathcal{Q}_d\mathcal{A}}$ to be the set of all tuples $(p_1, \ldots, p_k)$ such that:

- (QR1) For all $i, j \in [k]$, $x, x' \in A$: $[p_i(x), p_j(x')] = \mathbf{0}$.
- (QR2) For all $\mathbf{x} \in A^k$, if $\mathbf{x} \notin R^{\mathcal{A}}$, then $\mathbf{p}(\mathbf{x}) = \mathbf{0}$, where $\mathbf{p}(\mathbf{x}) := p_1(x_1) \cdots p_k(x_k)$.

Note that the first condition implies that the product of projectors in the second is a well-defined projector.

▶ Proposition 8. Let $\mathcal{A}$ and $\mathcal{B}$ be finite σ-structures. There is a bijective correspondence between quantum homomorphisms $\{P_{x,y}\}_{x\in A,y\in B}$ from $\mathcal{A}$ to $\mathcal{B}$ in dimension d and standard homomorphisms $h : \mathcal{A} \to \mathcal{Q}_d\mathcal{B}$.

Proof. Given a quantum homomorphism $\{P_{x,y}\}$, define $h : \mathcal{A} \to \mathcal{Q}_d\mathcal{B}$ by $h(x) = p$, where $p(y) := P_{x,y}$. (QH1) implies normalization. Given $\mathbf{x} \in R^{\mathcal{A}}$, we have to show that $\mathbf{p} = h(\mathbf{x}) \in R^{\mathcal{Q}_d\mathcal{A}}$. (QH2) implies that (QR1) is satisfied for $p_i(y)$, $p_j(y')$, where $p_i = h(x_i)$, $p_j = h(x_j)$. Similarly, (QH3) implies (QR2).

For the converse, given $h : \mathcal{A} \to \mathcal{Q}_dB$, define $P_{x,y} := h(x)(y)$. Again, normalization implies (QH1), (QR1) implies (QH2), and (QR2) implies (QH3). ◀

This correspondence is analogous to the familiar one between relations and set-valued functions, which shows that the category of relations is the Kleisli category of the powerset monad on **Set**.

Now we show that $\mathcal{Q}_d$ extends to a functor on $\mathcal{R}(\sigma)$. Given a homomorphism $h : \mathcal{A} \to \mathcal{B}$, we define $\mathcal{Q}_dh : \mathcal{Q}_d\mathcal{A} \to \mathcal{Q}_d\mathcal{B}$ by $\mathcal{Q}_dh(p)(y) := \sum_{h(x)=y} p(x)$.

▶ Proposition 9. $\mathcal{Q}_dh$ is a well-defined homomorphism. Moreover, $\mathcal{Q}_d$ is functorial: $\mathcal{Q}_dg \circ \mathcal{Q}_dh = \mathcal{Q}_d(g \circ h)$ and $\mathcal{Q}_d\mathsf{id}_A = \mathsf{id}_{\mathcal{Q}_dA}$.

Proof. The finite support and normalization conditions ensure that $\mathcal{Q}_dh$ is well-defined. Functoriality is proved exactly as for the distribution monad. We verify that $\mathcal{Q}_dh$ is a homomorphism. Suppose that $(p_1, \ldots, p_k) \in R^{\mathcal{Q}_d\mathcal{A}}$. By (QR1), this implies that $[p_i(x), p_j(x')] = \mathbf{0}$

for all $i, j \in [k]$, $x, x' \in A$. This implies that

$$[\mathcal{Q}_d h(p_i)(y), \mathcal{Q}_d h(p_j)(y')] \;=\; [\sum_{h(x)=y} p_i(x), \sum_{h(x')=y'} p_j(x')] \;=\; \mathbf{0}$$

so $\mathcal{Q}_d h(\mathbf{p})$ satisfies (QR1). For (QR2), if $\mathbf{y} \notin R^{\mathcal{B}}$,

$$\mathcal{Q}_d h(\mathbf{p})(\mathbf{y}) \;=\; \prod_{j=1}^{k} \sum_{h(\mathbf{x}_j)\,=\,\mathbf{y}_j} p_j(\mathbf{x}_j) \;=\; \sum_{h(\mathbf{x})\,=\,\mathbf{y}} \mathbf{p}(\mathbf{x}) \;=\; \mathbf{0},$$

by (QR2) for $\mathbf{p} \in R^{\mathcal{Q}_d\mathcal{A}}$, since $\mathbf{y} \notin R^{\mathcal{B}}$ and $h(\mathbf{x}) = \mathbf{y}$ implies $\mathbf{x} \notin R^{\mathcal{A}}$. Thus $\mathcal{Q}_d h(\mathbf{p}) \in R^{\mathcal{Q}_d\mathcal{B}}$. ◀

The unit of the monad $\eta_{\mathcal{A}} : \mathcal{A} \to \mathcal{Q}_1\mathcal{A}$ sends $x \in A$ to the "delta distribution" $\delta_x \in \mathcal{Q}_1 A$, where $\delta_x(x) = I_1$, $\delta_x(x') = \mathbf{0}$ if $x \neq x'$. Verification that this is well-defined and yields a natural transformation is straightforward.

We also have the graded monad multiplication: $\mu_{\mathcal{A}}^{d,d'} : \mathcal{Q}_d\mathcal{Q}_{d'}\mathcal{A} \to \mathcal{Q}_{dd'}\mathcal{A}$. This is defined as follows: $\mu_{\mathcal{A}}^{d,d'}(P)(x) := \sum_{p \in \mathcal{Q}_{d'}\mathcal{A}} P(p) \otimes p(x)$. We prove that this gives a well-defined natural transformation in the Appendix. Thinking of $\otimes$ as the graded semiring multiplication on projectors, we can see the correspondence to the distribution monad.

We recall that given a category $\mathcal{C}$, the endofunctor category $[\mathcal{C}, \mathcal{C}]$ is monoidal; a monad on $\mathcal{C}$ is a monoid in this category [26]. Now let $(M, \cdot, 1)$ be a monoid, which we can view as a discrete category with a strict monoidal structure. An *M-graded monad* [30] on $\mathcal{C}$ is a lax monoidal functor from M into $[\mathcal{C}, \mathcal{C}]$. Such a functor is given by the following data: an assignment $m \mapsto T_m$ of an endofunctor on $\mathcal{C}$ to each element of M; a natural transformation $\eta : \mathsf{Id} \xrightarrow{\cdot} T_1$ (the graded unit); and a natural transformation $\mu^{m,m'} : T_m T_{m'} \xrightarrow{\cdot} T_{m \cdot m'}$ for all m, m' (the graded multiplication). These are subject to coherence conditions, which generalize the usual monad equations. We refer to [30] for details.

In our case, we use the monoid $\mathbb{N}^+$ of positive integers under multiplication.

▶ **Theorem 10.** *The triple $(\{\mathcal{Q}_d\}_d, \eta, \{\mu^{d,d'}\}_{d,d'})$ is a $\mathbb{N}^+$-graded monad on $\mathcal{R}(\sigma)$.*

The proof of this result involves verifying a number of equations, and is fairly lengthy but straightforward. We provide details in the Appendix.

We are particularly interested in the Kleisli category for this graded monad. The objects of this category are the same as those of $\mathcal{R}(\sigma)$. A morphism from $\mathcal{A}$ to $\mathcal{B}$ is a homomorphism $h : \mathcal{A} \to \mathcal{Q}_d\mathcal{B}$. By Proposition 8, we know that Kleisli morphisms correspond exactly to quantum homomorphisms.

The graded composition of Kleisli arrows $h : \mathcal{A} \to \mathcal{Q}_d\mathcal{B}$ and $k : \mathcal{B} \to \mathcal{Q}_{d'}\mathcal{C}$ is the arrow $k \bullet h : \mathcal{A} \to \mathcal{Q}_{dd'}\mathcal{C}$ given by $k \bullet h := \mu_{\mathcal{B}}^{d,d'} \circ \mathcal{Q}_d k \circ h$. An explicit description can be calculated from the graded monad structure given above: $(k \bullet h)(x)(z) \;=\; \sum_{y \in B} h(x)(y) \otimes k(y)(z)$. If we write this in terms of the corresponding quantum homomorphisms $\{P_{x,y}\}_{x \in A, y \in B}$, $\{Q_{y,z}\}_{y \in B, z \in C}$, we obtain $\{R_{x,z}\}_{x \in A, z \in C}$ given by the formula $R_{x,z} = \sum_{y \in B} P_{x,y} \otimes Q_{y,z}$. This recovers the concrete definition given for quantum graph homomorphisms in [27].

4 Quantum advantage via the quantum monad

We shall now show how the quantum monad provides a unified framework for expressing quantum advantage in a wide range of information processing tasks. We shall show equivalences between:

- state-independent strong contextuality arguments
- quantum advantage in constraint satisfaction
- existence of quantum (but not classical) homomorphisms between relational structures.

4.1 Classical correspondences

We begin with the standard classical correspondence between constraint satisfaction problems and the existence of homomorphisms. A CSP instance has the form $\mathcal{K} = (V, D, C)$ where V is a set of variables, D is a domain of values[1], and C is a set of constraints of the form $(\mathbf{x}, r)$, where for some k, $\mathbf{x} \in V^k$, and $r \subseteq D^k$. We say that $c = (\mathbf{x}, r)$ is a k-ary constraint. A *solution* of the CSP is a function $s : V \to D$ such that, for all $(\mathbf{x}, r) \in C$, $s(\mathbf{x}) \in r$, where $s(\mathbf{x}) := (s(x_1), \ldots, s(x_k))$.

Given $\mathcal{K} = (V, D, C)$ we define two structures over the signature with a k-ary relation symbol R_c for each k-ary constraint c. First, $\mathcal{B}_\mathcal{K}$ has as universe D, and for each $c = (\mathbf{x}, r) \in C$, $R_c^{\mathcal{B}_\mathcal{K}} = r$. Secondly, $\mathcal{A}_\mathcal{K}$ has universe V, and for each $c = (\mathbf{x}, r) \in C$, $R_c^{\mathcal{A}_\mathcal{K}} = \{\mathbf{x}\}$. The following is immediate:

▶ Proposition 11. There is a one-to-one correspondence between solutions for $\mathcal{K}$ and homomorphisms $\mathcal{A}_\mathcal{K} \to \mathcal{B}_\mathcal{K}$.

There is also a converse to this result. Given σ-structures $\mathcal{A}$ and $\mathcal{B}$, we can define the CSP $\mathcal{K}_{AB} = (V, D, C)$, where $V = A$, $D = B$, and $C = \{(\mathbf{a}, R^\mathcal{B}) \mid R \in \sigma,\ \mathbf{a} \in R^\mathcal{A}\}$.

▶ Proposition 12. There is a one-to-one correspondence between homomorphisms $\mathcal{A} \to \mathcal{B}$ and solutions for $\mathcal{K}_{AB}$.

We will now look at *empirical models over measurement scenarios*, introduced in [4] as a general setting for studying contextuality, in quantum mechanics and beyond, with non-locality as a special case.

A measurement scenario is a triple $(X, \mathcal{M}, O)$, where X is a set of measurement labels; $\mathcal{M}$ is a family of subsets of X, where we think of $C \in \mathcal{M}$ as a set of compatible measurements, or a context; and O is a set of measurement outcomes. An empirical model $e : (X, \mathcal{M}, O)$ for a scenario is a family $e = \{e_C\}_{C \in \mathcal{M}}$ of probability distributions $e_C \in \mathsf{Prob}(O^C)$ on the joint outcomes of measuring all the variables in a context C. Such empirical models can arise from observational data (hence the name), or be generated by measuring a quantum state in contexts comprising jointly measurable observables. A hierarchy of notions of contextuality can be defined in this general setting [4, 2]. We shall be concerned with *strong contextuality*. We say that $e : (X, \mathcal{M}, O)$ is strongly contextual if there is no global assignment $g : X \to O$ such that, for all $C \in \mathcal{M}$, $e_C(g|_C) > 0$. That is, there is no global assignment consistent with the model in the sense of yielding *possible* outcomes (non-zero probability) in all contexts. This form of contextuality is witnessed by the GHZ construction [15, 28], as well as by Kochen–Specker paradoxes [22], and post-quantum devices such as the PR box [33].

Given $e : (X, \mathcal{M}, O)$, we fix an ordering on X, and define a CSP $\mathcal{K}_e = (X, O, C)$, where C is the set of constraints $((x_1, \ldots, x_k), r)$ such that $x_1 < \cdots < x_k$, $\{x_1, \ldots, x_k\} \in C$ and $r = \{s(\mathbf{x}) \mid e_C(s) > 0\}$.

▶ Proposition 13. There is a one-to-one correspondence between consistent global assignments for e and solutions of $\mathcal{K}_e$. Thus e is strongly contextual iff $\mathcal{K}_e$ has no (classical) solution.

[1] One could have different domains associated with different variables. However, this is an inessential generalization, which we omit to keep notation simple.

We thus have a three-way correspondence between CSPs, empirical models, and homomorphisms between relational structures.

4.2 Quantum solutions

We now consider how quantum resources enter the picture. Since we already have a notion of quantum homomorphism for general relational structures, the correspondences established in the previous subsection give us ready-made notions of quantum solutions for CSPs and empirical models. We define a *quantum solution* for a CSP $\mathcal{K} = (V, D, C)$ to be a quantum homomorphism $\mathcal{A}_{\mathcal{K}} \xrightarrow{q} \mathcal{B}_{\mathcal{K}}$, i.e. a Kleisli morphism $\mathcal{A}_{\mathcal{K}} \to \mathcal{Q}_d\mathcal{B}_{\mathcal{K}}$ for some d. Similarly, we define a quantum solution for an empirical model $e : (X, \mathcal{M}, O)$ to be a quantum homomorphism $\mathcal{A}_{\mathcal{K}_e} \xrightarrow{q} \mathcal{B}_{\mathcal{K}_e}$.

We shall now compare these notions to existing ones for empirical models and constraints. These will turn out to be special cases.

Note first that given an empirical model e, the corresponding CSP $\mathcal{K}_e$ is determined purely by the *supports* of the probability distributions e_C, i.e. the possibilistic content of the model. It is only this information which is relevant to strong contextuality. We can consider a fine-grained notion of realization of a probabilistic empirical model, as in [4]. However, if our focus is strong contextuality, it is natural to consider the notion of *quantum witness* for an empirical model $e : (X, \mathcal{M}, O)$, given by a state ψ, and a PVM $P_x = \{P_{x,o}\}_{o \in O}$ for each $x \in X$, such that $[P_{x,o}, P_{x',o'}] = \mathbf{0}$ whenever x and x' both occur in some $C \in \mathcal{M}$. These must then satisfy, for all $C \in \mathcal{M}$ and $s \in O^C$, $e_C(s) = 0 \Rightarrow \psi^* P_{\mathbf{x},s(\mathbf{x})}\psi = 0$, where $P_{\mathbf{x},s(\mathbf{x})} = P_{x_1,s(x_1)} \cdots P_{x_k,s(x_k)}$. This provides a quantum witness for strong contextuality if e is a strongly contextual empirical model. An example is provided by the GHZ state, using X and Y measurements for each party [4]. An infinite family of such examples using three-qubit states is described in [1].

We can also consider a stronger notion. A *state-independent* quantum witness for $e : (X, \mathcal{M}, O)$ is given by a family of PVMs $\{P_x\}_{x \in X}$ which, for *any* state ψ, yield a quantum witness for e. The Mermin magic square and pentagram [29], and Kochen–Specker constructions [22, 10], provide examples of state-independent quantum realizations of strong contextuality. Note that in the state-independent case, we have the condition: $e_C(s) = 0 \Rightarrow P_{\mathbf{x},\mathbf{o}} = \mathbf{0}$. Comparison with the definition of quantum homomorphism $\mathcal{A}_{\mathcal{K}_e} \xrightarrow{q} \mathcal{B}_{\mathcal{K}_e}$ immediately yields the following result:

▶ Proposition 14. For an empirical model $e : (X, \mathcal{M}, O)$ there is a one-to-one correspondence between state-independent quantum witnesses for e and quantum solutions for $\mathcal{K}_e$.

An interesting point arising from this result, taken together with the results from Section 2, is that state-independent strong contextuality proofs can always be underwritten by non-locality arguments. This can be seen as a general form of constructions for turning Kochen–Specker contextuality proofs into Bell non-locality arguments [19]. Indeed, the role of the entangled state and of Bob in the non-local game is to provide an operational or physical underpinning for the compatibility or generalized no-signalling assumption which is made for empirical models [4]. Can we find a similar underpinning in the state-dependent case? We shall return to this point in the final section.

We now consider *binary constraint systems* (BCS), which have been extensively studied [13, 12, 36, 21]. We shall follow the account in [13]. A BCS (V, C) is simply a boolean CSP $(V, \{0,1\}, C)$. In this case, constraints can be written in the form $c = (\mathbf{x}, b_c)$, where $b_c : \{0,1\}^k \to \{0,1\}$ is a boolean function. Our general notion of quantum solution yields in this case a family of PVMs $P_x = \{P_{x,o}\}_{o \in \{0,1\}}$ for $x \in V$, such that $[P_{x,o}, P_{x',o'}] = \mathbf{0}$ whenever x and x' both occur in some constraint, and $P_{\mathbf{x},\mathbf{o}} = \mathbf{0}$ for $c = (\mathbf{x}, b_c)$ with $b_c(\mathbf{o}) = 0$.

In [13], a notion of *operator solution* for a BCS is defined. This is an assignment of a self-adjoint operator (aka observable) A_x to each variable x, such that: (1) each A_x is binary, i.e. $A_x^2 = I$; (2) $[A_x, A_{x'}] = \mathbf{0}$ when x and x' both occur in the same constraint. To express constraints, the representation of boolean values $b \mapsto (-1)^b$, $b \in \{0,1\}$, is used. It is standard that each boolean function $\{-1,+1\}^k \to \{-1,+1\}$ can be uniquely represented by a real multilinear polynomial $p(X_1, \ldots, X_k)$ [31]. Moreover, if the corresponding boolean function in the $\{0,1\}$-representation is $b : \{0,1\}^k \to \{0,1\}$, then for $\mathbf{o} \in \{0,1\}^k$: $(-1)^{b(\mathbf{o})} = p((-1)^{\mathbf{o}_1}, \ldots, (-1)^{\mathbf{o}_k})$. It is also standard that if we substitute pairwise commuting self-adjoint operators $A_1, \ldots, A_k$ for the variables $X_1, \ldots, X_k$, we obtain a self-adjoint operator $p(A_1, \ldots, A_k)$. The condition for an operator solution to satisfy the constraints is then expressed as follows: for each constraint $c = (\mathbf{x}, b_c)$, where b_c is represented by the polynomial $p_c(X_1, \ldots, X_k)$, we must have $p_c(A_1, \ldots, A_k) = -I$.

To relate this notion of operator solution to our quantum solution, note that a binary observable A_x with $A_x^2 = I$ has a spectral decomposition $A_x = P_{x,0} - P_{x,1}$, where $P_{x,0}$, $P_{x,1}$ are projectors. Commutation of the observables for variables occurring in the same context is equivalent to commutation of the corresponding projectors. Given a constraint $c = ((x_1, \ldots, x_k), p_c)$, since the observables $A_{x_1}, \ldots, A_{x_k}$ pairwise commute, we obtain a resolution of the identity $\sum_{\mathbf{o} \in \{0,1\}^k} P_{\mathbf{x},\mathbf{o}} = I$. Multiplying $p_c(A_1, \ldots, A_k)$ by this expression yields

$$p_c(A_1, \ldots, A_k) \;=\; \sum_{\mathbf{o} \in \{0,1\}^k} p_c((-1)^{\mathbf{o}_1}, \ldots, (-1)^{\mathbf{o}_k}) P_{\mathbf{x},\mathbf{o}}.$$

It follows that $p_c(A_1, \ldots, A_k) = -I$ iff for all $\mathbf{o}$ with $b_c(\mathbf{o}) = 0$, $P_{\mathbf{x},\mathbf{o}} = \mathbf{0}$. As an immediate consequence, we have:

▶ Proposition 15. Given a BCS (V, C), there is a one-to-one correspondence between operator solutions of the BCS and quantum solutions of the corresponding CSP.

4.3 Graphs

The results we have seen thus far show that our notion of quantum homomorphism subsumes a number of existing notions in contextuality and non-local games. However, as we shall now see, the situation in the setting which provided the original motivation for our approach, namely graph homomorphisms, is somewhat more subtle.

Graphs arise as structures for the signature with a single binary relation. Simple graphs are those where the relation is symmetric and irreflexive. If we specialize our definition of quantum homomorphism to the case $G \xrightarrow{q} H$ between graphs G and H, this gives a family $\{P_{x,y}\}_{x \in V(G), y \in V(H)}$ of projectors satisfying the following conditions:

- for all $x, x' \in V(G)$ and $y, y' \in V(H)$ with $x \sim x'$, $[P_{x,y}, P_{x',y'}] = 0$;
- for all $x, x' \in V(G)$ and $y, y' \in V(H)$ with $x \sim x'$ and $y \not\sim y'$, $P_{x,y} P_{x',y'} = 0$.

This definition differs from that introduced by Mančinska and Roberson [27] as a generalization of quantum graph colouring [11], in that the latter does *not* impose the first condition forcing commutativity between the PVMs corresponding to adjacent vertices of G. We refer to that as an *MR quantum graph homomorphism* in order to distinguish it from our notion.[2]

This reflects a difference in the Alice–Bob game used to motivate each definition. In the game we consider, Alice receives an ordered edge of G as an input and Bob a vertex of G,

[2] See also the locally commuting graph homomorphisms from [18], which require the commutativity condition, but differ from ours by not restricting to finite-dimensional quantum resources.

and we require that Alice answers with an edge of H and Bob with a vertex of H in a fashion consistent with Alice's choice. By contrast, Mančinska and Roberson consider a symmetric Alice–Bob game, where each player receives a vertex of G as input and outputs a vertex of H, and their outputs are required to be the same when they receive the same vertex, and required to form an edge of H whenever their inputs form an edge of G.

It is clear that a quantum homomorphism between graphs in the sense of this paper is also an MR quantum graph homomorphism. But the precise relationship between the two notions is yet to be understood in general: in particular, whether the existence of an MR quantum graph homomorphism implies the existence of a quantum homomorphism in our sense. However, by adapting a construction due to Ji [21], we can capture the existence of MR quantum graph homomorphisms in terms of quantum homomorphisms of relational structures, via a BCS.

Given graphs G and H, we define $V = \{r_{xy} \mid x \in V(G), y \in V(H)\}$. For each $x \in V(G)$, we have a boolean constraint $\bigvee_y r_{xy}$; a constraint $\neg(r_{xy} \wedge r_{xy'})$ when $y \neq y'$; and a constraint $\neg(r_{xy} \wedge r_{x'y'})$ whenever $x \sim x'$ and $y \not\sim y'$. This defines a BCS (V, C).

▶ **Theorem 16.** *Given graphs G and H, there is a one-to-one correspondence between MR quantum graph homomorphisms from G to H and quantum homomorphisms $\mathcal{A}_\mathcal{K} \xrightarrow{q} \mathcal{B}_\mathcal{K}$ for the associated CSP $\mathcal{K} = (V, \{0,1\}, C)$.*

Proof. We recall that an MR quantum graph homomorphism is given by a family of projectors $\{P_{x,y}\}_{x \in V(G), y \in V(H)}$ such that (MR1) $\sum_y P_{x,y} = I$, and (MR2) $P_{x,y}P_{x',y'} = \mathbf{0}$ whenever $x \sim x'$ and $y \not\sim y'$. A quantum homomorphism $\mathcal{A}_\mathcal{K} \xrightarrow{q} \mathcal{B}_\mathcal{K}$ is given by a family of projectors $\{Q_{xy,o}\}$, $x \in V(G)$, $y \in V(H)$, $o \in \{0,1\}$, such that the following conditions hold: (QH1) $Q_{xy,0} + Q_{xy,1} = I$; (QH2) $Q_{xy,1}Q_{xy',1} = \mathbf{0}$, $(y \neq y')$; (QH3) $Q_{xy,1}Q_{x'y',1} = \mathbf{0}$, $(x \sim x' \wedge y \not\sim y')$; (QH4) $Q_{xy_1,0} \cdots Q_{xy_k,0} = \mathbf{0}$, $V(H) = \{y_1, \ldots, y_k\}$. The commutativity conditions which are additionally required are implied by the orthogonality conditions (QH2) and (QH3).

Given an MR homomorphism $\{P_{x,y}\}$, we define $Q_{xy,1} := P_{x,y}$, $Q_{xy,0} := I - P_{x,y}$. Clearly (QH1) is satisfied. (MR1) implies (QH2), while (MR2) implies (QH3). Finally, using (QH2), $(I - P_{x,y_1}) \cdots (I - P_{x,y_k}) = I - \sum_y P_{x,y}$, and by (MR1), (QH4) holds.

Conversely, given a quantum homomorphism $\{Q_{xy,o}\}$, we define $P_{x,y} := Q_{xy,1}$. (MR2) follows from (QH3), while using (QH1), we can reverse the reasoning in the previous paragraph to show that (QH4) implies (MR1). These passages are clearly mutually inverse, so the result follows. ◀

It is noteworthy that our approach allows us to avoid *ad hoc* coding of constraints by polynomials, as in [13, 21]. Instead, we quantize the standard classical notions in a uniform way, using the quantum monad.

5 Outlook

This work suggests a number of directions for further study. We list a few:

- A notion of quantum graph isomorphism, with an equivalent characterization via an Alice–Bob game, has been studied in [8]. This can be generalized to relational structures. How does this fit in our quantum monad framework?
- Our approach captures quantum advantage provided by state-independent strong contextuality. Does state-dependent contextuality admit a similar treatment?
- Any strategy for an Alice–Bob game has a winning probability, which is related to the contextual fraction [3]. Can our approach be adapted to deal with quantitative aspects?

- Homomorphisms are intimately related with the existential positive fragment. Can this be extended to provide a notion of quantum validity for first-order formulae?
- Can other concepts from finite model theory, such as pebble games, which admit a comonadic formulation [7], be similarly quantized?
- The algebras of the quantum monad can be described as convex structures with mixing weighted by projectors rather than just numbers in $[0, 1]$. Is this viewpoint useful?

Acknowledgements. We would like to thank Jaroslav Nešetřil, Laura Mančinska, David Roberson, and Simone Severini for helpful suggestions, comments, and discussions.

References

1 Samson Abramsky, Rui Soares Barbosa, Giovanni Carù, Nadish de Silva, Kohei Kishida, and Shane Mansfield. Minimum quantum resources for strong non-locality, 2017. To appear in *Proceedings of the 12th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2017)*. Available as arXiv:1705.09312 [quant-ph].

2 Samson Abramsky, Rui Soares Barbosa, Kohei Kishida, Raymond Lal, and Shane Mansfield. Contextuality, cohomology and paradox. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 211–228. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.211.

3 Samson Abramsky, Rui Soares Barbosa, and Shane Mansfield. Contextual fraction as a measure of contextuality, 2017. To appear in *Physical Review Letters*. Available as arXiv:1705.07918 [quant-ph].

4 Samson Abramsky and Adam Brandenburger. The sheaf-theoretic structure of non-locality and contextuality. *New Journal of Physics*, 13(11):113036, 2011. doi:10.1088/1367-2630/13/11/113036.

5 Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LiCS 2004)*, pages 415–425, 2004. doi:10.1109/LICS.2004.1319636.

6 Samson Abramsky and Bob Coecke. Categorical quantum mechanics. In Kurt Engesser, Dov M. Gabbay, and Daniel Lehmann, editors, *Handbook of quantum logic and quantum structures: Quantum logic*, pages 261–323. Elsevier, 2009. doi:10.1016/B978-0-444-52869-8.50010-4.

7 Samson Abramsky, Anuj Dawar, and Pengming Wang. The pebbling comonad in finite model theory, 2017. To appear in *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LiCS 2017)*. Available as arXiv:1704.05124 [cs.LO].

8 Albert Atserias, Laura Mančinska, David E Roberson, Robert Šámal, Simone Severini, and Antonios Varvitsiotis. Quantum and non-signalling graph isomorphisms, 2016. Available as arXiv:1611.09837 [quant-ph].

9 Bruce Blackadar. *Operator algebras: Theory of C^*-algebras and von Neumann algebras*, volume 122 of *Encyclopaedia of Mathematical Sciences*. Springer, 2006. doi:10.1007/3-540-28517-2.

10 Adán Cabello, José M. Estebaranz, and Guillermo García-Alcaine. Bell-Kochen-Specker theorem: A proof with 18 vectors. *Physics Letters A*, 212(4):183–187, 1996. doi:10.1016/0375-9601(96)00134-X.

11 Peter J. Cameron, Ashley Montanaro, Michael W. Newman, Simone Severini, and Andreas Winter. On the quantum chromatic number of a graph. *Electronic Journal of Combinatorics*, 14(1):R81, 2007.

12 Richard Cleve, Li Liu, and William Slofstra. Perfect commuting-operator strategies for linear system games. *Journal of Mathematical Physics*, 58(1):012202, 2017. doi:10.1063/1.4973422.

13 Richard Cleve and Rajat Mittal. Characterization of binary constraint system games. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming, Part I (ICALP 2014)*, pages 320–331. Springer, 2014. doi:10.1007/978-3-662-43948-7_27.

14 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

15 Daniel M. Greenberger, Michael A. Horne, and Anton Zeilinger. Going beyond Bell's theorem. In Menas Kafatos, editor, *Bell's theorem, quantum theory, and conceptions of the universe*, volume 37 of *Fundamental Theories of Physics*, pages 69–72. Kluwer, 1989. doi:10.1007/978-94-017-0849-4_10.

16 Teiko Heinosaari, Daniel Reitzner, and Peter Stano. Notes on joint measurability of quantum observables. *Foundations of Physics*, 38(12):1133–1147, 2008. doi:10.1007/s10701-008-9256-7.

17 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford Lecture Series in Mathematics and Its Applications*. Oxford University Press, 2004. doi:10.1093/acprof:oso/9780198528173.001.0001.

18 William Helton, Kyle P. Meyer, Vern I. Paulsen, and Matthew Satriano. Algebras, synchronous games and chromatic numbers of graphs, 2017. Available as arXiv:1703.00960 [math.OA].

19 Peter Heywood and Michael L. G. Redhead. Nonlocality and the Kochen–Specker paradox. *Foundations of physics*, 13(5):481–499, 1983. doi:10.1007/BF00729511.

20 Bart Jacobs. Convexity, duality and effects. In Cristian S. Calude and Vladimiro Sassone, editors, *Proceedings of 6th IFIP TC 1/WG 2.2 International Conference on Theoretical Computer Science (TCS 2010)*, pages 1–19. Springer, 2010. doi:10.1007/978-3-642-15240-5_1.

21 Zhengfeng Ji. Binary constraint system games and locally commutative reductions, 2013. Available as arXiv:1310.3794 [quant-ph].

22 Simon Kochen and Ernst P. Specker. The problem of hidden variables in quantum mechanics. *Journal of Mathematics and Mechanics*, 17(1):59–87, 1967.

23 Anders Kock. Monads on symmetric monoidal closed categories. *Archiv der Mathematik*, 21(1):1–10, 1970. doi:10.1007/BF01220868.

24 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61(2):302–332, 2000. doi:10.1006/jcss.2000.1713.

25 Leonid Libkin. *Elements of finite model theory*. Texts in Theoretical Computer Science. Springer, 2004. doi:10.1007/978-3-662-07003-1.

26 Saunders Mac Lane. *Categories for the working mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, 1971. doi:10.1007/978-1-4757-4721-8.

27 Laura Mančinska and David E Roberson. Quantum homomorphisms. *Journal of Combinatorial Theory, Series B*, 118:228–267, 2016. doi:10.1016/j.jctb.2015.12.009.

28 N. David Mermin. Quantum mysteries revisited. *American Journal of Physics*, 58(8):731–734, 1990. doi:10.1119/1.16503.

29 N. David Mermin. Simple unified form for the major no-hidden-variables theorems. *Physical Review Letters*, 65(27):3373–3376, Dec 1990. doi:10.1103/PhysRevLett.65.3373.

30 Stefan Milius, Dirk Pattinson, and Lutz Schröder. Generic trace semantics and graded monads. In Lawrence S. Moss and Pawel Sobocinski, editors, *6th Conference on Algebra*

and Coalgebra in Computer Science (CALCO 2015), volume 35 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 253–269. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CALCO.2015.253.

31 Ryan O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014. doi:10.1017/CBO9781139814782.

32 Gert K. Pedersen. *Analysis now*, volume 118 of *Graduate Texts in Mathematics*. Springer, 1989. doi:10.1007/978-1-4612-1007-8.

33 Sandu Popescu and Daniel Rohrlich. Quantum nonlocality as an axiom. *Foundations of Physics*, 24(3):379–385, 1994. doi:10.1007/BF02058098.

34 David E. Roberson. *Variations on a theme: Graph homomorphisms*. PhD thesis, University of Waterloo, 2013.

35 Volkher B. Scholz and Reinhard F. Werner. Tsirelson's problem, 2008. Available as arXiv:0812.4305 [math-ph].

36 William Slofstra. Tsirelson's problem and an embedding theorem for groups arising from non-local games, 2016. Available as arXiv:1606.03140 [quant-ph].

37 Boris Tsirelson. Bell inequalities and operator algebras, 2006. Pre-print.

A Review of linear algebra and quantum mechanics background

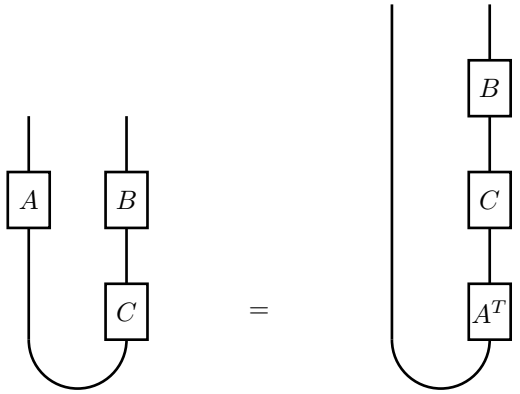
Since we are working in finite dimensions, we will use standard matrix notation. Thus we are dealing with $d \times d'$ complex matrices. We write matrix transpose as A^T. Apart from the usual operations of matrix addition and multiplication, there is the adjoint A^*, which is the conjugate transpose of A. Thus $[a_{i,j}]^* = [\overline{a_{j,i}}]$. The zero matrix is $\mathbf{0}$, the identity matrix in dimension d is $I = I_d$.[3] We view $d' \times d$ complex matrices interchangably as linear maps $\mathbb{C}^d \to \mathbb{C}^{d'}$, acting on "column vectors", i.e. $d \times 1$ matrices. We identify 1×1 matrices with scalars, i.e. complex numbers. The inner product of vectors $\mathbf{x}, \mathbf{y} \in \mathbb{C}^d$ is given by the matrix product $\mathbf{x}^*\mathbf{y}$. The norm of a vector $\mathbf{x}$ is $\|\mathbf{x}\| := \sqrt{\mathbf{x}^*\mathbf{x}}$. The standard basis vectors in dimension d are $e_1, \ldots, e_d$, where e_i has i'th component 1, and all other components 0.

A square matrix A is self-adjoint (aka Hermitian) if $A^* = A$. It is positive semidefinite if it self-adjoint, and $\mathbf{x}^*A\mathbf{x} \geq 0$ for all vectors $\mathbf{x}$. If A is positive semidefinite, so is C^*AC for any C. We have the order $A \leq B$ if $B - A$ is positive semidefinite. The condition $A \geq \mathbf{0}$ says exactly that A is positive semidefinite. The matrices $A \geq \mathbf{0}$ form a convex cone. A is a projector if $A^* = A = A^2$. We write $\mathsf{Proj}(d)$ for the set of $d \times d$ projectors. A fact we shall use frequently is that for any family of projectors $\{P_i\}$ in $\mathsf{Proj}(d)$, $\sum_i P_i \leq I$ iff $P_iP_j = \mathbf{0}$ whenever $i \neq j$.

If $A = [a_{i,j}]$ is a $m \times n$ matrix and B a $p \times q$ matrix, then the Kronecker product $A \otimes B := [a_{i,j}B]$ is an $mp \times nq$ matrix, which represents the tensor product of the corresponding linear maps. This operation is strictly associative, with unit $\mathbf{1} := [1]$. The key equation is the interchange law with matrix multiplication: $(A \otimes B)(C \otimes D) = AC \otimes BD$. This is functoriality. The category of complex matrices is a strict monoidal category with respect to this operation. Indeed, the category of complex matrices is equivalent to the category of finite-dimensional Hilbert spaces at the level of dagger compact closed categories, the basic setting for categorical quantum mechanics [5, 6]. The final operation we consider is vectorization of a matrix: $\mathsf{vec}(A)$ turns a $d \times d'$ matrix into a dd'-vector by stacking the columns of A on top of each other. In terms of the closed structure on the category of matrices, it is the name of the morphism A. The "cup" or unit of the compact closed

[3] We will omit dimensional subscripts whenever we can get away with it.

structure is $\mathsf{vec}(I)$. We have the equation $\mathsf{vec}(A) = (I \otimes A)\mathsf{vec}(I)$, and the "sliding rule": $(A \otimes I)\mathsf{vec}(I) = (I \otimes A^T)\mathsf{vec}(I)$, from which we can derive the key equation for vectorization: $(A \otimes B)\mathsf{vec}(C) = \mathsf{vec}(BCA^T)$. Diagrammatically, this is



A vector $\psi \in \mathbb{C}^{d_A} \otimes \mathbb{C}^{d_B}$ has a *Schmidt decomposition* $\psi = \sum_{i=1}^{d} \lambda_i \alpha_i \otimes \beta_i$, where $d \leq \min(d_a, d_b)$, $\{\alpha_i\}$ and $\{\beta_i\}$ are orthonormal sets of vectors in $\mathbb{C}^{d_A}$ and $\mathbb{C}^{d_B}$ respectively, and $\lambda_i > 0$ for all i. This follows directly from Singular Value Decomposition. We refer to d as the *Schmidt rank* of ψ.

The following result is standard, but we did not find an explicit reference so we include a proof.

▶ Proposition 17. Let A and B be positive semidefinite matrices. Then $\mathsf{Tr}(AB) = 0 \iff AB = \mathbf{0}$.

Proof. Since A and B are positive semidefinite, they have positive semidefinite square roots $\sqrt{A}$, $\sqrt{B}$, and $\mathsf{Tr}(AB) = \mathsf{Tr}(\sqrt{A}\sqrt{A}\sqrt{B}\sqrt{B}) = \mathsf{Tr}(\sqrt{B}\sqrt{A}\sqrt{A}\sqrt{B}) = \mathsf{Tr}(C^*C)$, where $C := \sqrt{A}\sqrt{B}$. Since C^*C is positive semidefinite, $\mathsf{Tr}(C^*C) = 0 \iff C^*C = \mathbf{0} \iff C = \mathbf{0}$. But $C = \mathbf{0}$ implies $AB = \sqrt{A}C\sqrt{B} = \mathbf{0}$. ◀

Now we briefly review the needed notions from quantum mechanics. A (pure) state in dimension d is a vector of unit norm in $\mathbb{C}^d$. A POVM (positive operator-valued measure) is a family $\{A_i\}_i$ with $A_i \geq \mathbf{0}$ for all i, and $\sum_i A_i = I$. The indices i label the measurement outcomes. Measuring a POVM $\{A_i\}_i$ on a state ψ yields outcome i with probability $\psi^* A_i \psi$. A POVM is projective (or a PVM) if A_i is a projector for all i. This implies that $A_i A_j = \mathbf{0}$ for all $i \neq j$, i.e. the projectors are mutually orthogonal. The product of projectors is a projector if and only if they commute, which is usually written as $[P, Q] = \mathbf{0}$, where $[P, Q] := PQ - QP$.

B Quantum monad

▶ Proposition 18. For each $\mathcal{A}$, $\mu_{\mathcal{A}}^{d,d'}$ is a well-defined homomorphism, and yields a natural transformation.

Proof. First, $\mu_{\mathcal{A}}^{d,d'}$ is a well-defined function, by finiteness of support. To show that it is a homomorphism, consider $(P_1, ..., P_k) \in R^{\mathcal{Q}_d \mathcal{Q}_{d'} \mathcal{A}}$. We must first show that for all $z, z' \in C$, and i, j, $[p_i(z), p_j(z')] = \mathbf{0}$, where $p_i := \mu_{\mathcal{A}}^{d,d'}(P_i)$, $p_j := \mu_{\mathcal{A}}^{d,d'}(P_j)$. Using linearity, this reduces to showing that $P_i(p) \otimes p(z)$ commutes with $P_j(p') \otimes p'(z')$ for all $p, p' \in \mathcal{Q}_{d'}$. Applying (QR1) to P_i and P_j, we have that $P_i(p)$ commutes with $P_j(p')$. If $p(z)$ commutes with $p'(z')$, we are done. If not, then we know that p cannot be adjacent to p' in the Gaifman graph of $\mathcal{Q}_{d'}$. Hence for any expansion $\mathbf{p} = (p_1, \ldots, p_k)$ with $p_i = p$, $p_j = p'$,

$\mathbf{p} \notin R^{\mathcal{Q}_{d'}\mathcal{A}}$, so by (QR2) we must have $P_1(p_1)\cdots P_k(p_k) = \mathbf{0}$. Using normalization, we have $P_i(p)P_j(p') = \sum_{\mathbf{p}_i=p,\mathbf{p}_j=p'} P_1(p_1)\cdots P_k(p_k) = \mathbf{0}$, and so

$$(P_i(p)\otimes p(z))(P_j(p')\otimes p'(z')) = \mathbf{0} = (P_j(p')\otimes p(z'))(P_i(p)\otimes p(z)).$$

Now let $q_i = \mu_{\mathcal{A}}^{d,d'}(P_i)$, $i = 1,\ldots,k$. To show that (QR2) holds for $(q_1,\ldots,q_k)$ reduces similarly to showing that, if $\mathbf{x} \notin R^{\mathcal{A}}$, then $P_1(p_1)\cdots P_k(p_k)\otimes p_1(x_1)\cdots p_k(x_k) = \mathbf{0}$ for all $p_1,\ldots,p_k$. If $p_1(x_1)\cdots p_k(x_k) = \mathbf{0}$ we are done; otherwise, since $\mathbf{x} \notin R^{\mathcal{A}}$, we must have $\mathbf{p} \notin R^{\mathcal{Q}_{d'}\mathcal{A}}$, and applying (QR2) to $(P_1,...,P_k)$, we must have $P_1(p_1)\cdots P_k(p_k) = \mathbf{0}$.

Naturality is commutativity of the following square.

$$\begin{array}{ccc} \mathcal{Q}_d\mathcal{Q}_{d'}\mathcal{A} & \xrightarrow{\mu_{\mathcal{A}}^{d,d'}} & \mathcal{Q}_{dd'}\mathcal{A} \\ \downarrow{\scriptstyle \mathcal{Q}_d\mathcal{Q}_{d'}f} & & \downarrow{\scriptstyle \mathcal{Q}_{dd'}f} \\ \mathcal{Q}_d\mathcal{Q}_{d'}\mathcal{B} & \xrightarrow[\mu_{\mathcal{B}}^{d,d'}]{} & \mathcal{Q}_{dd'}\mathcal{B} \end{array}$$

This is the following calculation:

$$\begin{aligned} \mathcal{Q}_{dd'}f\circ\mu_{\mathcal{A}}^{d,d'}(P)(y) &= \sum_{f(x)=y}\sum_{p} P(p)\otimes p(x) \\ &= \sum_{q}\sum_{\mathcal{Q}_{d'}f(p)=q} P(p)\otimes q(y) \\ &= \sum_{q}\mathcal{Q}_d\mathcal{Q}_{d'}(f)(P)(q)\otimes q(y) \\ &= \mu_{\mathcal{B}}^{d,d'}\circ\mathcal{Q}_d\mathcal{Q}_{d'}f(P)(y). \end{aligned}$$

The second step uses the fact that $\mathcal{Q}_{d'}f(p) = q \iff q(y) = \sum_{f(x)=y} p(x)$. ◀

The unit $\eta : \mathsf{Id} \xrightarrow{\cdot} T_1$ and graded multiplication $\mu^{m,m'} : T_mT_{m'} \xrightarrow{\cdot} T_{m\cdot m'}$ of a graded M-monad are required to satisfy the following coherence conditions:

$$\begin{array}{ccc} T_mX & \xrightarrow{\eta_{T_mX}} & T_1T_mX \\ \downarrow{\scriptstyle T_m\eta_X} & \searrow & \downarrow{\scriptstyle \mu_X^{1,m}} \\ T_mT_1X & \xrightarrow[\mu_X^{m,1}]{} & T_mX \end{array} \qquad \begin{array}{ccc} T_mT_{m'}T_{m''}X & \xrightarrow{T_m\mu_X^{m',m''}} & T_mT_{m'\cdot m''}X \\ \downarrow{\scriptstyle \mu^{m,m'}_{T_{m''}X}} & & \downarrow{\scriptstyle \mu_X^{m,m'\cdot m''}} \\ T_{m\cdot m'}T_{m''}X & \xrightarrow[\mu_X^{m\cdot m',m''}]{} & T_{m\cdot m'\cdot m''}X. \end{array}$$

We verify these for the quantum monad.

▶ **Lemma 19.** *Let $\mathcal{A}$ be a structure and $d \in \mathbb{N}^+$. Then, the following diagram commutes:*

$$\begin{array}{ccc} \mathcal{Q}_d\mathcal{A} & \xrightarrow{\eta_{\mathcal{Q}_d\mathcal{A}}} & \mathcal{Q}_1\mathcal{Q}_d\mathcal{A} \\ \downarrow{\scriptstyle \mathcal{Q}_d\eta_{\mathcal{A}}} & \searrow & \downarrow{\scriptstyle \mu_{\mathcal{A}}^{1,d}} \\ \mathcal{Q}_d\mathcal{Q}_1\mathcal{A} & \xrightarrow[\mu_{\mathcal{A}}^{d,1}]{} & \mathcal{Q}_d\mathcal{A} \end{array}$$

Proof. Let $p \in \mathcal{Q}_d\mathcal{A}$ and $x \in A$. The claim is that $(\mu_{\mathcal{A}}^{1,d} \circ \eta_{\mathcal{Q}_d\mathcal{A}})(p)(x) = p(x) = (\mu_{\mathcal{A}}^{d,1} \circ \mathcal{Q}_d\eta_{\mathcal{A}})(p)(x)$. The left-hand side of this equation expands to

$$\mu_{\mathcal{A}}^{1,d}(\eta_{\mathcal{Q}_d\mathcal{A}}(p))(x) = \sum_{p' \in \mathcal{Q}_d\mathcal{A}} \eta_{\mathcal{Q}_d\mathcal{A}}(p)(p') \otimes p'(x) = p(x),$$

and the right-hand side to

$$\begin{aligned} \mu_{\mathcal{A}}^{d,1}(\mathcal{Q}_d\eta_{\mathcal{A}}(p))(x) &= \sum_{p' \in \mathcal{Q}_1\mathcal{A}} \mathcal{Q}_d\eta_{\mathcal{A}}(p)(p') \otimes p'(x) \\ &= \mathcal{Q}_d\eta_{\mathcal{A}}(p)(\delta_x) \\ &= \sum_{\eta_{\mathcal{A}}(x')=\delta_x} \eta_{\mathcal{A}}(p)(x') \\ &= p(x), \end{aligned}$$

where $\delta_x : A \to \{0,1\}$ is defined by $\delta_x(x') := \delta_{x,x'}$ for all $x' \in A$. ◀

▶ **Lemma 20.** *Let $\mathcal{A}$ be a structure and $a, b, c \in \mathbb{N}^+$. Then, the following diagram commutes:*

$$\begin{array}{ccc} \mathcal{Q}_a\mathcal{Q}_b\mathcal{Q}_c\mathcal{A} & \xrightarrow{\mathcal{Q}_a\mu_{\mathcal{A}}^{b,c}} & \mathcal{Q}_a\mathcal{Q}_{bc}\mathcal{A} \\ \downarrow{\scriptstyle \mu_{\mathcal{Q}_c\mathcal{A}}^{a,b}} & & \downarrow{\scriptstyle \mu_{\mathcal{A}}^{a,bc}} \\ \mathcal{Q}_{ab}\mathcal{Q}_c\mathcal{A} & \xrightarrow[\mu_{\mathcal{A}}^{ab,c}]{} & \mathcal{Q}_{abc}\mathcal{A} \end{array}$$

Proof. Let $P \in \mathcal{Q}_a\mathcal{Q}_b\mathcal{Q}_c\mathcal{A}$ and $x \in A$. The claim is that $(\mu_{\mathcal{A}}^{a,bc} \circ \mathcal{Q}_a\mu_{\mathcal{A}}^{b,c})(P)(x) = (\mu_{\mathcal{A}}^{ab,c} \circ$

$\mu^{a,b}_{\mathcal{Q}_c\mathcal{A}})(P)(x)$. We have:

$$
\begin{aligned}
& \mu^{a,bc}_{\mathcal{A}}(\mathcal{Q}_a\mu^{b,c}_{\mathcal{A}}(P))(x) \\
= & \sum_{q\in\mathcal{Q}_{bc}\mathcal{A}} \mathcal{Q}_a\mu^{b,c}_{\mathcal{A}}(P)(q)\otimes q(x) \\
= & \sum_{q\in\mathcal{Q}_{bc}\mathcal{A}} \left(\sum_{\mu^{b,c}_{\mathcal{A}}(p')=q} P(p')\right)\otimes q(x) \\
= & \sum_{q\in\mathcal{Q}_{bc}\mathcal{A}} \sum_{\mu^{b,c}_{\mathcal{A}}(p')=q} P(p')\otimes \mu^{b,c}_{\mathcal{A}}(p')(x) \\
= & \sum_{q\in\mathcal{Q}_{bc}\mathcal{A}} \sum_{\mu^{b,c}_{\mathcal{A}}(p')=q} P(p')\otimes \left(\sum_{p\in\mathcal{Q}_c\mathcal{A}} p'(p)\otimes p(x)\right) \\
= & \sum_{q\in\mathcal{Q}_{bc}\mathcal{A}} \sum_{\mu^{b,c}_{\mathcal{A}}(p')=q} \sum_{p\in\mathcal{Q}_c\mathcal{A}} P(p')\otimes p'(p)\otimes p(x) \\
= & \sum_{p'\in\mathcal{Q}_b(\mathcal{Q}_c\mathcal{A})} \sum_{p\in\mathcal{Q}_c\mathcal{A}} P(p')\otimes p'(p)\otimes p(x) \\
= & \sum_{p\in\mathcal{Q}_c\mathcal{A}} \left(\sum_{p'\in\mathcal{Q}_b(\mathcal{Q}_c\mathcal{A})} P(p')\otimes p'(p)\right)\otimes p(x) \\
= & \sum_{p\in\mathcal{Q}_c\mathcal{A}} \mu^{a,b}_{\mathcal{Q}_c\mathcal{A}}(P)(p)\otimes p(x) \\
= & \ \mu^{ab,c}_{\mathcal{A}}(\mu^{a,b}_{\mathcal{Q}_c\mathcal{A}}(P))(x)
\end{aligned}
$$

◀

Additional material on the quantum monad

We shall now show that the quantum monad is monoidal (or commutative) and affine [23]. This continues the analogy with the distribution monad, which is well known to have these properties [20].

The category $\mathcal{R}(\sigma)$ has finite products, given by the usual cartesian product of structures. The terminal object $\top$ is the one-element structure, with each relation interpreted as the universal relation. Because of normalization, the following is immediate:

▶ Proposition 21. For all d, $\mathcal{Q}_d\top \cong \top$. Thus the quantum monad is affine.

Now given structures $\mathcal{A}$ and $\mathcal{B}$, we define a map $m^{d,d'}_{\mathcal{A},\mathcal{B}} : \mathcal{Q}_d\mathcal{A}\times\mathcal{Q}_{d'}\mathcal{B}\to\mathcal{Q}_{dd'}(A\times B)$ by $m^{d,d'}_{\mathcal{A},\mathcal{B}}(p,q)(x,y) := p(x)\otimes q(y)$.

▶ Proposition 22. This is a well-defined homomorphism, and the family $\{m^{d,d'}_{\mathcal{A},\mathcal{B}}\}$ defines a graded natural transformation satisfying the monoidal coherence conditions, thus witnessing a commutative strength.

Towards a Polynomial Kernel for Directed Feedback Vertex Set*

Benjamin Bergougnoux[1], Eduard Eiben[2], Robert Ganian[3], Sebastian Ordyniak[4], and M. S. Ramanujan[5]

1 **Université Clermont Auvergne, LIMOS, CNRS, Aubière, France**
benjamin.bergougnoux@uca.fr
2 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
eiben@ac.tuwien.ac.at
3 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
ganian@ac.tuwien.ac.at
4 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
ordyniak@ac.tuwien.ac.at
5 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
ramanujan@ac.tuwien.ac.at

Abstract

In the Directed Feedback Vertex Set (DFVS) problem, the input is a directed graph D and an integer k. The objective is to determine whether there exists a set of at most k vertices intersecting every directed cycle of D. DFVS was shown to be fixed-parameter tractable when parameterized by solution size by Chen, Liu, Lu, O'Sullivan and Razgon [JACM 2008]; since then, the existence of a polynomial kernel for this problem has become one of the largest open problems in the area of parameterized algorithmics.

In this paper, we study DFVS parameterized by the feedback vertex set number of the underlying *undirected graph*. We provide two main contributions: a polynomial kernel for this problem on general instances, and a linear kernel for the case where the input digraph is embeddable on a surface of bounded genus.

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

Keywords and phrases parameterized algorithms, kernelization, (directed) feedback vertex set

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.36

1 Introduction

Feedback Set problems are fundamental combinatorial optimization problems. Typically, in these problems, we are given a graph G (directed or undirected) and a positive integer k, and the objective is to select at most k vertices, edges or arcs to hit all cycles of the input graph. Feedback Set problems are among Karp's 21 NP-complete problems [27] and have been a topic of active research from algorithmic [1, 2, 3, 6, 7, 8, 9, 10, 13, 14, 20, 23, 28, 25, 30, 32, 37] as well as structural points of view [19, 26, 29, 31, 33, 34, 35]. In particular, such problems constitute one of the most important topics of research in parameterized algorithms [6, 8, 9, 10, 13, 14, 28, 25, 30, 32, 37], spearheading the development of several

* Supported by the French Agency for Research under the GraphEN project (ANR-15-CE-0009) and by the Austrian Science Fund (FWF), projects P26696 and W1255-N23. Robert Ganian is also affiliated with FI MU, Brno, Czech Republic.

new techniques. In this paper, we study the DFVS problem, where the objective is to find a set of k vertices that intersects all directed cycles in a given digraph.

For over a decade resolving the fixed-parameter tractability of DFVS (whether there is an algorithm running in time $f(k)\cdot n^{\mathcal{O}(1)}$ for some computable function f) was considered the most important open problem in parameterized complexity. In fact, this problem was posed as an open problem in the first few papers on fixed-parameter tractability (FPT) [16, 17]. The problem can be formally stated as follows.

Directed Feedback Vertex Set (DFVS)	
Instance:	A digraph D and a positive integer k.
Parameter:	k
Question:	Does there exist a vertex subset of size at most k that intersects every cycle in D?

DFVS was shown to be fixed-parameter tractable in a breakthrough paper by Chen, Liu, Lu, O'Sullivan and Razgon [9] in 2008.

One of the most natural follow-up questions once a problem has been classified as fixed-parameter tractable is 'does it admit a polynomial kernel?'. A *polynomial kernel* is essentially a polynomial-time preprocessing algorithm that transforms the given instance of the problem into an equivalent one whose size is bounded polynomially in the parameter. Following the resolution of the fixed-parameter tractability of DFVS, the question of whether the problem admits a polynomial kernel was raised and has since become one of the main open problems in the area of parameterized complexity.

The most frequently used way of parameterizing problems, like in the case of DFVS, is taking the solution size as the parameter. An alternate method of parameterization is choosing a parameter that never exceeds the solution size but can potentially be much smaller. A classical example of this approach is parameterizing the Vertex Cover problem by the feedback vertex set (FVS) number. Clearly, the number of vertices required to make a graph acyclic never exceeds the number of vertices needed to make the graph edgeless. On the other hand, the feedback vertex set number could be arbitrarily smaller than the size of the smallest vertex cover. This problem was first studied by Bodlaender and Jansen [24], who showed that Vertex Cover parameterized by the feedback vertex number has a polynomial kernel. This was later extended by Cygan et al. [12] who systematically studied several generalizations of this problem and obtained several positive as well as negative results with respect to the existence of polynomial kernels.

While this kind of an alternate parameterization is interesting for problems which are known to have polynomial kernels when parameterized by the solution size, it is the *exact opposite* approach which is useful when dealing with problems for which this question has been answered negatively or remains open. This paper deals with such a parameterization for DFVS as an intermediate step towards answering the main question, that of a polynomial kernel for DFVS. To this end, we chose a natural parameter which is never less than the solution size, in particular the feedback vertex set number of the undirected graph underlying the given digraph. The problem we are interested in can formally be stated as follows.

Directed Feedback Vertex Set parameterized by FVS (DFVS[FVS])	
Instance:	A digraph D, an integer p, and a set F such that F is an UFVS of D.
Parameter:	$\lvert F\rvert$
Question:	Does there exist a vertex subset of size at most p that intersects every cycle in D?

Since every UFVS (undirected feedback vertex set) of D is also a DFVS (directed feedback vertex set) of D, we may assume without loss of generality that $p \leq |F|$ for every instance of DFVS[FVS]. Furthermore, we use k to denote the size of F. Our first result is a polynomial kernel for DFVS[FVS], formally stated below.

▶ **Theorem 1.** *There is a kernel with $\mathcal{O}(k^4)$ vertices for* DFVS[FVS].

The overall approach for proving Theorem 1 is inspired by the result of Bodlaender and Dijk [5] on kernelizing the undirected feedback vertex set problem. However, several obstacles needed to be overcome for the techniques to be applicable in the directed setting.

Interestingly, the existence of a polynomial kernel for DFVS parameterized by the solution size remains open even in the restricted setting of planar graphs. While our Theorem 1 naturally also provides a polynomial kernel for DFVS[FVS] on planar graphs, as our second main contribution we show that one can in fact obtain a significantly stronger result not only on planar graphs, but on all graphs embeddable on orientable surfaces.

▶ **Theorem 2.** *There is a kernel with $\mathcal{O}(k)$ vertices for* DFVS[FVS] *when the input digraph is embeddable on a surface of constant genus.*

We note that the existence of such a linear kernel can also be obtained from the Meta-Kernelization framework [4] by observing that DFVS[FVS] has finite integer index. However, the framework is non-constructive; unlike our Theorem 2, it does not provide a concrete kernelization algorithm for the problem.

Some proofs have been omitted due to space limitation.

2 Preliminaries

Graphs and Digraphs. We consider undirected and directed graphs that may contain self-loops and multiple edges and mostly use standard notation that can be found, for instance, in the textbook by Diestel [15]. For an undirected or directed graph D we denote by $V(D)$ its vertex set and by $E(D)$ its edgeset (or arcset). For a set of vertices $A \subseteq V(D)$, we denote by $D \setminus A$ the (di-)graph obtained from D after deleting all vertices in A as well as all arcs/edges incident to some vertex in V. Moreover, $D[A]$ denotes the graph $D \setminus (V(D) \setminus A)$. We say that a vertex u is a *neighbor* of a vertex v in D if $\{u, v\} \in E(D)$ (in the case that D is undirected) or at least one of $(u, v) \in E(D)$ or $(v, u) \in E(D)$ holds (in the case that D is directed). We denote by $N_D(v)$ (or by $N(D)$ if D is clear from the context) the set of all neighbors of v in D and refer to $|N_D(v)|$ as the *degree* of v in D (if D is undirected) or as the *total degree* of v in D (if D is directed). For a set of vertices A, we denote by $N_D(A)$ the set $(\bigcup_{a \in A} N_D(a)) \setminus A$.

In addition to the above, we also use standard notions such as *in-neighbors* and *out-neighbors*, *paths* and *directed paths* as well as *endpoints* and *internal vertices* of such paths, *contractions*, minors and others. We denote by $N^-_D(u)$ and $N^+_D(u)$ the set of all in-neighbors and out-neighbors of v in D, respectively; once again, we drop the subscript D if it can be inferred from the context and for a vertex set $A \subseteq V(D)$ we write $N^-_D(A)$ and $N^+_D(A)$ to denote the sets $(\bigcup_{a \in A} N^-_D(a)) \setminus A$ and $(\bigcup_{a \in A} N^+_D(a)) \setminus A$, respectively. We denote by $\overline{D}$ the undirected graph obtained from D after replacing every arc $(u, v) \in E(D)$ with an edge $\{u, v\}$; if there are arcs in both directions between u and v, then we say that there is a *bidirectional arc* between u and v and $\overline{D}$ will contain two parallel u-v edges.

▶ **Proposition 3** ([22]). The orientable and nonorientable genus, denoted by γ and $\tilde{\gamma}$, of complete bipartite graphs is given by the following formulae:

$$\gamma(K_{m,n}) = \left\lceil \frac{(m-2)(n-2)}{4} \right\rceil, m,n \geq 2; \qquad \tilde{\gamma}(K_{m,n}) = \left\lceil \frac{(m-2)(n-2)}{2} \right\rceil, m,n \geq 2.$$

▶ **Corollary 4.** If G is a graph such that $\gamma(G) \leq g$ and $\tilde{\gamma}(G) \leq h$ for some constants g and h, then G does not contain $K_{3,4g+3}$ nor $K_{3,2h+3}$ as a minor.

For a more detailed treatment of topological graph theory the reader is referred to [22].

Parameterized Algorithms and Kernelization. For a detailed illustration of the following facts the reader is referred to [11, 18]. A *parameterized problem* is a language $\Pi \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet; the second component k of instances $(I,k) \in \Sigma^* \times \mathbb{N}$ is called the *parameter*. A parameterized problem Π is *fixed-parameter tractable* if it admits a *fixed-parameter algorithm*, which decides instances (I,k) of Π in time $f(k) \cdot |I|^{\mathcal{O}(1)}$ for some computable function f.

A *kernelization* for a parameterized problem Π is a polynomial-time algorithm that given any instance (I,k) returns an instance (I',k') such that $(I,k) \in \Pi$ if and only if $(I',k') \in \Pi$ and such that $|I'| + k' \leq f(k)$ for some computable function f. The function f is called the *size* of the kernelization, and we have a polynomial kernelization if $f(k)$ is polynomially bounded in k. It is known that a parameterized problem is fixed-parameter tractable if and only if it is decidable and has a kernelization. However, the kernels implied by this fact are usually of superpolynomial size.

A *reduction rule* is an algorithm that takes as input an instance $I = (D,p,F)$ of DFVS[FVS] and outputs an instance $I' = (D',p',F')$ of the same problem. We say that the reduction rule is *sound* if I is a *yes*-instance if and only if I' is a *yes*-instance. In order to describe our kernelization algorithm, we present a series of reduction rules. We prove the soundness of each reduction rule immediately after presenting its description, unless the soundness is obvious. The reduction rules we present will be executed in the order in which they appear. That is, if at any point we may apply Reduction Rule i as well as Reduction Rule j where $i < j$, we will execute Reduction Rule i.

3 A Polynomial Kernel for DFVS[FVS]

Note that every FVS of $\overline{D}$ is also a DFVS of D. Given an instance D of DFVS, our kernelization algorithm for DFVS parameterized by FVS first computes a 2-approximate FVS S of $\overline{D}$ (using for instance the algorithm given in [1]) and then uses S to reduce the instance in polynomial-time into an equivalent instance with at most $\mathcal{O}(|S|^4)$ vertices.

Hence in the following we will assume that D is a directed graph and S is a FVS of $\overline{D}$ of size k. Our first two reduction rules are sound because (a) neither sinks nor sources can appear on a directed cycle and (b) if a vertex v has exactly one in-neighbor u in D then every directed cycle containing v has to use the arc (u,v) (a symmetric statement holds for vertices with exactly one out-neighbor).

▶ **Reduction Rule 5.** Delete all sources and sinks from D.

▶ **Reduction Rule 6.** Let l be an arbitrary vertex in D.

- If $N^+(l) = \{p\}$, then we contract the arc (l,p) into a new vertex l^*.
- If $N^-(l) = \{p\}$, then we contract the arc (p,l) into a new vertex l^*.

After the exhaustive application of these two rules, we may assume without loss of generality that the digraph D has no sinks or sources, and that every vertex has at least 2 in-neighbors and at least 2 out-neighbors. We now state one of our main reduction rules.

▶ **Reduction Rule 7.** Let u and v be two (not necessarily distinct) vertices in S such that there are at least $k+1$ internally vertex-disjoint directed u-v paths in D. Then,

- if $u \neq v$ and $(u,v) \notin E(D)$, we add an arc from u to v to D, or
- if $u = v$, we remove u from D and decrease the parameter k by one.

Proof of soundness. Let $u, v \in S$ be as above and let D' be the digraph obtained from D after applying the reduction rule. If $u = v$ then clearly every DFVS for D of size at most k contains u, which shows the soundness of the reduction rule.

If on the other hand $u \neq v$, we will show that a set $S' \subseteq V(D) = V(D')$ of size at most k is a DFVS for D if and only if it is also a DFVS for D'. The backward direction is trivial because D is a subgraph of D'. For the forward direction let S' be a DFVS for D of size at most k and assume for a contradiction that S' is not a DFVS for D'. Then $D' \setminus S'$ contains a directed cycle C that contains the arc (u,v). Hence $D' \setminus S'$ and thus also $D \setminus S'$ contains a directed v-u path P. Moreover, since S' has size at most k and there are at least $k+1$ vertex-disjoint directed u-v paths in D, we conclude that there is a u-v path P' in $D \setminus S'$. But then $P \cup P'$ must contain a directed cycle, which is also a directed cycle in $D \setminus S'$, a contradiction to our assumption that S' is a DFVS of D. ◀

We will use Reduction Rule 7 to reduce the number of vertices in $D \setminus S$ that 'directly contribute' to (pairs of) vertices in S. We formalize this idea in the following definition.

▶ **Definition 8.** Let (u,v) be an ordered pair of vertices in S. If $u \neq v$, then we refer to (u,v) as a *potential arc* in $D[S]$ and if additionally $(u,v) \notin D$ then we refer to (u,v) as a *non-arc*. If on the other hand $u = v$, then we refer to (u,v) as a *self-loop*. We say that a vertex $v \in V(D) \setminus S$ *contributes* to a potential arc or self-loop (u,w), if $(u,v) \in E(D)$ and $(v,w) \in E(D)$.

After the exhaustive application of Reduction Rule 7, we have the following structural observation regarding the input.

▶ **Observation 9.** For every $u \in S$ there are at most k internally vertex-disjoint u-u paths in D; moreover for every two distinct vertices u and v in S with $(u,v) \notin E(D)$, there are at most k vertex disjoint u-v paths in D. As a result, for every non-arc or self-loop (u,v), there are at most k vertices that contribute to (u,v).

Since S has at most k vertices and at most $k(k-1)$ ordered pairs of vertices, Observation 9 implies the following.

▶ **Observation 10.** There are at most $k^2(k-1)$ vertices in $D \setminus S$ that contribute to some non-arc of $D[S]$. Moreover, there are at most k^2 vertices in $D \setminus S$ that contribute to some self-loop of $D[S]$.

Our next aim is to bound the number of vertices in $A = D \setminus S$ in terms of k. Towards achieving this we will distinguish these vertices in terms of their degree in $D \setminus S$. We therefore denote by A_0, A_1, A_2, and $A_{\geq 3}$ the sets of all vertices in A that have total degree 0, 1, 2, and at least 3, respectively, in $D \setminus S$.

3.1 Bounding A_0, A_1 and $A_{\geq 3}$.

Note that Observation 10 already provides a bound for the number of vertices in A_0 that contribute to some self-loop of $D[S]$. Hence, in order to bound A_0, it is sufficient to provide a bound for the remaining vertices, in the following denoted by A'_0, in A_0. In the following let v be a vertex in A'_0. Because of Rule 5 v must have at least one in-neighbor and one out-neighbor in S. Consequently, v contributes to at least one potential arc of $D[S]$.

▶ **Reduction Rule 11.** If v does not contribute to a non-arc of $D[S]$, then we remove v from D.

Proof of soundness. Let v be as above and let D' be the directed graph obtained from D after deleting v. We show that a set $S' \subseteq V(D)$ is a DFVS for D if and only if S' is a DFVS for D'. The forward direction of this claim is trivial because D' is a subgraph of D. Towards showing the backward direction let S' be a DFVS for D' and assume for a contradiction that S' is not a DFVS for D. Then there must exist a directed cycle C in $D \setminus S'$ that contains v as well as two arcs (s, v) and (v, s') for some $s, s' \in S$. Because v does not contribute to a self-loop of $D[S]$, we have that $s \neq s'$. Because v does not contribute to a non-arc of $D[S]$, it follows that $(s, s') \in E(D)$. Hence the arc (s, s') together with the directed path from s' to s contained in C forms a directed cycle in $D' \setminus S'$, a contradiction to our assumption that S' is a DFVS for D'. ◀

After the exhaustive application of the above rule, we obtain that v contributes to some non-arc of $D[S]$ and hence together with Observation 10, we obtain the following.

▶ **Observation 12.** There are at most $k^2(k-1)$ vertices in A'_0.

Bounding A_1. We now present the reduction rules we use to bound the size of A_1, i.e., the number of leaves in A. Again it is sufficient to bound the number of vertices in A_1 that do not contribute to some self-loop of $D[S]$, in the following denoted by A'_1. Namely, we will introduce reduction rule that ensure that every vertex in A'_1 contributes to at least one non-arc in $D[S]$. Together with Observation 10 this then bounds the size of A'_1. Recall that at this point every vertex in D has at least two in-neighbors and at least two out-neighbors and since moreover every vertex in A'_1 does not contribute to a self-loop, we obtain that every vertex in A'_1 has at least one in-neighbor and at least one out-neighbor in S that are distinct. Hence we obtain:

▶ **Observation 13.** Every vertex in A'_1 has at least one in-neighbor and at least one out-neighbor in S and hence every vertex in A'_1 contributes to a potential arc of $D[S]$.

The next reduction rule reduces leaves that do not contribute to a non-arc of $D[S]$.

▶ **Reduction Rule 14.** If $l \in A'_1$ does not contribute to a non-arc of $D[S]$, then:
- if l is a source in $D \setminus S$, then we delete all arcs from l to vertices in S,
- if l is a sink in $D \setminus S$, then we delete all arcs from vertices in S to l.

Note that after application of Rule 14, l will only have either in-neighbors or out-neighbors in S and can hence be reduced further using Rule 6. Consequently, after the exhaustive application of the above rules, we conclude that every vertex in A'_1 contributes to at least one non-arc of $D[S]$. Due to Observation 10 we conclude that there are at most $k^2(k-1)$ vertices in A'_1. Finally, since $\overline{D \setminus S}$ is a forest and the number of vertices of degree at least 3 in a forest is at most equal to the number of leaves minus two, we get the following.

▶ **Observation 15.** There are at most $k^3 - 2$ vertices in $A_{\geq 3}$.

Note that at this point, we have bounded the size of the sets A_0, A_1 and $A_{\geq 3}$ and the only set that remains is A_2.

3.2 Bounding A_2

Our next aim is to bound the number of vertices in A_2.

▶ **Definition 16.** Let v be a vertex in A_2. We say that v is a *sink-vertex* or a *source-vertex* if the two arcs of $D \setminus S$ incident on it are both incoming arcs or outgoing arcs respectively. Otherwise we say that v is a *balanced-vertex*.

Note that due to Reduction Rule 6, there are no balanced vertices in $D \setminus S$ which have *no* neighbors in S. This is because otherwise, we would have already contracted one of the two arcs incident to v in $D \setminus S$.

Therefore, at this point, we infer the following.

▶ **Observation 17.** Every vertex in A_2 has at least one neighbor in S.

▶ **Definition 18.** Let $P = (v_1, \ldots, v_r)$ be a directed path of maximum length in $D \setminus S$ whose internal vertices are in A_2. Then we say that P is a path segment in $D \setminus S$. We say that P is an *outer path segment* if at least one of its endpoints is not in A_2, otherwise we say that P is an *inner path segment*.

Note that path segments are by definition *directed paths*. Our strategy now is to obtain a bound on the total number of path segments and then proceed to bound the length of each path segment. We first bound the number of outer path segments in $D \setminus S$ as follows. Let G be the undirected graph obtained from $\overline{D} \setminus S$ after contracting all edges which are incident to at least one vertex of degree 2 . Then the number of outer path segments in $D \setminus S$ is equal to two times the number of edges of G. Because G is a forest without degree two vertices it holds that the number of edges of G is equal to the number of leaves plus the number of non-leaves in G minus one. Hence the number of outer path segments is at most $2(|A'_1 \cup A_{\geq 3}| - 1)$, which together with the already obtained bound on these sets and Observation 15 allows us to infer the following.

▶ **Observation 19.** The number of outer path segments in $D \setminus S$ is at most $4k^2(k-1)+2k^2-6$.

In order to bound the number of inner path segments, we need to introduce a new reduction rule. We begin by defining the notion of a path segment 'contributing' to a potential arc.

▶ **Definition 20.** We say that a path segment $P = (v_1, \ldots, v_r)$ *contributes* to a potential arc (s, s') of $D[S]$ if there are i and j with $1 \leq i \leq j \leq r$ such that $(s, v_i) \in E(D)$ and $(v_j, s') \in E(D)$. Moreover, we say that P contributes to a self-loop if there are i and j with $1 \leq i \leq j \leq r$ such that $(s, v_i) \in E(D)$ and $(v_j, s) \in E(D)$ for some $s \in S$.

▶ **Reduction Rule 21.** If an inner path segment does not contribute to a non-arc or to a self-loop of $D[S]$, then we remove all internal vertices of P.

After the exhaustive application of the above rule, we obtain:

▶ **Observation 22.** Every inner path segment contributes to at least one non-arc or self-loop of $D[S]$.

Because every pair of inner path segments that contribute to some non-arc or self-loop (s, s') of $D[S]$ increase the number of disjoint paths between s and s' in D by at least one, Observation 9 implies that for every non-arc or self-loop (s, s') of $D[S]$ there are at most $2k$ inner path segments that contribute to (s, s'). Finally, because S has at most k vertices and at most $k(k-1)$ ordered pairs of vertices, we conclude that there are at most $2k^2(k-1) + 2k^2$ inner path segments in $D \setminus S$. Having obtained a bound on the number of inner path segments too, we conclude the following.

▶ **Observation 23.** The number of path segments in $D \setminus S$ is at most $6k^2(k-1) + 4k^2 - 6$.

Our next aim is to provide a bound on the overall length of path segments and use it to bound the size of A_2. Towards this aim we introduce reduction rules that allow us to bound the in-degree and the out-degree w.r.t. S of any vertex occurring internally in path segments.

▶ **Definition 24.** Let $s \in S$ and let $P = (v_1, \ldots, v_r)$ be an induced directed path in $D \setminus S$, whose internal vertices are in A_2 and that satisfies:

- $(s, v_1) \in E(D)$ and $(s, v_r) \in E(D)$ and v_1 is a balanced vertex in A_2,
- for every i with $1 < i < r$, it holds that $(s, v_i) \notin E(D)$.

If P satisfies the above properties we call P an *out-segment* for s. We say that P *contributes* to a potential arc or self-loop (s, s') in $D[S]$ if there is an index i with $1 \leq i < r$ such that $(v_i, s') \in E(D)$ for some $s' \in S$.

We now introduce a reduction rule that allows us to preprocess and reduce certain out-segments.

▶ **Reduction Rule 25.** Let $s \in S$ and let $P = (v_1, \ldots, v_r)$ be an out-segment for s. If P does not contribute to any non-arc or self-loop of $D[S]$, then we remove the arc (s, v_1).

After the exhaustive application of the above rule, we obtain the following.

▶ **Observation 26.** For each $s \in S$, there are at most k^2 out-segments for s.

Proof. Since Rule 25 does not apply, every out-segment for s contributes to at least one non-arc or self-loop of $D[S]$. Furthermore, every out-segment for s that contributes to some non-arc or self-loop (s, s') of $D[S]$ increases the number of internally vertex-disjoint paths s-s' paths in D by one. Observation 9 implies that for every non-arc or self-loop (s, s') of $D[S]$, there are at most k out-segments for s in $D \setminus (S \cup B)$ that contribute to (s, s').

Finally, because every vertex s is contained in at most a single self-loop and in at most $k-1$ non-arcs of $D[S]$, we infer that there are at most $k(k-1) + k$ out-segments for s. This completes the proof of the observation. ◀

We are now ready to bound the size of the set A_2.

▶ **Observation 27.** The number of vertices in A_2 is at most $12k^4 - 2k^3 - 12k$.

Proof. We begin by arguing that for every $s \in S$, s has at most $2k \cdot (6k^2(k-1) + 4k^2 - 6 + k(k-1) + k)$ neighbors in A_2. Since the number of out-neighbors of s in A_2 is at most the number of path segments (bounded by Observation 23) plus the number of out-segments for s (bounded by Observation 26), we obtain an upper bound of $6k^2(k-1) + 4k^2 - 6 + k^2$ on the size of the out-neighborhood (and by symmetry, the in-neighborhood) of s in A_2.

Consequently, the total number of neighbors of vertices in S to vertices in A_2 and thus (because of Observation 17) the total number of vertices in A_2 is at most $2k \cdot (6k^2(k-1) + 4k^2 - 6 + k^2)$. Finally, since the size of S is bounded by k, the observation follows. ◀

We are now ready to prove a bound on the size of the kernel. Recall that thus far we have obtained the following bounds:

- there are at most k^2 vertices in $D \setminus S$ contributing to a self-loop in $D[S]$ (Observation 10),
- $|A'_0| \leq k^2(k-1)$ (Observation 12),
- $|A'_1| \leq k^2(k-1)$ (see paragraph before Observation 15),
- $|A_2| \leq 12k^4 - 2k^3 - 12k$ (Observation 27),
- $|A_{\geq 3}| \leq k^3 - 2$ (Observation 15),

It follows that the total number of vertices in the reduced graph is at most $k^2 + |A'_0 \cup A'_1 \cup A_2 \cup A_{\geq 3} \cup S|$ which is at most $k^2 + 2k^2(k-1) + k^3 - 2 + 12k^4 - 2k^3 - 12k + k$, thus proving Theorem 1. This completes the description of our kernel for general instances of the problem; we now proceed to the linear kernel on graphs of bounded genus.

4 A Linear Kernel for DFVS[FVS] on Bounded Genus graphs

Throughout this section we will use D to denote a directed graph of genus at most some fixed bound g. We let S be a feedback vertex set of $\overline{D}$ and let c be a constant such that $\overline{D}$ is a $K_{3,c}$-minor-free graph, where c depends only on g as per Corollary 4. We begin with the following lemma, which follows directly from [21, Lemma 4.3].

▶ **Lemma 28.** *Let $G = (X, Y, E)$ be a bipartite graph and c a constant such that G is $K_{3,c}$-minor-free. Then,*

- *there are $\mathcal{O}(|X|)$ subsets $X' \subseteq X$ such that $X' = N(u)$ for some $u \in Y$ and*
- *for any subset $X' \subseteq X$ such that $|X'| \geq 3$, the set $Y' = \{y \in Y : N(y) \supseteq X'\}$ has size at most $c - 1$.*

We mainly use this lemma to bound the number of connected components of $\overline{D} \setminus S$, it gives directly a bound on the number of connected components with at least 3 neighbors in S, for the connected components with at most 2 neighbors in S, we need to introduce some new reduction rules. We will need a few additional notions to provide a concise presentation of the results in this section. A digraph H is called a *road* iff $\overline{H}$ is a path; the first and last vertex on a road are called its *endpoints*, and all other vertices on a road are called *internal* vertices. Moreover, for a directed graph G consider a connected component of $\overline{G}$ with vertex set A such that $G[A]$ is acyclic. Then $G[A]$ (and, equivalently, the set A) is called an *acyclic component* of G. Observe that there is a one-to-one correspondence between connected components of $\overline{D} \setminus S$ and acyclic components of $D \setminus S$.

For each distinct $x, y \in S$, we denote by $\mathcal{C}_{x,y}$ the set of all acyclic components C of $D \setminus S$ with $N(C) = \{x, y\}$. Finally, we use $\mathcal{C}^{\rightarrow}_{x,y}$ to denote the subset of $\mathcal{C}_{x,y}$ of components C with the property that $D[C \cup \{x, y\}]$:

- contains a directed path from x to y, but
- contains neither an x-x directed path nor a y-y directed path intersecting C.

Observe that any road within D that is disjoint from a feedback vertex set of $\overline{D}$ may contain exactly one arc between two adjacent vertices. Furthermore, in any instance where Reduction Rule 5 is not applicable, the input digraph D itself does not contain an acyclic component. That is, every connected component of $\overline{D}$ contains a directed cycle.

▶ **Observation 29.** If Reduction Rule 5 is not applicable and C is an acyclic component of $D \setminus S$, then there is a directed $N(C)$-$N(C)$ path in $D[C \cup N(C)]$ containing at least one vertex of C.

Crucially, Observation 29 implies that for each component C in $\mathcal{C}_{x,y}$, either $D[C \cup \{x\}]$ (or $D[C \cup \{y\}]$ by symmetry) contains a cycle, or $C \in \mathcal{C}^{\rightarrow}_{x,y} \cup \mathcal{C}^{\rightarrow}_{y,x}$.

▶ **Reduction Rule 30.** If C is an acyclic component of $D \setminus S$ where $N_{\overline{D}}(C) = \{x\}$ for some $x \in S$ and $C \cup \{x\}$ contains a cycle, then we remove $D[C \cup \{x\}]$ from D and reduce k by 1.

The soundness of the above rule follows from the fact that any DFVS which does not contain x must necessarily intersect C, and hence there also exists a DFVS of at most the same size which contains x but does not intersect C.

By expanding the above argument, we observe that if there exists a DFVS T containing at least two vertices from $\mathcal{C}_{x,y} \cup \{x, y\}$, then the set $T' = (T \setminus \mathcal{C}_{x,y}) \cup \{x, y\}$ is also clearly a solution of at most the same size as T. Hence every minimum DFVS contains at most two vertices from $\mathcal{C}_{x,y} \cup \{x, y\}$. Consequently, if we have a minimum DFVS T and $C_1, C_2, C_3 \in \mathcal{C}_{x,y}$, then at least one of C_1, C_2 or C_3 has an empty intersection with T. The soundness of the following three Reduction Rules follows.

▶ **Reduction Rule 31.** If $\mathcal{C}_{x,y}$ contains at least 3 acyclic components C_1, C_2, C_3 such that $D[C_1 \cup x]$, $D[C_2 \cup x]$ and $D[C_3 \cup x]$ each contains a cycle, we remove x and decrease k by 1.

▶ **Reduction Rule 32.** If $\mathcal{C}^{\rightarrow}_{x,y} \cap \mathcal{C}^{\rightarrow}_{y,x}$ contains at least 4 components, then we remove all components of $\mathcal{C}^{\rightarrow}_{x,y} \cup \mathcal{C}^{\rightarrow}_{y,x}$ from D and add the arcs (x, y) and (y, x) to D.

▶ **Reduction Rule 33.** If $\mathcal{C}^{\rightarrow}_{x,y}$ contains at least 3 components and $\mathcal{C}^{\rightarrow}_{x,y} \setminus \mathcal{C}^{\rightarrow}_{y,x}$ is not empty, then we remove all components of $\mathcal{C}^{\rightarrow}_{x,y} \setminus \mathcal{C}^{\rightarrow}_{y,x}$ from D and add the arc (x, y) to D.

▶ **Lemma 34.** *After applying Reduction Rules 5 to 6 and Reduction Rules 30 to 33, the resulting digraph is also $K_{3,c}$-minor-free.*

Proof. Since Reduction Rule 5, 30 and 31 only remove vertices, it clearly does not affect the fact that D is $K_{3,c}$-minor-free. Furthermore, the operations in Reduction Rule 6, 32 and 33 result in a graph D' such that $\overline{D'}$ is a minor of $\overline{D}$. Hence, the graph resulting from these reduction rules is also $K_{3,c}$-minor-free. ◀

We now argue the main structural consequence of applying these reduction rules.

▶ **Lemma 35.** *Suppose that Reduction Rule 5 and Reduction Rules 30 to 33 do not apply. Then $D \setminus S$ has $\mathcal{O}(|S|)$ acyclic components.*

▶ **Lemma 36.** *Let C be a connected component of $\overline{D} \setminus S$, and ℓ be the number of neighbors of C in S. If Reduction Rules 5 and 6 are not applicable, then $\overline{D}[C]$ has $\mathcal{O}(\ell)$ leaves.*

Proof. Recall that when the first two reduction rules do not apply, every vertex in D is incident to at least 4 arcs and thus every leaf of C is incident to at least 3 arcs with endpoints in S. However, since there can be bidirectional arcs between C and S, every leaf of C has at least 2 neighbors in S. From Lemma 28 it follows that there are only $\mathcal{O}(\ell)$ vertices in C with at least 3 neighbors in S. Hence it suffices to obtain an $\mathcal{O}(\ell)$ bound on the leaves of C with exactly two neighbors in S.

Recalling Lemma 28, we observe that each of the leaves of C has one of $\mathcal{O}(\ell)$ possible neighborhoods in S. Let us fix two distinct vertices x and y in $N(C)$. We will show that there are at most $c - 1$ leaves of C with both x and y as neighbors. Suppose for a contradiction that there is a set L of at least c leaves of $\overline{D}[C]$ which are adjacent to x and y. Since $\overline{D}[C]$ is a tree and L is a subset of its leaves, the graph $\overline{D}[C \setminus L]$ is also a tree. If we contract $\overline{D}[C \setminus L]$ into a single vertex, say z, then the subgraph induced on the vertices in $L \cup \{x, y, z\}$ would be isomorphic to $K_{3,c}$, contradicting the fact that $\overline{D}$ is $K_{3,c}$-minor-free. We conclude that C can have at most $c - 1$ leaves with neighbors x and y, completing the proof. ◀

▶ **Lemma 37.** *If none of Reduction Rules 5, 6, 30-33 apply, then $\overline{D} \setminus S$ has at most $\mathcal{O}(|S|)$ vertices of degree at least 3.*

Proof. Let $\mathcal{C}$ be the set of all components of $D \setminus S$. Since none of the aforementioned reduction rules apply, we can invoke Lemma 35 and Lemma 36. That is, we conclude that there are only $\mathcal{O}(|S|)$ components in $\mathcal{C}$ and that the number of leaves in a component $C \in \mathcal{C}$ is $\mathcal{O}(|N(C)|)$.

Since the number of leaves in a tree gives an upper bound on the number of vertices of degree at least 3, this implies that the number of vertices of degree at least 3 in $\overline{D}[C]$ is also bounded by $\mathcal{O}(|N(C)|)$. Thus it suffices to show that $\sum_{C \in \mathcal{C}} |N(C)| = \mathcal{O}(|S|)$. However, $\sum_{C \in \mathcal{C}} |N(C)|$ is the same as the number of edges in the graph G which we obtain from $\overline{D}$ by contracting each component of $\mathcal{C}$ to a single vertex and removing all edges between vertices in S. Since G is clearly a minor of $\overline{D}$, it is also $K_{3,c}$-minor-free and hence $|E(G)|$ is at most $\mathcal{O}(|V(G)|) = \mathcal{O}(|S| + |\mathcal{C}|) = \mathcal{O}(|S|)$, which concludes the proof. ◀

The main consequence of the above lemma is that we can now add all the vertices of degree at least 3 to the set S in order to get a set S' which is also a feedback vertex set of $\overline{D}$ of size $\mathcal{O}(|S|) = \mathcal{O}(k)$. At the same time, the graph $\overline{D} \setminus S'$ is significantly more structured: every connected component of this graph is in fact a path and this will play a crucial role in the rest of this section. Since $|S'| \in \mathcal{O}(|S|)$, it suffices to obtain a reduced instance of size linear in $|S'|$, and so for ease of presentation we will hereinafter set $S := S'$.

▶ **Reduction Rule 38.** If none of the Reduction Rules 5-6, 30-33 apply, then we add all vertices of total degree three in $D \setminus S$ to S.

Observe that after applying Reduction Rule 38, $D \setminus S$ is a set of roads. Furthermore, once we ensure that $D \setminus S$ is a set of roads, none of the reduction rules in this section will ever create a new degree 3 vertex in $D \setminus S$. Hence we can exhaustively apply all the reduction rules in this section once again to ensure that the number of roads in $D \setminus S$ is $\mathcal{O}(|S|)$. In the rest of the section, we present reduction rules to handle the roads in $D \setminus S$.

4.1 Dealing with roads

Our first step will be to transform our instance so that all roads in $D \setminus S$ are even more structured with respect to their adjacencies with S.

▶ **Definition 39.** A road P in $D \setminus S$ is *nice* if $|N(P') \cap S| \leq 2$, where P' are the internal vertices of P.

In other words, nice roads are roads whose internal vertices are *all* adjacent to at most two specific vertices from S (other than the endpoints of the road); observe that this is equivalent to requiring that $|N(P')| \leq 4$, where P' is the set of internal vertices of the nice road P.

In order to achieve this transformation, we will iteratively construct an auxiliary vertex set Q to store certain vertices that form separators between nice road segments in $D - S$. In the course of this procedure, we will also construct an injective mapping δ from Q to the connected components of $D \setminus (S \cup Q)$. We initialize by setting $\delta = Q = \emptyset$.

▶ **Reduction Rule 40.** Let A be a connected component which is a road in $D \setminus (S \cup Q)$ that is not nice. Moreover, let A' be a maximal nice subroad of A which contains a leaf in $\overline{D} \setminus (S \cup Q)$ and let a' be the unique neighbor of A' in A. Then add a' to Q and add $a' \mapsto A'$ to δ.

For each vertex $q \in Q$, let $R_q = \{q\} \cup \delta(q)$. Observe that R_q is a road which contains at least 3 neighbors in S. Furthermore, for any $q, q' \in Q$ our construction of δ ensures that R_q and $R_{q'}$ are vertex-disjoint since $\delta(q)$ is a nice road in $D \setminus (S \cup Q)$, for all $q \in Q$.

▶ **Lemma 41.** *After the exhaustive application of Reduction Rule 40, we have* $|Q| = \mathcal{O}(|S|)$.

The next rule is only applied *once* after the exhaustive application of Reduction Rule 40. Note that it does not increase the parameter by more than a linear factor due to Lemma 41.

▶ **Reduction Rule 42.** Set $S := S \cup Q$.

Observe that after the exhaustive application of Reduction Rule 40 and the application of Reduction Rule 42, each road in $D - S$ is nice. Furthermore, the number of roads in $D - S$ is still linear in S, since removing $|Q|$ vertices from a set of roads only increases the number of roads in the set by at most $|Q|$. Our next task is to deal with nice roads, but we first state a useful observation about general roads.

▶ **Observation 43.** Let P be a road in $D \setminus S$ and let P' be the internal vertices of P. For any DFVS T of D, the set $(T \setminus P') \cup N(P')$ is also a DFVS of D. In particular, every minimum DFVS contains at most $|N(P')|$ vertices of $P \cup N(P')$.

▶ **Reduction Rule 44.** Let P be a nice road in $D \setminus S$, P' internal vertices of P, and x a vertex in $N(P') \setminus V(P)$. If $D[P' \cup \{x\}]$ contains at least $|N(P')|$ directed cycles intersecting only in x, then we remove x from D and set $k = k - 1$.

For internal vertices of a road P, we define an equivalence relation $\sim_P$ such as $a \sim_P b$ if and only if $N^+(a) \setminus V(P) = N^+(b) \setminus V(P)$ and $N^-(a) \setminus V(P) = N^-(b) \setminus V(P)$ (i.e., a and b have same out- and in- neighborhoods outside of P). We are now ready to state our final reduction rule, which will later allow us to bound the length of each nice road by a constant.

▶ **Reduction Rule 45.** Let P be a nice road in $D \setminus S$, and let P' be internal vertices of P with $\ell = |N(P')|$. If P' contains a directed subpath $Q = (q_1, \ldots, q_{\ell+2})$, such that $q_i \sim_P q_j$ for all $1 \leq i, j \leq \ell + 1$, then we remove q_ℓ from D and add the arc $(q_{\ell-1}, q_{\ell+1})$.

We are now ready to complete the proof of our linear kernelization by bounding the size of an instance after the exhaustive application of our reduction rules.

▶ **Lemma 46.** *If none of Reduction Rules 5, 6 and 30-45 apply, then* $|D| = \mathcal{O}(|S|)$.

Proof Sketch. Let $\mathcal{P}$ be the set of all acyclic component of $D \setminus S$. Recall that since Reduction Rules 5, 6 and 30-42 do not apply, every acyclic component in $\mathcal{P}$ is a nice road in D and there are at most $|\mathcal{P}| = \mathcal{O}(|S|)$ such nice roads. Therefore, it suffices to show that there is a constant d which bounds the size of any nice road P in $\mathcal{P}$. This is achieved by the following three-step process: (a) we bound the number of vertices with bidirectional arcs to (from) a vertex from S, (b) we bound the number of sinks and sources on P, and (c) we bound the number of remaining vertices in P, i.e., vertices on directed path segments between vertices covered under points (a) and (b).

It is not difficult to show that points (a) and (b) follow from the exhaustive application of Reduction Rule 44. For point (c), we first observe that each vertex on a directed path must have precisely one in- and one out-neighbor in S, which means that there are only 2 'types' of vertices on these directed path segments. Then Reduction Rule 44 allows us to bound the number of alternations of types on a directed path, while Reduction Rule 45 provides a bound on the number of consecutive types on such paths. ◀

5 Conclusions and Future Work

Our results provide a stepping stone towards resolving the existence of a polynomial kernel for DFVS, and to the best of our knowledge also represent the first explicit kernelization results for DFVS with respect to any natural parameter. They also open up several new directions for future research. For instance, can we find reasonable parameters that lie "between" DFVS number and FVS number, and would it be possible to generalize our polynomial kernel to these? What about parameters which are incomparable to the FVS number but also upper-bound the DFVS number? Can our linear kernel be lifted to graph classes of bounded expansion or nowhere dense graphs? Can Theorem 1 be improved to a cubic or quadratic kernel, for instance by using techniques similar to the improvement obtained for the undirected setting by Thomasse [36]? Another related problem of interest is whether DFVS can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, which remains open even on planar graphs.

References

1 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.*, 12(3):289–297, 1999.

2 Reuven Bar-Yehuda, Dan Geiger, Joseph Naor, and Ron M. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and bayesian inference. *SIAM J. Comput.*, 27(4):942–959, 1998.

3 Ann Becker and Dan Geiger. Optimization of pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artif. Intell.*, 83(1):167–188, 1996.

4 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016.

5 Hans L. Bodlaender and Thomas C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theory Comput. Syst.*, 46(3):566–597, 2010.

6 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set new measure and new structures. In Haim Kaplan, editor, *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*, volume 6139 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2010.

7 Chandra Chekuri and Vivek Madan. Constant factor approximation for subset feedback set problems via a new LP relaxation. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 808–820. SIAM, 2016.

8 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008. doi:10.1016/j.jcss.2008.05.002.

9 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008. doi:10.1145/1411509.1411511.

10 Rajesh Hemant Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Transactions on Algorithms*, 11(4):28, 2015.

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

12 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory Comput. Syst.*, 54(1):73–82, 2014.

13 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159, 2011.

14 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM J. Discrete Math.*, 27(1):290–309, 2013. doi:10.1137/110843071.

15 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer Verlag, New York, 2nd edition, 2000.

16 Rodney G. Downey and Michael R. Fellows. Fixed-parameter intractability. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference, Boston, Massachusetts, USA, June 22-25, 1992*, pages 36–49, 1992.

17 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

18 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

19 P Erdős and L Pósa. On independent circuits contained in a graph. *Canad. J. Math*, 17:347–352, 1965.

20 Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

21 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. *J. Comput. Syst. Sci.*, 84:219–242, 2017.

22 Jonathan L. Gross and Thomas W. Tucker. *Topological Graph Theory*. Wiley-Interscience, New York, NY, USA, 1987.

23 Venkatesan Guruswami and Euiwoong Lee. Inapproximability of h-transversal/packing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, volume 40 of *LIPIcs*, pages 284–304. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

24 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013.

25 Naonori Kakimura, Ken-ichi Kawarabayashi, and Yusuke Kobayashi. Erdös-pósa property and its algorithmic applications: parity constraints, subset feedback set, and subset packing. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1726–1736. SIAM, 2012.

26 Naonori Kakimura, Ken-ichi Kawarabayashi, and Dániel Marx. Packing cycles through prescribed vertices. *J. Comb. Theory, Ser. B*, 101(5):378–381, 2011.

27 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pages 85–103, 1972. URL: http://www.cs.berkeley.edu/~luca/cs172/karp.pdf.

28 Ken-ichi Kawarabayashi and Yusuke Kobayashi. Fixed-parameter tractability for the subset feedback set problem and the s-cycle packing problem. *J. Comb. Theory, Ser. B*, 102(4):1020–1034, 2012. doi:10.1016/j.jctb.2011.12.001.

29 Ken-ichi Kawarabayashi, Daniel Král', Marek Krcál, and Stephan Kreutzer. Packing directed cycles through a specified vertex set. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 365–377, 2013.

30 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014. doi:10.1016/j.ipl.2014.05.001.

31 M. Pontecorvi and Paul Wollan. Disjoint cycles intersecting a set of vertices. *J. Comb. Theory, Ser. B*, 102(5):1134–1141, 2012.

32 Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Transactions on Algorithms*, 2(3):403–415, 2006. doi:10.1145/1159892.1159898.

33 Bruce A. Reed, Neil Robertson, Paul D. Seymour, and Robin Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996.

34 Paul D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.

35 Paul D. Seymour. Packing circuits in eulerian digraphs. *Combinatorica*, 16(2):223–231, 1996.

36 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010.

37 Magnus Wahlström. Half-integrality, LP-branching and FPT algorithms. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1762–1781. SIAM, 2014.

Timed Network Games*

Guy Avni[1], Shibashis Guha[2], and Orna Kupferman[3]

1 **IST Austria, Klosterneuburg, Austria**
2 **Hebrew University of Jerusalem, Israel**
3 **Hebrew University of Jerusalem, Israel**

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 vertex. The cost of traversing an edge depends on the 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 defining 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, the traversal of the network involves an inherent delay, and so sharing and congestion of resources crucially depends on time.

We study *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. In addition, each edge has a guard, describing time intervals in which the edge can be traversed, forcing the players to spend time on vertices. Unlike earlier work that add a time component to network games, the time in our model is continuous and cannot be discretized. In particular, players have uncountably many strategies, and a game may have uncountably many pure Nash equilibria. We study properties of timed network games with cost-sharing or congestion cost functions: their stability, equilibrium inefficiency, and complexity. In particular, we show that the answer to the question whether we can restrict attention to boundary strategies, namely ones in which edges are traversed only at the boundaries of guards, is mixed.

1998 ACM Subject Classification I.2.11 Distributed Artificial Intelligence – Multiagent systems, G.2.2 Graph Theory – network problems, F.2 Analysis of Algorithm and Problem Complexity

Keywords and phrases Network Games, Timed Automata, Nash Equilibrium, Equilibrium Inefficiency

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.37

1 Introduction

Network games (NGs, for short) [9, 37, 38] 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 of a player is the sum of costs of the edges her path traverses, where a cost of an edge depends on the *load* on it, namely the number of players using the edge. We distinguish between two types of costs. In *cost-sharing* games (a.k.a. network

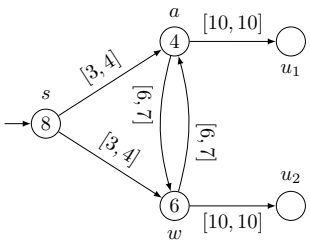
* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013, ERC grant no 278410) and the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award).

formation games), each edge has a cost and the players that use it split the cost among them, thus the load has a positive effect on cost. For example, when the costs correspond to prices, users that share a resource also share its price. Then, in *congestion* games, the 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 given this load. For example, when the network models a road system and costs correspond to the traversal time, an increased load on an edge corresponds to a traffic jam, increasing the cost of the players that use it.

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 communication systems, congestion only affects cars or messages that use a road or a channel simultaneously. We are interested in settings in which congestion affects the QoS or the way a price is shared (rather than the travel time). For example, discomfort increases in a crowded train or price is shared by the passengers in a taxi without affecting the travel time. Similarly, in mobile networks, the call quality depends on the number of subscribers using the network simultaneously. As a third example, when processing a task in a production system, jobs move from one station to another. The way the cost of running the stations is shared by the jobs that use it depends on the time spent in the stations and on the synchronization among the jobs.

We introduce and study *timed network games* (TNGs, for short) – a new model that adds a time component on NGs. Similar to NGs, the game is played on a network and the players need to find a path from their source to target vertices. Rather than paying for the traversal of edges, in TNGs the players pay for spending time in vertices. Each edge in the network has a *guard*, which is a disjunction of time intervals that specifies when an edge can be traversed. Traversing an edge is done instantaneously. So, a strategy for a player is a *timed path*: a sequence of pairs $\langle v, t\rangle$ of a vertex v and the time t spent on v. When the path traverses an edge in the network, the guard of the edge must be satisfied. For an integer $k \in \mathbb{N}$, let $[k] = \{1, \ldots, k\}$. Each vertex v has a cost function $r_v : [k] \to \mathbb{R}_{\geq 0}$ that assigns the cost of using v for one time unit, given the load in v. A *profile* in a TNG is then a vector of timed paths, namely the strategies of all players. Given a profile P, the cost of each player is induced by the cost functions of the vertices visited in her timed path, the time spent at each vertex, and the load on the vertices during these visits.

▶ **Example 1.** Consider an automobile service center with three stations: (s) tuning engine, (a) tire and air check, and (w) dry and wet wash. The costs for operating the stations per one time unit are 8, 4 and 6 respectively, and they are independent of the number of cars served. Accordingly, cost is shared by the users. There are two billing counters, u_1 and u_2, for dropped-in and registered cars. The setting is modeled by the TNG below. As described in the TNG, after spending some time in s, the cars can alternate between stations w and a. Assume that there are two players, and consider the profile P in which the first player chooses the timed path $(s, 3), (a, 7), u_1$ and the second player chooses the timed path $(s, 3), (a, 4), (w, 3), u_2$. Player 1's cost in P is $8/2 \cdot 3 + 4/2 \cdot 4 + 4 \cdot 3 = 32$ and Player 2's cost is $8/2 \cdot 3 + 4/2 \cdot 4 + 6 \cdot 3 = 38$. Another possible profile in this game is P', in which the strategies of the two players are $(s, 3), (w, 4), (a, 3), u_1$ and $(s, 3), (w, 7), u_2$. Now, the costs are 36 and 42, respectively.



There has been reference to *time* already in early work on *flow networks* [26]. Research spans from pioneering and theoretical work on flow networks in which congestion leads to queues (c.f., [41, 42]) to nowadays practical research on traffic engineering in software defined networks [2]. These works, however, do not address the problem from a game-theoretic perspective. To the best of our knowledge, the first works to consider network games with a time component are [36] and [28]. In [36], the focus is still on flow networks, and it enriches [41, 42] by viewing infinitesimal flow particles as selfish agents (see also [14]). Closer to our work, network games with time components where studied in [28, 31, 35]. These models differ from our model in two main aspects. First, the cost a player pays in these models is the time it takes to reach its destination, and our cost represents the QoS. Second, time is discrete in these models so the set of strategies the players choose is finite, whereas the source of the difficulty of our model is the real-time and the fact that the players have uncountably many strategies. The closest to our model is a model studied in [28], which studies a QoS pricing but using discrete time.

Our model of TNGs is the first to add real-time considerations to the strategies of the players. Indeed, a strategy for a player is not just the path of edges she is going to traverse, but also the time spent in vertices, which can be any number in $\mathbb{R}_{\geq 0}$. Thus, even if we restrict attention to simple paths, each player has uncountably many strategies. This continuous time and the richness of strategies that it brings with it is also a key difference between TNGs and NGs. Our model is inspired by *timed automata* [5]. There too, time is continuous, transitions between states are guarded by time constraints, and so is the time spent in a state. There are typically uncountably many runs of a timed automaton, corresponding to the uncountably many strategies a player typically has in our TNGs. The fact timed automata handle continuous time makes them the prominent formalism for specifying real-time on-going behaviors, and they are way more useful than formalisms in which time has been discretized (c.f., temporal logic with discrete clocks [23], or the fictitious-clock approach of [27]). We note that our TNGs correspond to a restricted class of timed automata, as our guards refer to the global time and cannot express, for example, a bound on the time spent in a vertex. In Section 7 we discuss the extension of our model to a richer one.

Note that, as in the *time-dependent cost* model of [28], load does not affect travel time and only affects the cost. Unlike [28], in TNGs time is continuous, which enables TNGs to model richer settings in practice. Note also that the cost function may model various applications. Consider, for example, a communication network with servers that encode or decode messages. A typical cost function for a server is the inverse of the quality of the signal, which is related to the number of bits needed to encode a message. Assuming that a server can handle a certain amount of data per unit time, this cost is the reciprocal of the number of bits used to encode a message. If the server allows a 16-bit encoding of a message when it serves less than 128 users simultaneously, and allows an 8-bit encoding when it serves between 128 and 256 users simultaneously, then the cost function maps x to $\frac{1}{16}$, for $x \leq 128$, and to $\frac{1}{8}$, for $129 \leq x \leq 256$, reflecting a better quality of the received message when load goes below 128 [33].

The first question that arises in the context of games is the existence of *stable outcomes* of the game. In the context of NGs, the most prominent stability concept is that of a (pure) *Nash equilibrium* (NE, for short) – a profile such that no player can decrease her cost by unilaterally deviating from her current strategy[1]. Decentralized decision-making may lead to solutions that are sub-optimal from the point of view of society as a whole. The standard measures to quantify the inefficiency incurred due to selfish behavior is the *price of stability* (PoS) [9] and the *price of anarchy* (PoA) [30]. In both measures we compare against a *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 an SO. In Example 1, profile P is an SO, and is also a (best) NE, while profile P' is a worst NE. Note that there can be uncountably many NEs in the TNG in Example 1. Indeed, for all $t \in [3,4]$, the profile P_t with the strategies $(s,3),(a,t),(w,4-t)(a,3)u_1$ and $(s,3),(a,t),(w,7-t)u_2$, is an NE with costs $8/2\cdot 3+4/2\cdot t+6/2\cdot(4-t)+4\cdot 3 = 36-t$ and $8/2\cdot 3+4/2\cdot t+6/2\cdot(4-t)+6\cdot 3 = 42-t$.

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* [37], thus every sequence of *best response* moves, namely moves that the players perform in order to reduce their costs, converges to an NE. For k-player cost-sharing NGs, the PoS and PoA are $\log k$ and k, respectively [9]. For congestion games with affine cost functions, PoS ≈ 1.577 [21, 3] and PoA $= \frac{5}{2}$ [22].

The fact a TNG has uncountably many profiles makes the adoption of results known for NGs questionable. Let us elaborate on this point. Consider a TNG $\mathcal{T}$, and a finite set $T \subseteq \mathbb{R}_{\geq 0}$ of time points. Note that there are only finitely many T-profiles in $\mathcal{T}$ (that is, profiles with T-strategies, in which all edges are taken at some time point in T). We show that once we restrict attention to T-profiles, we can construct an NG that is isomorphic to $\mathcal{T}$, in the sense that there is a cost-preserving bijection between profiles in the NG and T-profiles in the TNG $\mathcal{T}$. While this enables us to reduce questions about T-profiles in the TNG to questions on NGs, it is not clear to which finite set T we can restrict attention. In the setting of timed automata, much work has been done on obtaining decidability by partitioning $\mathbb{R}_{\geq 0}$ into finitely many *regions*. Essentially, all time points within a region are bisimilar, in the sense that the actions the automaton may take inside all time points in a region, coincide [5]. Our challenge here is similar: searching for a finite set of time points that partitions $\mathbb{R}_{\geq 0}$ to finitely many intervals.

Recall that the source for delays in TNGs are time guards on the edges, where each guard is a disjunction of intervals $[a,b]$, for $a \leq b \in \mathbb{Q}_{\geq 0}$. We refer to the two end points of all guards as *boundaries*. One can suspect that we can restrict attention to *boundary strategies*, namely timed paths that traverse edges only at boundary time points, and *boundary profiles* in which all the players choose boundary strategies. We show that the situation is mixed. The good news follows from choosing T above to be the boundaries, thus we show that a boundary NE and SO exist and an NE can be found by performing best-response moves that use only boundary strategies. Unfortunately, however, one cannot restrict attention to boundary profiles, as the best and worst NEs need not be boundary. We show a best and worst NE is attained in TNGs, which is not a-priori guaranteed.

In terms of inefficiency, the reduction from TNGs to isomorphic NGs enables us to extend upper bounds on the PoS and PoA from NGs to TNGs. The adoption of lower bounds

[1] Throughout this paper, we consider *pure* strategies, as is the case for the vast literature on cost-sharing games.

requires a reduction in the other direction – from NGs to TNGs, which we can show only for acyclic NGs. Consequently, we can apply only lower bounds known for acyclic NGs, which forces us to either prove direct bounds or to tighten lower bounds known for NGs to acyclic NGs. All in all, we are able to show that the PoS and PoA coincide for NGs and TNGs, except for the lower bound on the PoS of congestion TNGs, which we leave open. Finally, in terms of computational complexity, we prove that the problem of finding an NE is PLS-complete [29] for TNGs, which coincides with the complexity bounds for NGs [24, 40]. Proving membership in PLS follows easily from the reduction from TNGs to NGs. Proving hardness is more complex. For congestion TNGs, we are able to rely on known hardness results for congestion NGs, as they apply already for acyclic congestion NGs [1]. For cost-sharing TNGs we need a similar reduction from acyclic cost-sharing NGs, whose precise complexity is an open problem. Accordingly, we first settle the latter problem and prove that finding an NE in acyclic cost-sharing NG is PLS-hard, which allows us to prove the hardness result for cost-sharing TNGs.

Due to lack of space, some proofs appear in the full version, which can be found in the authors' homepages.

2 Preliminaries

We describe a *(closed) time interval* by $[m_1, m_2]$, for $m_1, m_2 \in \mathbb{R}_{\geq 0}$. We refer to m_1 and m_2 as the *start* and the *end interval boundaries*, respectively. A *guard* is the constant *true* or a disjunction of time intervals. A point in time $t \in \mathbb{R}_{\geq 0}$ *satisfies* a guard g if g is *true* or g includes a disjunct $[m_1, m_2]$ such that $m_1 \leq t \leq m_2$.

A *timed network* (TN) is a tuple $\langle V, E, \{g_e\}_{e \in E}\rangle$, where V is a set of vertices, $E \subseteq V \times V$ is a set of directed edges, and for each edge $e \in E$, the guard g_e specifies the time intervals during which e may be traversed. A *timed network game* (TNG) is $\mathcal{T} = \langle k, V, E, \{g_e\}_{e \in E}, \{r_v\}_{v \in V}, \langle s_i, u_i\rangle_{i \in [k]}\rangle$, where k is the number of players; $\langle V, E, \{g_e\}_{e \in E}\rangle$ is a timed network; for $v \in V$, the cost function $r_v : [k] \to \mathbb{R}_{\geq 0}$ maps the load on vertex v, namely the number of players that simultaneously visit vertex v, to the cost each of them pays for staying in v for one time unit with this load; and for $i \in [k]$, the pair $\langle s_i, u_i\rangle \in V \times V$ describes the objective of Player i: choosing a timed path from s_i to u_i. A timed network game is *symmetric* if all the players have the same objective, i.e. the same source and target pair. We use $B(\mathcal{T})$ to denote the set of interval boundaries appearing in the guards of $\mathcal{T}$.

In order to satisfy her objective, Player i has to choose a path in $\mathcal{T}$ from s_i to u_i as well as the duration spent in each vertex in the path. Indeed, while edges are traversed instantaneously, the guards on the edges force the players to spend time on vertices. Each player then aims to minimize the cost of these stays. In order to formally define the strategies of the players and their costs, we first need some definitions.

A *timed path* in the TNG $\mathcal{T}$ is a sequence $\pi = \langle v_0, t_0\rangle, \ldots, \langle v_{n-1}, t_{n-1}\rangle, v_n \in (V \times \mathbb{R}_{\geq 0})^* \cdot V$, such that for all $0 \leq i < n$, we have that $\langle v_j, v_{j+1}\rangle \in E$; that is, $v_0, \ldots, v_n$ is a path in the graph $\langle V, E\rangle$. Intuitively, for all $0 \leq i < n$, we have that t_j describes the time spent in the vertex v_j before the path continues to v_{j+1}. Let $\tau_0 = t_0$ and $\tau_j = \tau_{j-1} + t_j$, for $0 < j < n$. Note that $\tau_j = \sum_{l=0}^{j} t_l$. Thus, τ_j is the time that has elapsed since the traversal of π starts and until π leaves the vertex v_j. We sometimes refer to π also as the sequence $\langle \tau_0, e_1\rangle, \ldots, \langle \tau_{n-1}, e_n\rangle \in (\mathbb{R}_{\geq 0} \times E)^*$, where for all $1 \leq j \leq n$, we have that $e_j = \langle v_{j-1}, v_j\rangle$ is the j-th edge in π and is taken at time τ_{j-1}. We say that the timed path π is *legal* if for all $0 \leq j < n$, we have that τ_j satisfies the guard $g_{e_{j+1}}$.

A *strategy* for a player with an objective $\langle s, u\rangle$ is a legal timed path $\pi = \langle v_0, t_0\rangle, \ldots, \langle v_{n-1}, t_{n-1}\rangle, v_n$ such that $v_0 = s$ and $v_n = u$. Consider a finite set $T \subseteq \mathbb{R}_{\geq 0}$

of time points. We say that the strategy π is a *T-strategy* if all edges in π are taken at times in T. Formally, for all $0 \leq j < n$, we have that $\tau_j \in T$. A *profile* is a tuple $P = \langle \pi_1, \ldots, \pi_k \rangle$ of strategies for the players. That is, for $1 \leq i \leq k$, we have that π_i is a strategy for Player i. A profile is a T-profile if all its strategies are T-strategies. Of special interest are *boundary* strategies and profiles, namely T-strategies and T-profiles for $T = B(\mathcal{T})$. Note that each profile P has a finite minimal set $T \subseteq \mathbb{R}_{\geq 0}$ such that P is a T-profile. We denote this set by T_P.

Given $T \subseteq \mathbb{R}_{\geq 0}$, let $t_{\max} = \max(T)$. Also, for $t \in T$ such that $t \neq t_{\max}$, let $next_T(t)$ be the time point $t' \in T$ such that $t < t'$ and there is no $t'' \in T$ such that $t < t'' < t'$. That is, $next_T(t)$ is the time point successor to t in T. We can partition the interval $[0, t_{\max}]$ to a set Υ of sub-intervals $[m_1, m_2]$ such that m_1 and m_2 are in $T \cup \{0\}$, and $m_2 = next_T(m_1)$. We refer to the sub-intervals in Υ as *periods*. When T is T_P for some profile P, then the set Υ is the coarsest partition of $[0, t_{\max}]$ into periods such that no player crosses an edge within each period. We denote this partition by Υ_P.

Consider a T-profile 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 the legal timed path that is the 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 $\tau_{j-1}^i \leq m_1 \leq m_2 \leq \tau_j^i$. Note that since P is a T-profile, then for each period $\gamma = [m_1, m_2] \in \Upsilon_P$, 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.

Recall that the cost function $r_v : [k] \longrightarrow \mathbb{R}_{\geq 0}$ maps the load of v to the cost of v per time unit. Accordingly, if $visits_P(i, \gamma) = v$, then the cost of Player i in P over the period γ is $cost_{\gamma,i}(P) = r_v(load_P(v, \gamma)) \cdot |\gamma|$. We distinguish between two types of cost functions. We say that in *uniform cost-sharing games* (CS-TNGs, for short), the players that visit a vertex share its cost equally. Formally, each vertex v is associated with a rate $b_v \in \mathbb{R}_{\geq 0}$, and for all $l \geq 1$, we have $r_v(l) = \frac{b_v}{l}$. Note that increasing the load in uniform cost-sharing games decreases the cost of the players. On the other hand, in *congestion games* (CON-TNGs, for short), the cost functions are non-decreasing, thus increasing the load also increases the cost for each player. The total cost of Player i in profile P is then $cost_i(P) = \sum_{\gamma \in \Upsilon_P} cost_{\gamma,i}(P)$. The cost of the profile P, denoted $cost(P)$, is the total cost incurred by all the players, i.e., $cost(P) = \sum_{i=1}^{k} cost_i(P)$.

Consider a TNG $\mathcal{T}$. 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 for Player i 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. In other words, for every player i and every strategy π that Player i can deviate to from her current strategy in P, it holds that $cost_i(P[i \leftarrow \pi]) \geq cost_i(P)$. The set of NEs of the game $\mathcal{T}$ is denoted by $NE(\mathcal{T})$.

A *social optimum* (SO) of a game $\mathcal{T}$ is a profile that attains the infimum cost over all profiles. We denote by $SO(\mathcal{T})$ the cost of an SO profile; i.e., $SO(\mathcal{T}) = \inf_P cost(P)$. Note that since a TNG may have infinitely many profiles, we should indeed take the infimum (rather than minimum) over all profiles, and thus, an SO profile may not exist. As we shall show, however, all TNGs have boundary SO profiles. An SO profile may be achieved by a centralized authority and need not be an NE. The following parameters measure the inefficiency caused as a result of the selfish interests of the players. First, the *price of stability* (PoS) [8] of a timed network game $\mathcal{T}$ is the ratio between the infimum cost of an NE and the cost of a social optimum of $\mathcal{T}$. That is, $\text{PoS}(\mathcal{T}) = inf_{P \in NE(\mathcal{T})} cost(P)/SO(\mathcal{T})$. Then, the

price of anarchy (PoA) [34] of $\mathcal{T}$ is the ratio between the supremum cost of an NE and the cost of a social optimum of $\mathcal{T}$. That is, $\text{PoA}(\mathcal{T}) = sup_{P \in NE(\mathcal{T})} cost(P)/SO(\mathcal{T})$. Note that here too, we have to use infimum and supremum rather than minimum and maximum, yet we are going to show that best and worst NEs are always attained. For a family $\mathcal{F}$ of games, we say that the PoA of $\mathcal{F}$ is *at most* x if for all games F in $\mathcal{F}$, we have $PoA(F) \leq x$ and is *at least* x, if there exists a game F in $\mathcal{F}$ such that $PoA(F) = x$, and similarly for PoS.

3 Reduction to and from Network Games

A *network game* (NG) is $\mathcal{N} = \langle k, V, E, \{l_e\}_{e \in E}, \langle s_i, u_i \rangle_{i \in [k]} \rangle$, and has a similar structure to a TNG. A strategy of a player $i \in [k]$ is a path from s_i to u_i. The cost function $l_e : [k] \to \mathbb{R}_{\geq 0}$ maps the load on edge e to the cost each player pays for using e. As is the case with TNGs, one can consider both cost-sharing (CS-NGs) and congestion (CON-NGs) network games. Consider a profile $P = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ in the game. Since all the costs are positive, we can restrict attention to strategies in which the paths chosen by the players are simple. Then, we can also ignore the order between the edges in the paths and assume that for all $i \in [k]$, we have that $\sigma_i \subseteq E$ is a set of edges that composes a path from s_i to u_i.[2] For an edge $e \in E$, we denote by $load_P(e)$, the number of players that use the edge e in P. Each player that uses e then pays $l_e(load_P(e))$, and the cost of Player i in P is $\sum_{e \in \sigma_i} l_e(load_P(e))$.

Given an NG $\mathcal{N}$, a TNG $\mathcal{T}$ and a finite set $T \subset \mathbb{R}_{\geq 0}$, we say that $\mathcal{N}$ and $\mathcal{T}$ are *isomorphic with respect to* T if $\mathcal{N}$ and $\mathcal{T}$ have the same number of players and there exists a 1-to-1 cost-preserving correspondence between the profiles in $\mathcal{N}$ and the T-profiles in $\mathcal{T}$. Formally, there exists a bijection f from the set of T-profiles in $\mathcal{T}$ and the profiles in $\mathcal{N}$ such that for every T-profile P in $\mathcal{T}$ and $i \in [k]$, the costs of Player i in P and $f(P)$ coincide.

NGs have been extensively studied. In this section, we show that once we fix a set $T \subseteq \mathbb{R}_{\geq 0}$ of time points, we can reduce a TNG $\mathcal{T}$ with edges taken only at time points in T to an NG. Formally, we have the following.

▶ **Theorem 2.** *Given a TNG $\mathcal{T}$ and a finite set $T \subseteq \mathbb{R}_{\geq 0}$, we can construct an NG $\mathcal{N}$ such that $\mathcal{N}$ and $\mathcal{T}$ are isomorphic with respect to T. The size of $\mathcal{N}$ is polynomial in the size of $\mathcal{T}$ and T, and it is constructed in polynomial time.*

Proof. In TNGs, cost is associated with vertices and the time is spent in them, whereas in NGs, cost is associated with the edges and there is no reference to time. Thus, the construction translates the cost of staying in vertices during time intervals induced by T to the cost of traversing edges.

Consider a TNG $\mathcal{T} = \langle k, V, E, \{r_v\}_{v \in V}, \{g_e\}_{e \in E}, (s_i, u_i)_{i \in [k]} \rangle$ and the given set T. We assume that $0 \in T$. We construct an NG $\mathcal{N} = \langle k, V', E', \{l_e\}_{e \in E'}, (\langle s_i, 0 \rangle, u_i)_{i \in [k]} \rangle$, where $V' \subseteq (V \times T) \cup \{u_i\}_{i \in [k]}$ and $E' \subseteq V' \times V'$ is defined as follows (See an example in the full version. For every vertex $v \in V$, we have the following edges in E'. Let $\tau_{\max} = \max(T)$.

1. For every $\tau \neq \tau_{\max} \in T$, let $\tau' = next_T(\tau)$. Then, the edge $e = ((v, \tau), (v, \tau'))$ is in E', corresponding to players staying in vertex v during the interval $[\tau, \tau']$. Accordingly, the cost of e is such that for every $m \in [k]$, we have $l_e(m) = r_v(m)(\tau' - \tau)$.
2. For every $v' \neq v$ with $(v, v') \in E$ and $\tau \in T$ such that τ satisfies $g_{\langle v, v' \rangle}$, we have an edge $e = ((v, \tau), (v', \tau))$ in E'. This edge corresponds to the edge (v, v') in E. Recall that the

[2] Note that the assumptions on each edge being visited at most once in strategies in NGs does not apply to TNGs. Indeed, there, a player may benefit from visiting a vertex multiple times (see Example 1).

cost of crossing an edge in a TNG is 0. Accordingly, the cost of e is such that for every $m \in [k]$, we have $l_e(m) = 0$.

3. If $v = u_i$ for some $i \in [k]$, then for all $\tau \in T$, we have an edge $e = ((v, \tau), v)$ in E', with $l_{e'}(m) = 0$ for every $m \geq 1$. In $\mathcal{N}$, the target vertex for Player i is u_i.

It is easy to see that the size of $\mathcal{N}$ is polynomial in $\mathcal{T}$ and T. In the full version, we prove that $\mathcal{N}$ and $\mathcal{T}$ are indeed isomorphic with respect to T. That is, we show a bijection f from the set of T-profiles in $\mathcal{T}$ and the profiles in $\mathcal{N}$ such that for every T-profile P in $\mathcal{T}$ and $i \in [k]$, the costs of Player i in P and $f(P)$ coincide. ◀

A reduction in the other direction, namely of NGs to TNGs, is not obvious, as the dynamic of TNGs requires a synchronization among all the traversals in each of the edges. We illustrate this in the full version of the paper. When, however, the NG is acyclic, we can use a topological ordering on the edges and synchronize the traversals. Intuitively, each edge in the NG induces a vertex in the TNG, and the guards are defined so that the vertex associated with the j-th edge in the topological order is visited during the period $[j-1, j]$. This can be easily forced by guarding the edges entering the vertex by $[j-1, j-1]$ and guarding these that leave it by $[j, j]$. See the full version for the proof.

▶ **Theorem 3.** *Given an acyclic NG $\mathcal{N}$, we can construct in polynomial time a TNG $\mathcal{T}$ that is isomorphic to $\mathcal{N}$ with respect to $B(\mathcal{T}) \cup \{0\}$. The size of $\mathcal{T}$ is polynomial in the size of $\mathcal{N}$.*

4 On Boundary Strategies and Profiles

Since a strategy for a player in a TNG is a timed path with time points in $\mathbb{R}_{\geq 0}$, then each player has uncountably many possible strategies, and hence it is possible to have uncountably many profiles. In NGs, a strategy is a non-timed path from the source to the target. Even there, in the non-timed setting, there may be infinitely many paths from the source to the target. It is easy to see, however, that every strategy that is a non-simple path is dominated by the strategy obtained by removing cycles, and thus one can restrict attention to the finitely many profiles that consist of strategies that are simple paths. Our goal in this section is to examine whether some similar restriction can be made in TNGs. Indeed, being able to restrict attention to finitely many profiles would simplify our understanding of TNGs and their analysis. A natural candidate is a restriction to boundary strategies, namely these in which all edges are taken at interval boundaries. We show that while a boundary NE exists in all TNGs, and that all TNGs have a boundary SO, there may be uncountably many NEs that are not boundary. Moreover, there are TNGs in which the best and worst NEs are not boundary.

We first need the following lemma.

▶ **Lemma 4.** *Consider a TNG $\mathcal{T}$ and a finite set $T \subset \mathbb{R}_{\geq 0}$ such that $B(\mathcal{T}) \subseteq T$. Let $\pi_1, \ldots, \pi_{k-1}$ be T-strategies of players $1, \ldots, k-1$ respectively. There exists a T-strategy π_k of Player k such that for every strategy π'_k of Player k that is not a T-strategy, we have $cost_k(\langle \pi_1, \ldots, \pi_{k-1}, \pi_k \rangle) \leq cost_k(\langle \pi_1, \ldots, \pi_{k-1}, \pi'_k \rangle)$.*

Intuitively, Lemma 4 states that if all players but one use boundary strategies, then a best strategy for the k-th player can also be a boundary one. It implies that when we want to prove that a certain boundary profile is an NE, we can restrict attention to deviations that use boundary strategies.

▶ **Theorem 5.** *All TNGs have a boundary NE. Moreover, from every profile P, there is a sequence of best-response moves that converges to an NE. When P is boundary, so is the obtained NE.*

Proof. Given a TNG $\mathcal{T}$, let $\mathcal{N}$ be an NG that is isomorphic to $\mathcal{T}$ with respect to $B(\mathcal{T})$. Let f be the bijection from the set of profiles in $\mathcal{N}$ to the set of $B(\mathcal{T})$-profiles in $\mathcal{T}$ such that for every profile P in $\mathcal{N}$ and $i \in [k]$, the costs of Player i in P and $f(P)$ coincide. By Theorem 2, such an NG and bijection f exist. By [8, 24, 37], all NGs have an NE. Consider an NE $P_{\mathcal{N}}$ in $\mathcal{N}$. In the full version we prove that $f(P_{\mathcal{N}})$ is an NE in $\mathcal{T}$.

For the second claim, the above considerations also imply that starting from a profile P, we can restrict attention to best-response moves in which edges are taken in time points in $T_P \cup B(\mathcal{T})$, and reach the desired NE. In particular, when P is boundary, so is the obtained NE. ◀

Recall that an SO profile attains the infimum cost over all profiles. We now show that an SO profile always exists, and in fact there always exists a boundary SO. We show that a boundary SO profile always exists. The idea is that if, in a profile P, an edge e is taken at a non-boundary time τ, then it is possible to obtain a profile P' in which e is taken at a boundary time and $cost(P') \leq cost(P)$. We formalize this intuition in the full version.

▶ **Theorem 6.** *All TNGs have a boundary SO.*

We proceed to show that there are non-trivial TNGs that have uncountably many NEs, which implies they also have uncountably many non-boundary NEs.

▶ **Theorem 7.** *There exist CS-TNGs and CON-TNGs that have uncountably many NEs.*

Proof. The CS-TNG from Example 1 has uncountably many NEs. In the full version, we present and analyze in detail a different CS-TNG with uncountably many NEs.

We continue to CON-TNGs. Consider the TNG appearing in Figure 1. The objectives of Players 1 and 2 is $\langle a, d\rangle$ and the objectives of Players 3 and 4 is $\langle b, d\rangle$. The cost functions are written in the vertices. For $y \in [0, 0.5]$, let P_y be the profile in which Players 1 and 2 traverse the edge (a, b), and Players 3 and 4 traverse the edge (b, c), all at time y. In the full version, we prove that for every $y \in [0, 0.5]$, the profile P_y is an NE. Since y can have any value in $[0, 0.5]$, we are done. ◀

Theorem 7 suggests that the values of a best and worst NEs should be defined by means of infimum and supremum, and may not be attained. In the full version we prove that best and worst NEs do exist. Essentially, it follows from the fact that our guards are closed intervals, implying that the time points in an NE should satisfy a system of inequalities with no strict inequalities. As bad news, we now show that while a boundary NE alway exists, the best and worst NEs need not be boundary.

▶ **Theorem 8.** *There exists a CS-TNG in which the best NE is not a boundary profile.*

Proof. Consider the two-player TNG $\mathcal{N}$ that is played on the network depicted in Figure 2. The objective of Player i is $\langle s, u_i\rangle$. Player 1 has two boundary strategies: A, in which she traverses the edge $\langle s, a\rangle$ at time 0, and B, in which she takes it at time 2. Note that the suffixes of the strategies are fixed, as Player 1 must traverse the edge $\langle a, u_1\rangle$ at time 3. Player 2 has three boundary strategies: Strategies A and B, in which she traverses edge $\langle s, a\rangle$ at time 0 and 2, respectively, and Strategy C, in which she traverses the edge $\langle s, b\rangle$ at time 2. Again, the suffixes of the strategies A and B are fixed. In the full version,

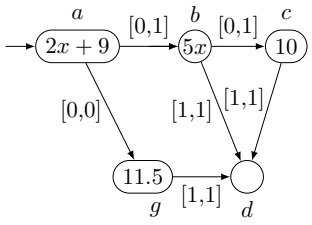


Figure 1 A CON-TNG in which the worst NE is not boundary.

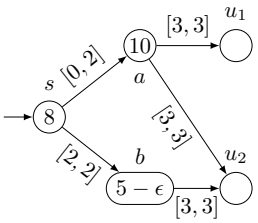


Figure 2 A CS-TNG in which the best NE is not boundary.

we prove that $\langle A, A\rangle$ and $\langle B, C\rangle$ are the only boundary NEs with $cost(\langle A, A\rangle) = 30$ and $cost(\langle B, C\rangle) = 31 - \epsilon$.

For $x \in (0, 2)$, let P_x be the profile in which both players traverse the edge $\langle s, a\rangle$ at time $2 - x$. In the full version, we show that we can define ϵ and x so that P_x is an NE with $cost(P_x) < \min\{cost(\langle A, A\rangle), cost(\langle B, C\rangle)\}$. For example, by taking $x = 0.25$ and $\epsilon = 0.5$ we get an NE with $cost(P_x) = 26.5$ ◀

▶ **Theorem 9.** *There exists a CS-TNG in which the worst NE is not a boundary profile.*

Proof. Consider the two-player TNG $\mathcal{N}$ that is played on the network depicted in Figure 3. The objective of Player i is $\langle s, u_i\rangle$. Player 1 has three boundary strategies: A, in which she traverses the edge (v_1, v_2) at time 0; B, in which she takes it at time 2; and D, in which she traverses the edge (v_1, v_5) at time 2.

Player 2 has four boundary strategies: A, in which she traverses edge (v_1, v_2) at time 0; B, where she takes (v_1, v_2) at time 2; C, where she traverses the edge (v_1, v_4) at time 2; and E, where she traverses the edge (s, v_3). Note that strategy E has a fixed cost of 13.2.

In the full version, we prove that the only boundary profile that is an NE is the profile $\langle D, C\rangle$, whose cost is 26.3, and that the non-boundary profile $P_{0.2}$ in which Players 1 and 2 traverse the edge (v_1, v_2) together at time 1.8 is an NE with cost 26.4, which is higher than $cost(\langle D, C\rangle)$. ◀

▶ **Theorem 10.** *There exists a CON-TNG in which the worst NE is not a boundary profile.*

Proof. Recall the CON-TNG presented in Figure 1. In the proof of Theorem 7, we proved that for all $0 \leq y \leq 0.5$, the profile P_y, in which Players 1 and 2 traverse the edge (a, b) and Players 3 and 4 traverse the edge (b, c), all at time y, is an NE. We have $cost_1(P_y) = cost_2(P_y) = 13y + 10 \cdot (1 - y) = 3y + 10$, whereas $cost_3(P_y) = cost_4(P_y) = 10y + 10 \cdot (1 - y) = 10$. Thus $cost(P_y) = 6y + 40$.

Players 1 and 2 have three boundary strategies: A, in which they traverse the edge (a, b) at time 0; B, in which they traverse the edge (a, b) at time 1; and C, in which they traverse the edge (a, g) at time 0. Players 3 and 4 have three boundary strategies: D, in which they

traverse the edge (b, c) at time 0, and E, in which they traverse the edge (b, c) at time 1, and F, in which they traverse the edge (b, d) at time 1.

In the full version, we show that the boundary NEs with the highest cost are $\langle C, B, E, E\rangle$ and $\langle C, B, F, F\rangle$ having a cost of 42.5. The cost of the profile $P_{0.5}$ is $6 \cdot 0.5 + 40 = 43$. This implies that the worst NE in the CON-TNG in Figure 1 is a non-boundary profile. ◀

We note that it might appear that whenever there exists a non-boundary NE in a TNG $\mathcal{T}$, there exist uncountably many NEs in $\mathcal{T}$. This, however, is not the case as can be seen in the TNG in Figure 3. As argued in the proof of Theorem 9, this TNG has only one non-boundary NE. We also note that while we showed that the best and worst NEs in a CS-TNG need not be boundary, for congestion games we only showed that the worst NE need not be boundary. Thus, the problem of whether there is a CON-TNG in which the best NE is not boundary is left open.

5 Equilibrium Inefficiency

As discussed in Section 1, decentralized decision-making often leads to solutions that are sub-optimal from the point of view of the society as a whole. Recall that the measures PoS and PoA measure the inefficiency caused by the selfish behavior of the players. It refers to the ratio between the best (PoS) and worst (PoA) NEs and the SO. In this section we discuss these measures for TNGs. For NGs, the PoS and PoA are well understood. In order to use Theorem 2 and apply the results known for NGs to TNGs, we need to find a set of time points with respect to which the models are isomorphic. As discussed in Section 4, the natural candidate for this is the set of interval boundaries. While, however, we can restrict attention to boundary strategies when we consider the SO, such a restriction is not sound when we consider the infimum and supremum values of NEs. We show that our results in Theorem 2 and Section 4 do imply the required upper bounds, and that the lower bounds known for NGs can be extended to TNGs by carefully revising the examples known there.

▶ **Theorem 11.** *The PoS and PoA for TNGs are upper-bounded by these for NGs. Thus, for CS-TNGs with k players, the PoS is at most $\log k$ and the PoA is at most k. For CON-TNGs with affine cost functions, the PoS is at most 1.577 and the PoA is at most $\frac{5}{2}$.*

Proof. Consider a TNG $\mathcal{T}$. Let P be an NE in $\mathcal{T}$ and let $\mathcal{N}_P$ be the NG isomorphic to $\mathcal{T}$ with respect to $B(\mathcal{T}) \cup T_P$. Let f be a cost preserving bijection from the $(B(\mathcal{T}) \cup T_P)$-profiles of $\mathcal{T}$ and these of $\mathcal{N}_P$. As argued in the proof of Theorem 5, the profile $f(P)$ is an NE in $\mathcal{N}_P$. It follows that the cost of an NE in $\mathcal{T}$ is upper and lower bounded by the cost of an NE in an NG. Also, by Theorem 6, there exists a boundary SO in $\mathcal{T}$, which, by Theorem 2, corresponds to an SO in $\mathcal{N}$. Thus, the ratio between an NE in $\mathcal{T}$ and the cost of its SO is upper and lower bounded by this ratio in an NG. Since the above holds for all TNGs, we are done. ◀

Adopting the lower bounds on PoS and PoA from NGs to TNGs is more difficult, as the reduction from NGs to TNGs can be applied only to acyclic NGs. Fortunately, for CS-NGs, matching lower bounds have been proven for acyclic networks. Hence, using considerations that are similar to these in the proof of Theorem 11 (in fact, simpler ones, as there is no need to refer to T_P), we can use the reduction described in Theorem 3 in order to conclude the following.

▶ **Theorem 12.** *The PoS and PoA for TNGs are lower-bounded by these for acyclic NGs. Thus, for CS-TNGs with k players, the PoS is at least $\log k$ and the PoA is at least k.*

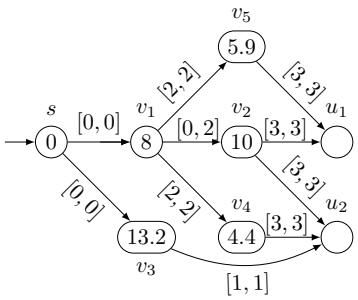


Figure 3 A CS-TNG in which the worst NE is not boundary.

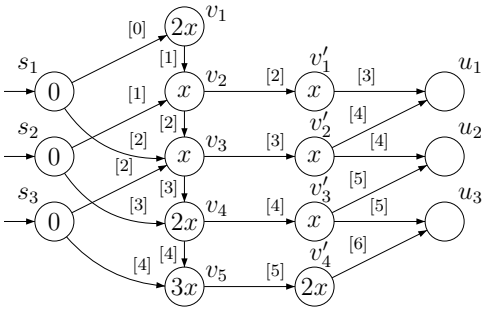


Figure 4 A lower bound of PoA $= \frac{5 \cdot k}{2(k-2)+3+5}$ for CON-TNGs.

For CON-TNG, the adoption of results from CON-NGs is more challenging, as known lower bounds use cyclic network. We are still able to prove a lower bound for the PoA. A bound for CON-NGs with linear cost function has been shown in [21]. In our case, we show that the upper bound is matched asymptotically

▶ **Theorem 13.** *There are CON-TNGs with linear cost functions such that for k=3 or more players, the PoA is* $\frac{5 \cdot k}{2(k-2)+3+5}$. *Hence as* $k \to \infty$, *the PoA approaches* $\frac{5}{2}$.

Proof. Consider the three-player CON-TNG appearing in Figure 4. The sources and targets of the three players are s_1, s_2, s_3 and u_1, u_2, u_3, respectively. The cost of staying in the source vertices is 0. For the rest of the vertices, the cost functions are as follows: $r_{v_1}(x) = r_{v_4}(x) = 2x$, $r_{v_2}(x) = r_{v_3}(x) = x$, $r_{v_5}(x) = 3x$, $r_{v'_1}(x) = r_{v'_2}(x) = r_{v'_3}(x) = x$, and $r_{v'_4}(x) = 2x$.

Consider the profile P in which Player i, for all $i \in \{1, 2, 3\}$, visits the vertices v_i, v_{i+1}, v'_i. The profile P is an NE in which each player pays 5, so $cost(P) = 15$. However, an SO is obtained when each Player i moves from her source to target through vertices v_{i+2}, v'_{i+1}. In this profile, the costs of the players are $2, 3$, and 5. Thus PoA $= \frac{15}{2+3+5} = \frac{3}{2}$.

If there are k players, we consider the game with vertices $v_1, \ldots, v_{k+2}$ and vertices $v'_1, \ldots, v'_{k+1}$. The cost functions are $r_{v_1}(x) = r_{v_{k+1}}(x) = r_{v'_{k+1}}(x) = 2x$, $r_{v_{k+2}}(x) = 3x$, while for the remaining vertices v apart from the source and the target vertices, $r_v(x) = x$. The PoA is $\frac{5 \cdot k}{2(k-2)+3+5}$. Hence, PoA asymptotically reaches its upper bound as k tends to ∞. ◀

6 The Complexity of Finding an NE

The complexity class PLS contains local search problems with polynomial time searchable neighborhoods [29]. Essentially, a problem is in PLS if there is a set of feasible solutions for it such that it is possible to find, in polynomial time, an initial feasible solution and then

iteratively improve it, with each improvement being performed in polynomial time, until a local optimum is reached. See the full version for the formal definition.

In this section we prove that the problem of finding an NE is PLS-complete for TNGs, which coincides with the complexity bounds for NGs [24, 40]. Proving membership in PLS would follow easily from the reduction to NGs. Proving hardness is more involved: While for CON-TNGs we are able to rely on previous results, corresponding to CS-TNGs, we first solve the problem for acyclic CS-NGs. We start with the upper bound.

▶ **Theorem 14.** *The problem of finding an NE in CS-TNGs and CON-TNGs is in PLS. For symmetric TNGs, the problem can be solved in polynomial time.*

Proof. For membership in PLS, we describe an algorithm to find an NE. Consider a TNG $\mathcal{T}$, and let $\mathcal{N}$ be the isomorphic NG with respect to $B(\mathcal{T})$. Recall that the size of $\mathcal{N}$ is polynomial in the size of $\mathcal{T}$. We run the PLS algorithm for finding an NE P in $\mathcal{N}$. As in Theorem 5, the profile $f^{-1}(P)$ is an NE in $\mathcal{T}$, thus we are done. When $\mathcal{T}$ is symmetric, so is $\mathcal{N}$. Since finding an NE in a symmetric NG can be done in polynomial time [24], the claim follows. ◀

For PLS-hardness, we describe a reduction from the problem of finding a local MAX CUT in a weighted network (LMC, for short) which is known to be PLS-complete [39]. In [1], a polynomial-time reduction is shown from the LMC problem to the problem of finding an NE in CON-NGs. The reduction involves two steps: from the LMC problem to the problem of finding an NE in a class of games called *quadratic threshold games*, which in turn is reduced to the problem of finding an NE in a CON-NG. The reduction in [1] always produces an acyclic CON-NG. By Theorem 3, the latter can be reduced to an isomorphic CON-TNG. In order to use a similar technique for CS-TNGs, we first establish PLS-hardness for acyclic CS-NGs, which is an open problem. The proof uses a non-trivial reduction from the LMC problem and can be found in the full version.

▶ **Theorem 15.** *The problem of finding an NE in acyclic CS-NGs is PLS-hard.*

We thus have a matching lower bound also for CS-TNGs leading to the following theorem.

▶ **Theorem 16.** *The problems of finding an NE in CS-TNGs and CON-TNGs are PLS-complete.*

7 Discussion and Directions for Future Research

We introduced and studied timed network games, which are an extension of network games with real-time considerations. TNGs are inspired by *timed automata* [5], which are automata extended by a finite set of *clocks*. A clock is a variable that takes values in $\mathbb{R}_{\geq 0}$ and whose values increase as time passes. In the full version we study *TNGs with clocks*, in which, as in timed automata, transitions are labeled by constraints on the clocks and clocks may be *reset* when traversing a transition. For example, if we reset a clock x when we enter a vertex v, then a guard $x \leq 5$ in a transition that leaves v, bounds the time spent in v to be at most 5 time units. The TNGs we study here are equivalent to a model with clocks that are never reset. Indeed, then, all clocks maintain the time that has passed since the start of the game, and guards impose bounds on this time. TNGs with clocks are already interesting in the degenerate case when there is only one player, a.k.a. *priced timed automata* (PTA, for short) [7, 13].

We describe here briefly our results for TNGs with clocks. Clearly, the negative results we obtain here for TNGs without clocks follow to the general setting. Recall that a main tool for obtaining positive results is a reduction between TNGs and NGs. The key to such a reduction is a partition of $\mathbb{R}_{\geq 0}$ into finitely many intervals, which involves two questions: about the granularity to which we have to partition $\mathbb{R}_{\geq 0}$, and about the maximal point in time that is of interest. While the answer to the first question is not difficult also for TNGs with clocks, the answer to the second question is difficult and interesting in its own right. Our positive results are not obtained using such a reduction. In order to prove the existence of an NE in every TNG with clocks, we show that such games are potential games and we also find a lower bound on the decrease in potential in a best response. Note that only showing that TNGs with clocks are potential games does not suffice to prove existence of an NE as there are infinitely many profiles. We then turn to study computational-complexity problems and show that the best-response problem is PSPACE-complete, which matches the complexity of *cost optimal reachability* in PTAs [15]. Finally, we address the question above; namely, we find bounds on the minimal time at which the players reach their destinations in an NE and an SO.

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 [6, 19], studies of equilibria in games related to synthesis and repair problems [18, 17, 25, 4], and of non-zero-sum games in formal verification [20, 16]. This line of work also includes an extension of NGs to objectives that are richer than reachability [12], NGs in which the players select their paths dynamically [11], and efficient reasoning about NGs with huge networks [32, 10].

Additional extensions of TNGs that we plan to study are the following: (1) Richer objectives, where the vertices of the TNG are labeled by letters from an alphabet, allowing objectives that describe on-going behaviors [12]. For example, an objective may require each visit to vertex labeled *send* to be preceded by a vertex labeled *encode*. (2) A *dynamic* choice of paths, where strategies do not specify the full path but rather map prefixes of paths of all players to the next move [11]. For example, when the network models a network of roads and the players are drivers, it makes sense to allow drivers to observe the congestion in the network when reaching a junction (vertex) before choosing the next road (edge) in their path. (3) A *global-cost* mechanism, in which the load on a resource refers to the total time for which it is used, rather than to particular time instants.

References

1. H. Ackermann, H. Röglin, and B. Vöcking. On the impact of combinatorial structure on congestion games. *J. ACM*, 55(6):25:1–25:22, December 2008.
2. S. Agarwal, M. S. Kodialam, and T. V. Lakshman. Traffic engineering in software defined networks. In *Proc. 32nd IEEE International Conference on Computer Communications*, pages 2211–2219, 2013.
3. 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.
4. 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.
5. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–236, 1994.
6. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

7 R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. *Theoretical Computer Science*, 318(3):297–322, June 2004.

8 E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *Proc. 45th IEEE Symp. on Foundations of Computer Science*, pages 295–304. IEEE Computer Society, 2004.

9 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.

10 G. Avni, S. Guha, and O. Kupferman. An abstraction-refinement methodology for reasoning about network games. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia*, 2017.

11 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.

12 G. Avni, O. Kupferman, and T. Tamir. Network-formation games with regular objectives. *Information and Computation*, 251:165–178, 2016.

13 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. pages 147–161, London, UK, 2001. Springer-Verlag.

14 U. Bhaskar, L. Fleischer, and E. Anshelevich. A stackelberg strategy for routing flow over time. In *Proc. of SODA*, pages 192–201, 2011.

15 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, October 2007.

16 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.

17 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.

18 K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Games with secure equilibria. *Theoretical Computer Science*, 365(1-2):67–82, 2006.

19 K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In *Proc. 18th Int. Conf. on Concurrency Theory*, pages 59–73, 2007.

20 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.

21 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.

22 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.

23 E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Proc. 2nd Int. Conf. on Computer Aided Verification*, volume 531 of *Lecture Notes in Computer Science*, pages 136–145. Springer, 1990.

24 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.

25 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.

26 L.R. Ford and D.R. Fulkerson. *Flows in networks.* Princeton Univ. Press, Princeton, 1962.

27 T.A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proc. 19th Int. Colloq. on Automata, Languages, and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 545–558. Springer, 1992.

28 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.

29 D. S. Johnson, C. H. Papadimtriou, and M. Yannakakis. How easy is local search? *Journal of Computer and Systems Science*, 37(1):79–100, August 1988.

30 E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.

31 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.

32 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*, Lecture Notes in Computer Science. Springer, 2017.

33 B. P. Lathi and R. Green. *Essentials of digital signal processing.* Cambridge University Press, 2014.

34 C. H. Papadimitriou. Algorithms, games, and the internet. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 749–753, 2001.

35 M. Penn, M. Polukarov, and M. Tennenholtz. Random order congestion games. *Mathematics of Operations Research*, 34(3):706–725, 2009.

36 K. Ronald and S. Martin. Nash equilibria and the price of anarchy for flows over time. *Theoretical Computer Science*, 49(1):71–97, 2011.

37 R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

38 T. Roughgarden and E. Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002.

39 A. A. Schäffer and M. Yannakakis. Simple local search problems that are hard to solve. *SIAM J. Comput.*, 20(1):56–87, 1991.

40 V. Syrgkanis. The complexity of equilibria in cost sharing games. In *WINE*, volume 6484 of *Lecture Notes in Computer Science*, pages 366–377. Springer, 2010.

41 W. S. Vickrey. Congestion theory and transport investment. *The American Economic Review*, 59(2):251–260, 1969.

42 S. Yagar. Dynamic traffic assignment by individual path minimization and queuing. *Transportation Research*, 5(3):179–196, 1971.

Efficient Identity Testing and Polynomial Factorization in Nonassociative Free Rings

Vikraman Arvind[1], Rajit Datta[2], Partha Mukhopadhyay[3], and S. Raja[4]

1 **Institute of Mathematical Sciences (HBNI), Chennai, India**
arvind@imsc.res.in
2 **Chennai Mathematical Institute, Chennai, India**
rajit@cmi.ac.in
3 **Chennai Mathematical Institute, Chennai, India**
partham@cmi.ac.in
4 **Chennai Mathematical Institute, Chennai, India**
sraja@cmi.ac.in

Abstract

In this paper we study arithmetic computations in the nonassociative, and noncommutative free polynomial ring $\mathbb{F}\{x_1, x_2, \ldots, x_n\}$. Prior to this work, nonassociative arithmetic computation was considered by Hrubes, Wigderson, and Yehudayoff [7], and they showed lower bounds and proved completeness results. We consider Polynomial Identity Testing (PIT) and polynomial factorization over $\mathbb{F}\{x_1, x_2, \ldots, x_n\}$ and show the following results.

1. Given an arithmetic circuit C of size s computing a polynomial $f \in \mathbb{F}\{x_1, x_2, \ldots, x_n\}$ of degree d, we give a deterministic $\text{poly}(n, s, d)$ algorithm to decide if f is identically zero polynomial or not. Our result is obtained by a suitable adaptation of the PIT algorithm of Raz-Shpilka[13] for noncommutative ABPs.
2. Given an arithmetic circuit C of size s computing a polynomial $f \in \mathbb{F}\{x_1, x_2, \ldots, x_n\}$ of degree d, we give an efficient deterministic algorithm to compute circuits for the irreducible factors of f in time $\text{poly}(n, s, d)$ when $\mathbb{F} = \mathbb{Q}$. Over finite fields of characteristic p, our algorithm runs in time $\text{poly}(n, s, d, p)$.

1998 ACM Subject Classification F.2.1 Computations on Polynomials

Keywords and phrases Circuits, Nonassociative, Noncommutative, Polynomial Identity Testing, Factorization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.38

1 Introduction

Noncommutative computation, introduced in complexity theory by Hyafil [8] and Nisan [12], is an important subfield of algebraic complexity theory. The main algebraic structure of interest is the free noncommutative ring $\mathbb{F}\langle X\rangle$ over a field $\mathbb{F}$, where $X = \{x_1, x_2, \cdots, x_n\}$ is a set of free noncommuting variables. A central problem is Polynomial Identity Testing which may be stated as follows:

Let $f \in \mathbb{F}\langle X\rangle$ be a polynomial represented by a noncommutative arithmetic circuit C. The circuit C can either be given by a black box (using which we can evaluate C on matrices with entries from $\mathbb{F}$ or an extension field), or the circuit may be explicitly given. The algorithmic problem is to check if the polynomial computed by C is identically zero. We recall the formal definition of a noncommutative arithmetic circuit.

▶ **Definition 1.** An *arithmetic circuit* C over a field $\mathbb{F}$ and indeterminates $X = \{x_1, x_2, \cdots, x_n\}$ is a directed acyclic graph (DAG) with each node of indegree zero labeled by a variable or a scalar constant from $\mathbb{F}$: the indegree 0 nodes are the input nodes of the circuit. Each internal node of the DAG is of indegree two and is labeled by either a $+$ or a $\times$ (indicating that it is a plus gate or multiply gate, respectively). Furthermore, the two inputs to each $\times$ gate are designated as left and right inputs which prescribes the order of multiplication at that gate. A gate of C is designated as *output*. Each internal gate computes a polynomial (by adding or multiplying its input polynomials), where the polynomial computed at an input node is just its label. The *polynomial computed* by the circuit is the polynomial computed at its output gate.

When the multiplication operation of the circuit in Definition 1 is *noncommutative*, it is called a *noncommutative arithmetic circuit* and it computes a polynomial in the free noncommutative ring $\mathbb{F}\langle X\rangle$. Since cancellation of terms is restricted by noncommutativity, intuitively it appears noncommutative polynomial identity testing would be easier than polynomial identity testing in the commutative case. This intuition is supported by fact that there is a deterministic polynomial-time white-box PIT algorithm for noncommutative ABP [13]. In the commutative setting a deterministic polynomial-time PIT for ABPs would be a major breakthrough.[1] However, there is little progress towards obtaining an efficient deterministic PIT for general noncommutative arithmetic circuits. For example, the problem is open even for noncommutative *skew* circuits.

If *associativity* is also dropped then it turns out that PIT becomes easy, as we show in this work. More precisely, we consider the free noncommutative and nonassociative ring of polynomials $\mathbb{F}\{X\}$, $X = \{x_1, x_2, \ldots, x_n\}$, where a polynomial is an $\mathbb{F}$-linear combination of monomials, and each monomial comes with a bracketing order of multiplication. For example, in the nonassociative ring $\mathbb{F}\{X\}$ the monomial $(x_1(x_2x_1))$ is different from monomial $((x_1x_2)x_1)$, although in the associative ring $\mathbb{F}\langle X\rangle$ they clearly coincide.

When the multiplication operation is both noncommutative and nonassociative, it is called a *nonassociative noncommutative circuit* and it computes a polynomial in the free nonassociative noncommutative ring $\mathbb{F}\{X\}$. Previously, the nonassociative arithmetic model of computation was considered by Hrubes, Wigderson, and Yehudayoff [7]. They showed completeness and explicit lower bound results for this model. We show the following result about PIT.

- Let $f(x_1, x_2, \ldots, x_n) \in \mathbb{F}\{X\}$ be a degree d polynomial given by an arithmetic circuit of size s. Then in deterministic $\mathrm{poly}(s, n, d)$ time we can test if f is an identically zero polynomial in $\mathbb{F}\{X\}$.

▶ Remark. We note that our algorithm in the above result does not depend on the choice of the field $\mathbb{F}$. A recent result of Lagarde et al. [11] shows an exponential lower bound, and a deterministic polynomial-time PIT algorithm over $\mathbb{R}$ for noncommutative circuits where all parse trees in the circuit are isomorphic. We also note that in [4] an exponential lower bound is shown for set-multilinear arithmetic circuits with the additional semantic constraint that each monomial has a unique parse tree in the circuit (but different monomials can have different parse trees).

Next, we consider polynomial factorization in the ring $\mathbb{F}\{\mathrm{X}\}$. Polynomial factorization is very well-studied in the commutative ring $\mathbb{F}[X]$: Given an arithmetic circuit C computing a multivariate polynomial $f \in \mathbb{F}[\mathrm{X}]$ of degree d, the problem is to efficiently compute circuits

[1] The situation is similar even in the lower bound case where Nisan proved that noncommutative determinant or permanent polynomial would require exponential-size algebraic branching program [12].

for the irreducible factors of f. A celebrated result of Kaltofen [9] solves the problem in randomized $\text{poly}(n, s, d)$ time. Whether there is a polynomial-time deterministic algorithm is an outstanding open problem. Recently, it is shown (for fields of small characteristic and characteristic zero) that the complexity of deterministic polynomial factorization problem and the PIT problem are polynomially equivalent [10]. A natural question is to determine the complexity of polynomial factorization in the noncommutative ring $\mathbb{F}\langle X\rangle$. The free noncommutative ring $\mathbb{F}\langle X\rangle$ is not even a *unique factorization domain* [6]. However, unique factorization holds for homogeneous polynomials in $\mathbb{F}\langle X\rangle$, and it is shown in [2] that for homogeneous polynomials given by noncommutative circuits, the unique factorization into irreducible factors can be computed in randomized polynomial time (essentially, by reduction to the noncommutative PIT problem).

In this paper, we note that the ring $\mathbb{F}\{X\}$ is a *unique factorization domain*, and given a polynomial in $\mathbb{F}\{X\}$ by a circuit, we show that circuits for all its irreducible factors can be computed in deterministic polynomial time.

- Let $f(x_1, x_2, \ldots, x_n) \in \mathbb{F}\{X\}$ be a degree d polynomial given by an arithmetic circuit of size s. Then if $\mathbb{F} = \mathbb{Q}$, in deterministic $\text{poly}(s, n, d)$ time we can output the circuits for the irreducible factors of f. If $\mathbb{F}$ is a finite field such that $char(\mathbb{F}) = p$, we obtain a deterministic $\text{poly}(s, n, d, p)$ time algorithm for computing circuits for the irreducible factors of f.

1.1 Outline of the proofs

Identity Testing Result. The main ideas for our algorithm are based on the white-box Raz-Shpilka PIT algorithm for noncommutative ABPs [13]. As in the Raz-Shpilka algorithm [13], if the circuit computes a nonzero polynomial $f \in \mathbb{F}\{X\}$, then our algorithm output a *certificate monomial* m such that coefficient of m in f is nonzero.

We first sketch the main steps of the Raz-Shpilka algorithm. The Raz-Shpilka algorithm processes the input ABP (assumed homogeneous) layer by layer. Suppose layer i of the ABP has w nodes. The algorithm maintains a spanning set $\mathbb{B}_i$ of at most w many linearly independent w-dimensional vectors of monomial coefficients. More precisely, each vector $v_m \in \mathbb{B}_i$ is the vector of coefficients of monomial m computed at each of the w nodes in layer i. Furthermore, the coefficient vector at layer i of any monomial is in the span of $\mathbb{B}_i$. The construction of $\mathbb{B}_{i+1}$ from $\mathbb{B}_i$ can be done efficiently. Clearly the identity testing problem can be solved by checking if there is a nonzero vector in $\mathbb{B}_d$, where d is the total number of layers.

Now we sketch our PIT algorithm for polynomials over $\mathbb{F}\{\mathrm{X}\}$ given by circuits. Let f be the input polynomial given by the circuit C.

We encode monomials in the free nonassociative noncommutative ring $\mathbb{F}\{\mathrm{X}\}$ as monomials in the free noncommutative ring $\mathbb{F}\langle \mathrm{X}, (,)\rangle$, such that the encoding preserves the multiplication structure of $\mathbb{F}\{\mathrm{X}\}$ (Observation 2). For $1 \leq j \leq d$, we can efficiently find from C a homogeneous circuit C_j that computes the degree j homogeneous part of C. Thus, it suffices to test if $C_j \equiv 0$ for each j. Hence, it suffices to consider the case when $f \in \mathbb{F}\{\mathrm{X}\}$ is homogeneous and C is a homogeneous circuit computing f.

For $j \leq d$ let G_j denote the set of degree j gates of C. The algorithm maintains a set $\mathbb{B}_j$ of $|G_j$-dimensional linearly independent vectors of monomial coefficients such that any degree j monomial's coefficient vector is in the linear span of $\mathbb{B}_j$. Clearly, $|\mathbb{B}_j| \leq |G_j|$. We compute $\mathbb{B}_{j+1}$ from the sets $\{\mathbb{B}_i : 1 \leq i \leq j\}$. For each vector in $\mathbb{B}_j$ we also keep the corresponding monomial. In the nonassociative model a degree d monomial $m = (m_1 m_2)$ is generated in a *unique way*. To check if the coefficient vector of m is in the span of $\mathbb{B}_d$ it suffices to consider

vectors in the spans of $\mathbb{B}_{d_1}$ and $\mathbb{B}_{d_2}$, where $d_1 = \deg(m_1)$ and $d_2 = \deg(m_2)$. This is a crucial difference from a general noncommutative circuit and using this property we can compute $\mathbb{B}_{j+1}$.

Polynomial Factorization in $\mathbb{F}\{X\}$. For a polynomial $f \in \mathbb{F}\{\mathrm{X}\}$, let f_j denote the homogeneous degree j part of f. For a monomial m, let $c_m(f)$ denote the coefficient of m in f. We will use the PIT algorithm as subroutine for the factoring algorithm. Arvind et al. [2] have shown that given a monomial m and a homogeneous noncommutative circuit C, in deterministic polynomial time circuits for the formal left and right derivatives of C with respect to m can be efficiently computed. This result is another ingredient in our algorithm.

We sketch the easy case, when the given polynomial f of degree d has no constant term. Applying our PIT algorithm to the homogeneous circuit C_d (computing f_d) we find a nonzero monomial $m = (m_1\ m_2)$ of degree d in f_d along with its coefficient $c_m(f)$. Notice that for any nontrivial factorization $f = gh$, m_1 is a nonzero monomial in g and m_2 is a nonzero monomial in h. Suppose $|m_1| = d_1$ and $|m_2| = d_2$. Then the left derivative of C_d with respect to m_1 gives $c_{m_1}(g)\ h_{d_2}$ and the right derivative of C_d with respect to m_2 gives $c_{m_2}(h)\ g_{d_1}$. We now use the circuits for these derivatives and the nonassociative structure, to find circuits for different homogeneous parts of g and h. The details, including the general case when f has a nonzero constant term, is in Section 4.

1.2 Organization

In Section 2 we describe some useful properties of nonassociative and noncommutative polynomials. In Section 3 we give the PIT algorithm for $\mathbb{F}\{X\}$. In Section 4 we describe the factorization algorithm for $\mathbb{F}\{X\}$. Finally, we list some open problems in Section 5.

2 Preliminaries

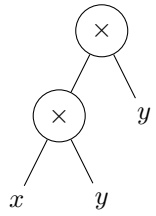
For an arithmetic circuit C, a *parse tree* for a monomial m is a multiplicative sub-circuit of C rooted at the output gate defined by the following process starting from the output gate:

- At each $+$ gate retain exactly one of its input gates.
- At each $\times$ gate retain both its input gates.
- Retain all inputs that are reached by this process.
- The resulting subcircuit is multiplicative and computes a monomial m (with some coefficient).

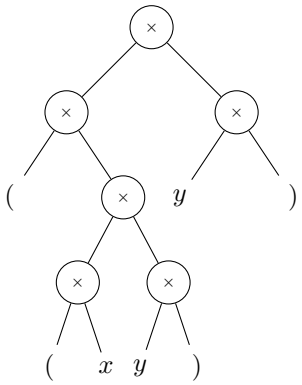
For arithmetic circuits C computing polynomials in the free nonassociative noncommutative ring $\mathbb{F}\{X\}$, the same definition for the parse tree of a monomial applies. As explained in the introduction, in this case each parse tree (generating some monomial) comes with a bracketed structure for the multiplication. It is convenient to consider a polynomial in $\mathbb{F}\{x_1, \ldots, x_n\}$ as an element in the noncommutative ring $\mathbb{F}\langle x_1, \ldots, x_n, (,)\rangle$ where we introduce two auxiliary variables (and) (for left and right bracketing) to encode the parse tree structure of any monomial. We illustrate the encoding by the following example.

Consider the monomial (which is essentially a binary tree with leaves labeled by variables) in the nonassociative ring $\mathbb{F}\{x, y\}$ shown in Figure 1a. Its encoding as a bracketed string in the free noncommutative ring $\mathbb{F}\langle x, y, (,)\rangle$ is ((x y) y) and its parse tree shown in Figure 1b.

Consider an arithmetic circuit C computing a polynomial $f \in \mathbb{F}\{X\}$. The circuit C can be efficiently transformed to a circuit $\tilde{C}$ that computes the corresponding polynomial $\tilde{f} \in \mathbb{F}\langle X, (,)\rangle$ by simply introducing the bracketing structure for each multiplication gate

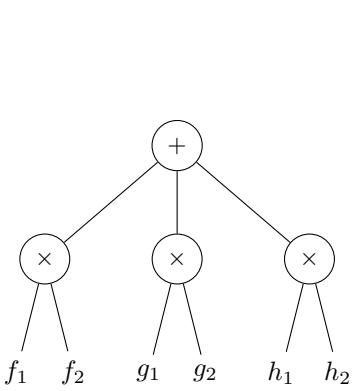


(a) A nonassociative and noncommutative monomial xyy.

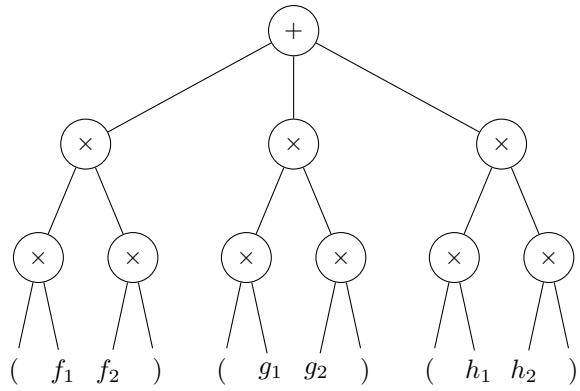


(b) Corresponding monomial $((xy)\ y) \in \mathbb{F}\langle X\rangle$.

Figure 1 Nonassociative & noncommutative monomial and its corresponding noncommutative bracketed monomial.



(a) C computing a nonassociative, noncommutative polynomial.



(b) $\tilde{C}$ that computes the corresponding noncommutative polynomial.

Figure 2 Nonassociative circuit and its corresponding noncommutative bracketed circuit.

of C in a bottom-up manner as indicated in the following example figures. Consider the circuits described in Figures 2a and 2b where f_i, g_i, h_i's are polynomials computed by subcircuits. Clearly the bracket variables preserve the parse tree structure. The following fact is immediate.

▶ **Observation 2.** *A nonassociative noncommutative circuit C computes a nonzero polynomial $f \in \mathbb{F}\{X\}$ if and only if the corresponding noncommutative circuit $\tilde{C}$ computes a nonzero polynomial $\tilde{f} \in \mathbb{F}\langle X, (,)\rangle$.*

We recall that the free noncommutative ring $\mathbb{F}\langle \mathrm{X}\rangle$ is not a unique factorization domain (UFD) [6] as shown by the following standard example : $xyx + x = x(yx + 1) = (xy + 1)x$. In contrast, the nonassociative free ring $\mathbb{F}\{\mathrm{X}\}$ is a UFD.

▶ **Proposition 3.** *Over any field $\mathbb{F}$, the ring $\mathbb{F}\{\mathrm{X}\}$ is a unique factorization domain. More precisely, any polynomial $f \in \mathbb{F}\{X\}$ can be expressed a product $f = g_1 g_2 \cdots g_r$ of irreducible polynomials $g_i \in \mathbb{F}\{X\}$. The factorization is unique upto constant factors and reordering.*

▶ Remark. Usually, even the ordering of the irreducible factors in the factorization is unique.

Exceptions arise because of the equality $(g+\alpha)(g+\beta) = (g+\beta)(g+\alpha)$ for any polynomial $g \in \mathbb{F}\{X\}$ and $\alpha, \beta \in \mathbb{F}$.

We shall indirectly see a proof of this proposition in Section 4 where we describe the algorithm for computing all irreducible factors.

Given a noncommutative circuit C computing a homogeneous polynomial in $\mathbb{F}\langle \mathrm{X} \rangle$ and a monomial m over X, one can talk of the left and right derivatives of C w.r.t m [2]. Let $f = \sum_{m'} c_{m'}(f)m'$ for some $f \in \mathbb{F}\langle \mathrm{X} \rangle$ and A be the subset of monomials m' of f that have m as prefix. Then the left derivative of f w.r.t. m is

$$\frac{\partial^{\ell} f}{\partial m} = \sum_{m' \in A} c_{m'}(f)m'',$$

where $m' = m \cdot m''$ for $m' \in A$. Similarly we can define the right derivative $\frac{\partial^{r} f}{\partial m}$. As shown in [2], if f is given by a circuit C then in deterministic polynomial time we can compute circuits for $\frac{\partial^{\ell} f}{\partial m}$ and $\frac{\partial^{r} f}{\partial m}$. We briefly discuss this in the following lemma.

▶ **Lemma 4.** [2] *Given a noncommutative circuit C of size s computing a homogeneous polynomial f of degree d in $\mathbb{F}\langle \mathrm{X} \rangle$ and monomial m, there is a deterministic* $\mathrm{poly}(n, d, s)$ *time algorithm that computes circuits $C_{m,\ell}$ and $C_{m,r}$ for the left and right derivatives $\frac{\partial^{\ell} C}{\partial m}$ and $\frac{\partial^{r} C}{\partial m}$, respectively.*

Proof. We explain only the left partial derivative case. Let m be a degree d' monomial and $f \in \mathbb{F}\langle X \rangle$ be a homogeneous degree d polynomial f computed by circuit C. In [2],a small substitution deterministic finite automaton A with $d'+2$ states is constructed that recognizes all length d strings with prefix m and substitutes 1 for prefix m. The transition matrices of this automaton can be represented by $(d'+2) \times (d'+2)$ matrices. From the evaluation of circuit C on these transition matrices will recover the circuit for $\frac{\partial^{\ell} C}{\partial m}$ in the $(1, d^{'}+1)^{th}$ entry of the output matrix. ◀

The left and right partial derivatives of inhomogeneous polynomials are similarly defined. The same matrix substitution works for non-homogeneous polynomials as well [2]. As discussed above, given a nonassociative arithmetic circuit C computing a polynomial $f \in \mathbb{F}\{\mathrm{X}\}$, we can transform C into a noncommutative circuit $\tilde{C}$ that computes a polynomial $\tilde{f} \in \mathbb{F}\langle X, (,) \rangle$. Suppose we want to compute the left partial derivative of f w.r.t. a monomial $m \in \mathbb{F}\{X\}$. Using the tree structure of m we transform it into a monomial $\tilde{m} \in \mathbb{F}\langle X, (,) \rangle$ and then we can apply Lemma 4 to $\tilde{C}$ and $\tilde{m}$ to compute the required left partial derivative. We can similarly compute the right partial derivative. We use this in Section 4.

We also note the following simple fact that the homogeneous parts of a polynomial $f \in \mathbb{F}\{X\}$ given by a circuit C can be computed efficiently. We can apply the above transformation to obtain circuit $\tilde{C}$ and use a standard lemma (see e.g., [14]) to compute the homogeneous parts of $\tilde{C}$.

▶ **Lemma 5.** *Given a noncommutative circuit C of size s computing a noncommutative polynomial f of degree d in $\mathbb{F}\langle \mathrm{X}, (,) \rangle$, one can compute homogeneous circuits C_j (where each gate computes a homogeneous polynomial) for j^{th} homogeneous part f_j of f, where $0 \leq j \leq d$, deterministically in time* $\mathrm{poly}(n, d, s)$.

3 Identity Testing in $\mathbb{F}\{X\}$

In this section we describe our identity testing algorithm.

▶ **Theorem 6.** *Let $f(x_1, x_2, \ldots, x_n) \in \mathbb{F}\{X\}$ be a degree d polynomial given by an arithmetic circuit of size s. Then in deterministic* $\mathrm{poly}(s, n, d)$ *time we can test if f is an identically zero polynomial in $\mathbb{F}\{X\}$.*

Proof. By Lemma 5 we can assume that the input is a homogeneous nonassociative circuit C computing some homogeneous degree d polynomial in $\mathbb{F}\{X\}$ (i.e. every gate in C computes a homogeneous polynomial). Also, all the $\times$ gates in C have fanin 2 and $+$ gates have unbounded fanin. We can assume the output gate is a $+$ gate. We can also assume w.l.o.g. that the $+$ and $\times$ gates alternate in each input gate to output gate path in the circuit (otherwise we introduce sum gates with fan-in 1).

The j^{th}-layer of circuit C to be the set of all $+$ gates in computing degree j homogeneous polynomials. Let s^+ be the total number of $+$ gates in C. To each monomial m we can associate a vector $v_m \in \mathbb{F}^{s^+}$ of coefficients, where v_m is indexed by the $+$ gates in C, and $v_m[g]$ is the coefficient of monomial m in the polynomial computed at the $+$ gate g. We can also write

$$v_m[g] = c_m(p_g),$$

where p_g is the polynomial computed at the sum gate g.

For the j^{th} layer of $+$ gates, we will maintain a maximal linearly independent set $\mathbb{B}_j$ of vectors v_m of monomials. These vectors correspond to degree j monomials. Although $v_m \in \mathbb{F}^{s^+}$, notice that $v_m[g] = 0$ at all $+$ gates that do not compute a degree j polynomial. Thus, $|\mathbb{B}_j|$ is bounded by the number of $+$ gates in the j^{th} layer. Hence, $|\mathbb{B}_j| \leq s$.

The sets $\mathbb{B}_j$ are computed inductively for increasing values of j. For the base case, the set $\mathbb{B}_1$ can be easily constructed by direct computation. Inductively, suppose the sets $\mathbb{B}_i : 1 \leq i \leq j-1$ are already constructed. We describe the construction of $\mathbb{B}_j$. Computing $\mathbb{B}_d$ and checking if there is a nonzero vector in it yields the identity testing algorithm.

We now describe the construction for the j^{th} layer assuming we have basis $\mathbb{B}_{j'}$ for every $j' < j$. Consider a $\times$ gate with its children computing homogeneous polynomials of degree d_1 and d_2 respectively. Notice that $j = d_1 + d_2$ and $0 < d_1, d_2 < j$. Consider the monomial[2] set

$$M = \{m_1 m_2 \mid v_{m_1} \in \mathbb{B}_{d_1} \text{ and } v_{m_2} \in \mathbb{B}_{d_2}\}.$$

We construct vectors $\{v_m \mid m \in M\}$ as follows.

$$v_{m_1 m_2}[g] = \sum_{(g_{d_1}, g_{d_2})} v_{m_1}[g_{d_1}] v_{m_2}[g_{d_2}],$$

where g is a $+$ gate in the j^{th} layer, g_{d_1} is a $+$ gate in the d_1^{th} layer, g_{d_2} is a $+$ gate in the d_2^{th} layer, and there is a $\times$ gate which is input to g and computes the product of g_{d_1} and g_{d_2}.

Let $\mathbb{B}_{d_1,d_2}$ denote a maximal linearly independent subset of $\{v_m \mid m \in M\}$. Then we let $\mathbb{B}_d$ be a maximal linearly independent subset of

$$\bigcup_{d_1+d_2=d} \mathbb{B}_{d_1,d_2}.$$

[2] We note that the nonassociative monomial $m_1 m_2$ is a binary tree with the root having two children: the left child is the root of the binary tree for m_1 and the right child is the root of the binary tree for m_2.

▶ **Claim 7.** *For every monomial m of degree j, v_m is in the span of $\mathbb{B}_j$.*

Proof of Claim. Let $m = m_1 m_2$ and the degree of m_1 is d_1 and the degree of m_2 is d_2 [3]. By *Induction Hypothesis* vectors v_{m_1} and v_{m_2} are in the span of $\mathbb{B}_{d_1}$ and $\mathbb{B}_{d_2}$ respectively. Hence, we can write

$$v_{m_1} = \sum_{i=1}^{D_1} \alpha_i v_{m_i} \quad v_{m_i} \in \mathbb{B}_{d_1} \quad \text{and} \quad v_{m_2} = \sum_{j=1}^{D_2} \beta_j v_{m'_j} \quad v_{m'_j} \in \mathbb{B}_{d_2},$$

where $|\mathbb{B}_{d_j}| = D_j$. Now, for a gate g in the j^{th} layer, By Induction Hypothesis and by construction we have

$$\begin{aligned} v_m[g] &= \sum_{(g_{d_1}, g_{d_2})} v_{m_1}[g_{d_1}] v_{m_2}[g_{d_2}] = \sum_{g_{d_1}, g_{d_2}} (\sum_{i=1}^{D_1} \alpha_i v_{m_i}[g_{d_1}])(\sum_{j=1}^{D_2} \beta_j v_{m'_j}[g_{d_2}]) \\ &= \sum_{i=1}^{D_1} \sum_{j=1}^{D_2} \alpha_i \beta_j \sum_{g_{d_1}, g_{d_2}} v_{m_i}[g_{d_1}] v_{m'_j}[g_{d_2}] = \sum_{i=1}^{D_1} \sum_{j=1}^{D_2} \alpha_i \beta_j v_{m_i m'_j}[g]. \end{aligned}$$

Thus v_m is in the span of $\mathbb{B}_{d_1,d_2}$ and hence in the span of $\mathbb{B}_j$. This proves the claim.
The PIT algorithm only has to check if $\mathbb{B}_d$ has a nonzero vector. This proves the claim. ◀

Suppose the input nonassociative circuit C computing some degree d polynomial $f \in \mathbb{F}\{X\}$ is inhomogeneous. Then, using Lemma 5 we can first compute in polynomial time homogeneous circuits $C_j \ : \ 0 \leq j \leq d$, where C_j computes the degree-j homogeneous part f_j. Then we run the above algorithm on each C_j to check whether f is identically zero. This completes the proof of the theorem. ◀

4 Polynomial Factorization in $\mathbb{F}\{X\}$

In this section we describe our polynomial-time white-box factorization algorithm for polynomials in $\mathbb{F}\{X\}$. More precisely, given as input a nonassociative circuit C computing a polynomial $f \in \mathbb{F}\{X\}$, the algorithm outputs circuits for all irreducible factors of f. The algorithm uses as subroutine the PIT algorithm for polynomial in $\mathbb{F}\{X\}$ described in Section 3.

To facilitate exposition, we completely describe a deterministic polynomial-time algorithm that computes a nontrivial factorization $f = g \cdot h$ of f, by giving circuits for g and h, unless f is irreducible. We will briefly outline how this extends to finding all irreducible factors efficiently.

We start with a special case.

▶ **Lemma 8.** *Let $f \in \mathbb{F}\{X\}$ be a degree d polynomial given by a circuit C of size s such that the constant term in f is zero. Furthermore, suppose there is a factorization $f = g \cdot h$ such that the constant terms in g and h are also zero. Then in deterministic* $\mathrm{poly}(n, d, s)$ *time we can compute the circuits for polynomials g and h.*

Proof. We first consider the even more restricted case when C computes a homogeneous degree d polynomial $f \in \mathbb{F}\{X\}$. For the purpose of computing partial derivatives, it is convenient to transform C into the noncommutative circuit $\tilde{C}$, as explained in Section 2,

[3] Here a crucial point is that for a nonassociative monomial of degree d, such a choice for d_1 and d_2 is *unique*. This is a place where a general noncommutative circuit behaves very differently.

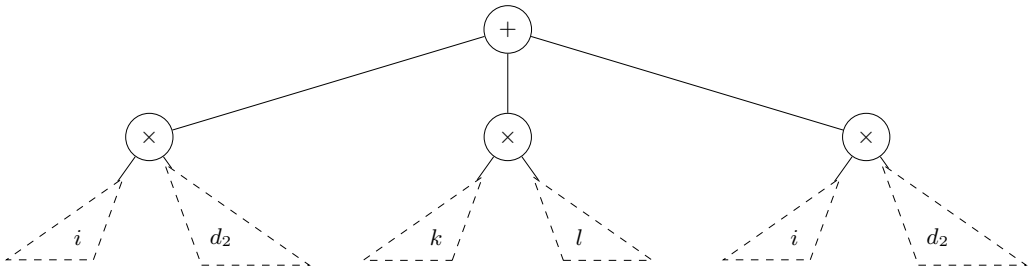


Figure 3 Circuit C_{i+d_2} for f_{i+d_2}.

which computes the fully bracketed polynomial $\tilde{f} \in \mathbb{F}\langle X, (,)\rangle$. Using Theorem 6 we compute a monomial $m = (m_1 m_2)$ where m_1 and m_2 are also fully bracketed. We can transform $\tilde{C}$ to drop the outermost opening and closing brackets. Now, using Lemma 4, we compute the resulting circuits left partial derivative w.r.t. m_1 and right partial derivative w.r.t. m_2. Call these $\tilde{f}_1$ and $\tilde{f}_2$. We can check if $\tilde{f} = (\tilde{f}_1 \tilde{f}_2)$: we first recover the corresponding nonassociative circuits for f_1 and f_2 from the circuits for $\tilde{f}_1$ and $\tilde{f}_2$. Then we can apply the PIT algorithm of Theorem 6 to check if $f = f_1 f_2$. Clearly, f is irreducible iff $f \neq f_1 f_2$. Continuing thus, we can fully factorize f into its irreducible factors.

Now we prove the actual statement. Applying Lemma 5, we compute homogeneous circuits $C_j : 1 \leq j \leq d$ for the homogeneous degree j component f_j of the polynomial f. Clearly $f_d = g_{d_1} h_{d_2}$. We run the PIT algorithm of Theorem 6 on the circuit C_d to extract a monomial m of degree d along with its coefficient $c_m(f_d)$ in f_d. Notice that the monomial m is of the form $m = (m_1\ m_2)$. If g and h are nontrivial factors of f then m_1 and m_2 are monomials in g and h respectively. Compute the circuits for the left and right derivatives with respect to m_1 and m_2.

$$\frac{\partial^{\ell} C_d}{\partial m_1} = c_{m_1}(g_{d_1}) \cdot h_{d_2} \quad \text{and} \quad \frac{\partial^{r} C_d}{\partial m_2} = c_{m_2}(h_{d_2}) \cdot g_{d_1}.$$

In general the $(i + d_2)^{th} : i \leq d - d_2$ homogeneous part of f can be expressed as

$$f_{i+d_2} = g_i h_{d_2} + \sum_{t=i+1}^{i+d_2-1} g_t h_{d_2\ -\ (t-i)}.$$

We depict the circuit C_{i+d_2} for the polynomial f_{i+d_2} in Figure 3. The top gate of the circuit is a + gate. From C_{i+d_2}, we construct another circuit C'_{i+d_2} keeping only those × gates as children whose left degree is i and right degree is d_2. The resulting circuit is shown in Figure 4. The circuit C'_{i+d_2} must compute $g_i h_{d_2}$. By taking the right partial of C'_{i+d_2} with respect to m_2, we obtain the circuit for $c_{m_2}(h_{d_2})\ g_i$.

We repeat the above construction for each $i \in [d_1]$ to obtain circuits for $c_{m_2}(h_{d_2}) g_i$ for $1 \leq i \leq d_1$. Similarly we can get the circuits for $c_{m_1}(g_{d_1}) h_i$ for each $i \in [d_2]$ using the left derivatives with respect to the monomial m_1.

By adding the above circuits we get the circuits C_g and C_h for $c_{m_2}(h_{d_2}) g$ and $c_{m_1}(g_{d_1}) h$ respectively. We set $C_g = \frac{c_{m_2}(h_{d_2})}{c_m(f)} g$ so that $C_g C_h = f$. Using PIT algorithm one can easily check whether g and h are nontrivial factors. In that case we further recurse on g and h to obtain their irreducible factors. ◀

Now we consider the general case when f and its factors g, h have arbitrary constant terms. In the subsequent proofs we assume, for convenience, that $\deg(g) \geq \deg(h)$. The case when $\deg(g) < \deg(h)$ can be handled analogously. We first consider the case $\deg(g) = \deg(h)$.

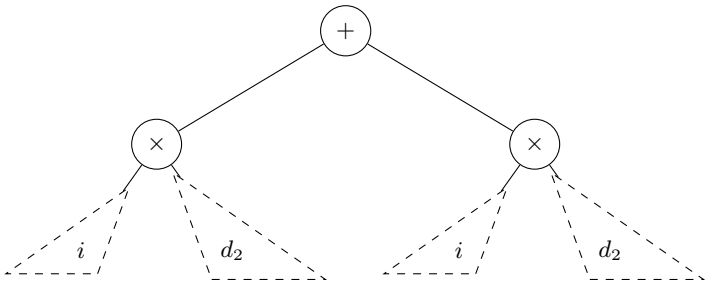


Figure 4 C'_{i+d_2} keeps only degree (i, d_2) type $\times$ gates.

▶ **Lemma 9.** *For a degree d polynomial $f \in \mathbb{F}\{X\}$ given by a circuit C suppose $f = (g+\alpha)(h+\beta)$, where $g, h \in \mathbb{F}\{X\}$ such that $\deg(g) = \deg(h)$, and $\alpha, \beta \in \mathbb{F}$. Suppose $m = (m_1 m_2)$ is a nonzero degree d monomial. Then, in deterministic polynomial time we can compute circuits for the polynomials $c_{m_1}(g) \cdot h$ and $c_{m_2}(h) \cdot g$, where $c_{m_1}(g)$ and $c_{m_2}(h)$ are coefficient of m_1 and m_2 in g and h respectively.*

Proof. We can write $f = (g+\alpha)(h+\beta) = g \cdot h + \beta \cdot g + \alpha \cdot h + \alpha \cdot \beta$. Applying the PIT algorithm of Theorem 6 on f, we compute a maximum degree monomial $m = (m_1 m_2)$. Computing the left derivative of circuit C w.r.t. monomial m_1, after removing the outermost brackets, we obtain a circuit computing $c_{m_1}(g)h + \beta c_{m_1}(g) + \alpha c_{m_1}(h)$. Dropping the constant term, we obtain a circuit computing polynomial $c_{m_1}(g)h$. Similarly, computing the right derivative w.r.t m_2 yields a circuit for $c_{m_2}(h)g + \beta c_{m_2}(g) + \alpha c_{m_2}(h)$. Removing the constant term we get a circuit for $c_{m_2}(h)g$. ◀

When $\deg(g) > \deg(h)$ we can recover $h + \beta$ entirely (upto a scalar factor) and we need to obtain the homogeneous parts of g separately.

▶ **Lemma 10.** *Let $f = (g+\alpha) \cdot (h+\beta)$ be a polynomial of degree d in $\mathbb{F}\{X\}$ given by a circuit C. Suppose $\deg(g) > \deg(h)$. Then, in deterministic polynomial time we can compute the circuit C' for $c_{m_1}(g)(h+\beta)$.*

Proof. Again, applying the PIT algorithm to f we obtain a nonzero degree d monomial $m = (m_1\ m_2)$ of f. If $f = (g+\alpha)(h+\beta)$ then $f = g \cdot h + \alpha h + \beta g + \alpha\beta$. As $\deg(g) > \deg(h)$, the left partial derivative of C with respect to m_1 yields a circuit C' for $c_{m_1}(g)\ (h+\beta)$. ◀

Extracting the homogeneous components from the circuit C' given by Lemma 10, yields circuits for $\{c_{m_1}(g)h_i \ : \ i \in [d_2]\}$. We also get the constant term $c_{m_1}(g)\beta$. Now we obtain the homogeneous components of g as follows.

▶ **Lemma 11.** *Suppose circuit C computes f, where $f = (g+\alpha)\ (h+\beta)$ of degree d, $\alpha, \beta \in \mathbb{F}$, $\deg(g) = d_1$ and $\deg(h) = d_2$ such that $d_1 > d_2$.*

- *Let m be a nonzero degree d monomial of f such that $m = (m_1\ m_2)$. Then circuits for $\{c_{m_2}(h)g_i : i \in [d_1 - d_2 + 1, d_1]\}$ can be computed in deterministic polynomial time.*
- *The $(d_2+i)^{th}$ homogeneous part of f is given by $f_{d_2+i} = \sum_{j=0}^{d_2-1} g_{d_2+i-j}\ h_j + g_i\ h_{d_2}$ for $1 \le i \le d_1 - d_2$. From the circuit C_{d_2+i} of f_{d_2+i}, we can efficiently compute circuits for $\{c_{m_2}(h_{d_2})g_i : 1 \le i \le d_1 - d_2\}$.*

Proof. For the first part, fix any $i \in [d_1 - d_2 + 1, d_1]$, and compute the homogeneous $(i+d_2)^{th}$ part f_{i+d_2} of f by a circuit C_{i+d_2}. Similar to Lemma 8, we focus on the sub-circuits of C_{i+d_2} formed by $\times$ gate of the degree type (i, d_2). Since i is at least $d_1 - d_2 + 1$, such gates can

compute the multiplication of a degree i polynomial with a degree d_2 polynomial. Then, by taking the right partial derivative with respect to m_2 we recover the circuits for $c_{m_2}(h_{d_2})\ g_i$ for any $i \in [d_1 - d_2 + 1, d_1]$.

Next, the goal is to recover the circuits for g_i (upto a scalar multiple), where $1 \leq i \leq d_1 - d_2$, and also recover the constant terms α and β. When $i \leq d_1 - d_2$ a product gate of type (i, d_2) can entirely come from g which requires a different handling.

We explain only the case when $i = d_1 - d_2$ (the others are similar). For $i = d_1 - d_2$, we have $f_{d_1} = \beta g_{d_1} + \sum_{j=1}^{d_2-1} g_{d_1-j}\ h_j + g_{d_1-d_2}\ h_{d_2}$. By Lemma 10, we can compute a circuit C' for $c_{m_1}(g)(h + \beta)$. Extracting the constant term yields $c_{m_1}(g)\beta$. From Lemma 11 we have a circuit C'' for $c_{m_2}(h)g_{d_1}$. Multiplying these circuits, we obtain a circuit C^* for $c_{m_2}(h)c_{m_1}(g)\beta g_{d_1}$. Since $c_{m_2}(h)c_{m_1}(g) = c_m(f)$, dividing C^* by $c_m(f)$ yields a circuit for βg_{d_1}. Note that, by the first part of this lemma, we already have circuits for every term g_{d_1-j} appearing in the above sum. Subtracting $\beta g_{d_1} + \sum_{j=1}^{d_2-1} g_{d_1-j}\ h_j$ from the circuit C_{d_1} for f_{d_1}, yields a circuit for polynomial $g_{d_1-d_2}h_{d_2}$. Computing the right derivative of the resulting circuit w.r.t m_2 (Lemma 4) yields a circuit for $c_{m_2}(h)g_{d_1-d_2}$.

For general $i \leq d_1 - d_2$, when we need to compute g_i, again we will have already computed circuits for all $g_j, j > i$. A suitable right derivative computation will yield a circuit for $c_{m_2}(h)g_i$. ◀

Lemmas 8, 9, 10, and 11 yield an efficient algorithm for computing circuits for the two factors $c_{m_2}(h)(\sum_{i=1}^{d_1} g_i)$ and $c_{m_1}(g)(\sum_{i=1}^{d_2} h_i)$ when $\deg(g) \geq \deg(h)$. The case when $\deg(g) < \deg(h)$ is similarly handled using left partial derivatives in the above lemmas.

Now we explain how to compute the constant terms of the individual factors. We discuss the case when $\alpha \neq 0$. The other case is similar.

First we recall that given a monomial m and a noncommutative circuit C, the coefficient of m in C can be computed in deterministic polynomial time [3]. We know that $f_0 = \alpha \cdot \beta$. We compute the coefficient of the monomial m_1 in the circuits for polynomials $c_{m_2}(h)c_{m_1}(g)gh$, $c_{m_2}(h)g$, and $c_{m_1}(g)h$. Let these coefficients be a, b and c, respectively. Moreover, we know that $c_{m_2}(h)c_{m_1}(g)$ is the coefficient of monomial $m = (m_1\ m_2)$ in f. Let the coefficient of m_1 in f be γ. Let $\gamma_1 = c_{m_1}(g)$ and $\gamma_2 = c_{m_2}(h)$ and $\delta = c_{m_1}(g)c_{m_2}(h)$.

Now equating the coefficient of m_1 from both side of the equation $f = (g + \alpha)(h + \beta)$ and substituting $\beta = \frac{f_0}{\alpha}$, we get

$$\gamma = \frac{a}{\gamma_1\gamma_2} + \frac{\alpha c}{\gamma_1} + \frac{f_0 b}{\alpha\gamma_2} = \frac{a}{\delta} + \frac{\alpha c}{\gamma_1} + \frac{f_0 b}{\alpha\gamma_2}.$$

Letting $\xi = \alpha\gamma_2$, this gives a quadratic equation in the unknown ξ.

$$c\xi^2 + (a - \gamma\delta)\xi + f_0 b\delta = 0.$$

By solving the above quadratic equation we get two solutions A_1 and A_2 for $\xi = \alpha\gamma_2$. Notice that $\beta\gamma_1 = \frac{\delta f_0}{\xi}$. As we have circuits for $c_{m_2}(h)g = \gamma_2 g$ and for $c_{m_1}(g)h = \gamma_1 h$, we obtain circuits for $\gamma_2(g + \alpha)$ and $\gamma_1(h + \beta)$ (two solutions, corresponding to A_1 and A_2). To pick the right solution, we can run the PIT algorithm to check if $\gamma_1\gamma_2 f$ equals the product of these two circuits that purportedly compute $\gamma_2(g + \alpha)$ and $\gamma_1(h + \beta)$.

Over $\mathbb{Q}$ we can just solve the quadratic equation in deterministic polynomial time using standard method. If $\mathbb{F} = \mathbb{F}_q$ for $q = p^r$, we can factorize the quadratic equation in deterministic time $\text{poly}(p, r)$ [15]. Using randomness, one can solve this problem in time $\text{poly}(\log p, r)$ using Berlekamp's factoring algorithm [5]. This also completes the proof of the following.

▶ **Theorem 12.** *Let $f \in \mathbb{F}\{X\}$ be a degree d polynomial given by a circuit of size s. If $\mathbb{F} = \mathbb{Q}$, in deterministic $\mathrm{poly}(s, n, d)$ time we can compute a nontrivial factorization of f or reports f is irreducible. If $\mathbb{F}$ is a finite field such that $char(\mathbb{F}) = p$, we obtain a deterministic $\mathrm{poly}(s, n, d, p)$ time algorithm that computes a nontrivial factorization of f or reports f is irreducible.*

Finally, we state the main result of this paper.

▶ **Theorem 13.** *Let $f \in \mathbb{F}\{X\}$ be a degree d polynomial given by a circuit of size s. Then if $\mathbb{F} = \mathbb{Q}$, in deterministic $\mathrm{poly}(s, n, d)$ time we can output the circuits for the irreducible factors of f. If $\mathbb{F}$ is a finite field such that $char(\mathbb{F}) = p$, we obtain a deterministic $\mathrm{poly}(s, n, d, p)$ time algorithm for computing circuits for the irreducible factors of f.*

▶ Remark. We could apply Theorem 12 repeatedly to find all irreducible factors of the input $f \in \mathbb{F}\{X\}$. However, the problem with that approach is that the circuits for g and h we computed in the proof of Theorem 12, where $f = gh$ is the factorization, is larger than the input circuit C for f by a polynomial factor. Thus, repeated application would incur a superpolynomial blow-up in circuit size. We can avoid that by computing the required partial derivative of g as a suitable partial derivative of the circuit C directly. This will keep the circuits polynomially bounded. This idea is from [2] where it is used for homogeneous noncommutative polynomial factorization. Combined with Theorem 12 this gives the polynomial-time algorithm of Theorem 13.

5 Conclusion

Motivated by the nonassociative circuit lower bound result shown in [7], we study PIT and polynomial factorization in the free nonassociative noncommutative ring $\mathbb{F}\{X\}$ and obtain efficient white-box algorithms for the problems.

Hrubes, Wigderson, and Yehudayoff [7] have also shown exponential circuit-size lower bounds for nonassociative, commutative circuits. It would be interesting to obtain an efficient polynomial identity testing algorithm for that circuit model too. Even a randomized polynomial-time algorithm is not known.

Obtaining an efficient *black-box* PIT in the ring $\mathbb{F}\{X\}$ is also an interesting problem. Of course, for such an algorithm the black-box can be evaluated on a suitable nonassociative algebra. To the best of our knowledge, there seems to be no algorithmically useful analogue of the Amitsur-Levitzki theorem [1].

References

1 Avraham Shimshon Amitsur and Jacob Levitzki. Minimal identities for algebras. *Proceedings of the American Mathematical Society*, 1(4):449–463, 1950.

2 Vikraman Arvind, Pushkar S. Joglekar, and Gaurav Rattan. On the complexity of noncommutative polynomial factorization. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, pages 38–49, 2015. doi:10.1007/978-3-662-48054-0_4.

3 Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. New results on noncommutative and commutative polynomial identity testing. *Computational Complexity*, 19(4):521–558, 2010. doi:10.1007/s00037-010-0299-8.

4 Vikraman Arvind and S. Raja. Some lower bound results for set-multilinear arithmetic computations. *Chicago J. Theor. Comput. Sci.*, 2016 (6), 2016.

5 E. R. Berlekamp. Factoring polynomials over large finite fields*. In *Proceedings of the Second ACM Symposium on Symbolic and Algebraic Manipulation*, SYMSAC'71, pages 223–, New York, NY, USA, 1971. ACM. doi:10.1145/800204.806290.

6 P.M. Cohn. Noncommutative unique factorization domains. *Transactions of the American Math. Society*, 109(2):313–331, 1963.

7 Pavel Hrubes, Avi Wigderson, and Amir Yehudayoff. Relationless completeness and separations. In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity, CCC 2010, Cambridge, Massachusetts, June 9-12, 2010*, pages 280–290, 2010. doi:10.1109/CCC.2010.34.

8 Laurent Hyafil. The power of commutativity. In *18th Annual Symposium on Foundations of Computer Science (FOCS), Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 171–174, 1977. doi:10.1109/SFCS.1977.31.

9 Erich Kaltofen. Factorization of polynomials given by straight-line programs. *Randomness in Computation*, vol. 5 of Advances in Computing Research:375–412, 1989.

10 Swastik Kopparty, Shubhangi Saraf, and Amir Shpilka. Equivalence of polynomial identity testing and polynomial factorization. *Computational Complexity*, 24(2):295–331, 2015. doi:10.1007/s00037-015-0102-y.

11 Guillaume Lagarde, Guillaume Malod, and Sylvain Perifel. Non-commutative computations: lower bounds and polynomial identity testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:94, 2016. URL: http://eccc.hpi-web.de/report/2016/094.

12 Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In *STOC*, pages 410–418, 1991. doi:10.1145/103418.103462.

13 Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005. doi:10.1007/s00037-005-0188-8.

14 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010. doi:10.1561/0400000039.

15 Joachim von zur Gathen and Victor Shoup. Computing frobenius maps and factoring polynomials. *Computational Complexity*, 2:187–224, 1992.

Faster Algorithms for Mean-Payoff Parity Games*†

Krishnendu Chatterjee[‡1], **Monika Henzinger**[§2], **and Alexander Svozil**[¶3]

1 **IST Austria, Klosterneuburg, Austria**
krish.chat@ist.ac.at
2 **Faculty of Computer Science, University of Vienna, Austria**
monika.henzinger@univie.ac.at
3 **Faculty of Computer Science, University of Vienna, Austria**
alexander.svozil@univie.ac.at

Abstract

Graph games provide the foundation for modeling and synthesis of reactive processes. Such games are played over graphs where the vertices are controlled by two adversarial players. We consider graph games where the objective of the first player is the conjunction of a qualitative objective (specified as a parity condition) and a quantitative objective (specified as a mean-payoff condition). There are two variants of the problem, namely, the *threshold* problem where the quantitative goal is to ensure that the mean-payoff value is above a threshold, and the *value* problem where the quantitative goal is to ensure the optimal mean-payoff value; in both cases ensuring the qualitative parity objective. The previous best-known algorithms for game graphs with n vertices, m edges, parity objectives with d priorities, and maximal absolute reward value W for mean-payoff objectives, are as follows: $\mathcal{O}(n^{d+1} \cdot m \cdot W)$ for the threshold problem, and $\mathcal{O}(n^{d+2} \cdot m \cdot W)$ for the value problem. Our main contributions are faster algorithms, and the running times of our algorithms are as follows: $\mathcal{O}(n^{d-1} \cdot m \cdot W)$ for the threshold problem, and $\mathcal{O}(n^d \cdot m \cdot W \cdot \log(n \cdot W))$ for the value problem. For mean-payoff parity objectives with two priorities, our algorithms match the best-known bounds of the algorithms for mean-payoff games (without conjunction with parity objectives). Our results are relevant in synthesis of reactive systems with both functional requirement (given as a qualitative objective) and performance requirement (given as a quantitative objective).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems – Computations on discrete structures, F.4.1 Mathematical logic – Temporal logic

Keywords and phrases graph games, mean-payoff parity, büchi

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.39

1 Introduction

Graph games. A graph game is played on a finite directed graph with two players, namely, player 1 and player 2 (the adversary of player 1). The vertex set is partitioned into player-1

* Full version available at https://arxiv.org/abs/1706.06139.
† The authors are partially supported by the Vienna Science and Technology Fund (WWTF) grant ICT15-003.
‡ K. C. is supported by the Austrian Science Fund (FWF) NFN Grant No S11407-N23 (RiSE/SHiNE) and an ERC Start grant (279307: Graph Games).
§ The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.
¶ The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

and player-2 vertices. At player-1 vertices, player 1 chooses a successor vertex; and at player-2 vertices, player 2 does likewise. The result of playing the game forever is an infinite path through the graph. There has been a long history of using graph games for modeling and synthesizing reactive processes [6, 17, 18]: a reactive system and its environment represent the two players, whose states and transitions are specified by the vertices and edges of a game graph. Consequently, graph games provide the theoretical foundation for modeling and synthesizing reactive processes.

Qualitative and quantitative objectives. For reactive systems, the objective is given as a set of desired paths (such as ω-regular specifications), or as a quantitative optimization objective with a payoff function on the paths. The class of ω-regular specifications provide a robust framework to express all commonly used specifications for reactive systems in verification and synthesis. Parity objectives are a canonical way to express ω-regular objectives [19], where an integer priority is assigned to every vertex, and a path satisfies the parity objective for player 1 if the minimum priority visited infinitely often is even. One of the classical and most well-studied quantitative objectives is the mean-payoff objective, where a reward is associated with every edge, and the payoff of a path is the long-run average of the rewards of the path.

Mean-payoff parity objectives. Traditionally the verification and the synthesis problems were considered with qualitative objectives. However, recently combinations of qualitative and quantitative objectives have received a lot of attention. Qualitative objectives such as ω-regular objectives specify the functional requirements of reactive systems, whereas the quantitative objectives specify resource consumption requirements (such as for embedded systems or power-limited systems). Combining quantitative and qualitative objectives is crucial in the design of reactive systems with both resource constraints and functional requirements [8, 11, 3, 1]. For example, mean-payoff parity objectives are relevant in synthesis of optimal performance lock-synchronization for programs [7], where one player is the synchronizer, the opponent is the environment; the performance criteria is specified as mean-payoff objective; and the functional requirement (e.g., data-race freedom or liveness) as an ω-regular objective. Mean-payoff parity objectives have been used in several other applications, e.g., define permissivity for parity games [4] and robustness in synthesis [2].

Threshold and value problems. For graph games with mean-payoff and parity objectives there are two variants of the problem. First, the *threshold* problem, where a threshold ν is given for the mean-payoff objective, and player 1 must ensure the parity objective and that the mean-payoff is at least ν. Second, the *value* problem, where player 1 maximizes the mean-payoff value while ensuring the parity objective. In the sequel of this section, we will refer to graph games with mean-payoff and parity objectives as mean-payoff parity games.

Previous results. Mean-payoff parity games were first studied in [11], and algorithms for the value problem were presented. It was shown in [9] that the decision problem for mean-payoff parity games lies in NP $\cap$ coNP (similar to the status of mean-payoff games and parity games). For game graphs with n vertices, m edges, parity objectives with d priorities, and maximal absolute reward value W for the mean-payoff objective, the previous known algorithmic bounds for mean-payoff parity games are as follows: For the threshold problem the results of [9] give an $\mathcal{O}(n^{d+4} \cdot m \cdot d \cdot W)$-time algorithm. This algorithmic bound was improved in [4] where an $\mathcal{O}(n^{d+2} \cdot m \cdot W)$-time algorithm was presented for the value problem. The result of [4] does not explicitly present any other better bound for the threshold problem. However, the recursive algorithm of [4] uses value mean-payoff games as a sub-routine, and replacing

value mean-payoff games with threshold mean-payoff games gives an $\mathcal{O}(n)$-factor saving, and yields an $\mathcal{O}(n^{d+1} \cdot m \cdot W)$-time algorithm for the threshold problem for mean-payoff parity games.

Contributions. In this work our main contributions are faster algorithms to solve mean-payoff parity games. Previous and our results are summarized in Table 1.

1. *Threshold problem.* We present an $\mathcal{O}(n^{d-1} \cdot m \cdot W)$-time algorithm for the threshold problem for mean-payoff parity games, improving the previous $\mathcal{O}(n^{d+1} \cdot m \cdot W)$ bound. The important special case of parity objectives with two priorities correspond to Büchi and coBüchi objectives. Our bound for mean-payoff Büchi games and mean-payoff coBüchi games is $\mathcal{O}(n \cdot m \cdot W)$, which matches the best-known bound to solve the threshold problem for mean-payoff objectives [5], and improves the previous known $\mathcal{O}(n^3 \cdot m \cdot W)$ bound [4].
2. *Value problem.* We present an $\mathcal{O}(n^d \cdot m \cdot W \cdot \log(n \cdot W))$-time algorithm for the value problem for mean-payoff parity games, improving the previous $\mathcal{O}(n^{d+2} \cdot m \cdot W)$ bound. Our bound for mean-payoff Büchi games and mean-payoff coBüchi games is $\mathcal{O}(n^2 \cdot m \cdot W \cdot \log(n \cdot W))$, which matches the bound of [5] to solve the value problem for mean-payoff objectives, and improves the previous known $\mathcal{O}(n^4 \cdot m \cdot W)$ bound.

Technical contributions. Our main technical contributions are as follows:

1. First, for the threshold problem, we present a decremental algorithm for mean-payoff games that supports a sequence of vertex-set deletions along with their player-2 reachability set. We show that the total running time is $\mathcal{O}(n \cdot m \cdot W)$, which matches the best-known bound for the static algorithm to solve mean-payoff games. We show that using our decremental algorithm we can solve the threshold problem for mean-payoff Büchi games in time $\mathcal{O}(n \cdot m \cdot W)$.
2. Second, for mean-payoff coBüchi games, the decremental approach does not work. We present a new static algorithm for threshold mean-payoff games that identifies subsets X of the winning set for player 1, where the time complexity is $\mathcal{O}(|X| \cdot m \cdot W)$, i.e., it replaces n with the size of the set identified. We show that with our new static algorithm we can solve the threshold problem for mean-payoff coBüchi games in time $\mathcal{O}(n \cdot m \cdot W)$.
3. Finally, we show for all mean-payoff parity objectives, given an algorithm for the threshold problem, the value problem can be solved in time $n \cdot \log(n \cdot W)$ times the complexity of the threshold problem.

Related works. The problem of graph games with mean-payoff parity objectives was first studied in [11]. The NP $\cap$ coNP complexity bound was established in [9], and an improved algorithm for the problem was given in [4]. The mean-payoff parity objectives has also been considered in other stochastic setting such as Markov decision processes [10, 12] and stochastic games [13]. The algorithmic approaches for stochastic games build on the results for non-stochastic games. In this work, we present faster algorithms for mean-payoff parity games.

2 Preliminaries

Graphs. A graph $G = (V, E)$ consists of a finite set V of vertices and a finite set of edges $E \subseteq V \times V$. Given a graph $G = (V, E)$ and a subset $U \subseteq V$ we denote by $G \restriction U = (V', E')$ the subgraph of G induced by U, i.e., $V' = U$, $E' = (U \times U) \cap E$. For $v \in V$ we denote by $In(v)$ (resp., $Out(v)$) the set of incoming (resp., outgoing) vertices, i.e., $In(v) = \{v' \mid (v', v) \in E\}$, and $Out(v) = \{v' \mid (v, v') \in E\}$.

Table 1 Algorithmic bounds for mean-payoff (MP) and parity objectives, and special cases: threshold problem (left) and value problem (right).

	threshold problem		value problem	
	Previous	Our	Previous	Our
MP-Büchi	$\mathcal{O}(n^3 \cdot m \cdot W)$	$\mathcal{O}(n \cdot m \cdot W)$	$\mathcal{O}(n^4 \cdot m \cdot W)$	$\mathcal{O}(n^2 \cdot m \cdot W \cdot \log(nW))$
MP-coBüchi	$\mathcal{O}(n^3 \cdot m \cdot W)$	$\mathcal{O}(n \cdot m \cdot W)$	$\mathcal{O}(n^4 \cdot m \cdot W)$	$\mathcal{O}(n^2 \cdot m \cdot W \cdot \log(nW))$
MP-parity	$\mathcal{O}(n^{d+1} \cdot m \cdot W)$	$\mathcal{O}(n^{d-1} \cdot m \cdot W)$	$\mathcal{O}(n^{d+2} \cdot m \cdot W)$	$\mathcal{O}(n^d \cdot m \cdot W \cdot \log(nW))$

Game graphs. A game graph $\Gamma = (V, E, \langle V_1, V_2 \rangle)$ is a graph whose vertex set is partitioned into V_1 and V_2, (i.e., $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$). In a game graph every vertex $v \in V$ has a successor $v' \in V$, i.e., $Out(v) \neq \emptyset$ for all $v \in V$. Given a game graph Γ and a set U such that for all vertices u in U we have $Out(u) \cap U \neq \emptyset$, we denote by $\Gamma \restriction U$ the subgame induced by U.

Plays. Given a game graph Γ and a starting vertex v_0, the game proceeds in rounds. In each round, if the current vertex belongs to player 1, then player 1 chooses a successor vertex, and player 2 does likewise if the current vertex belongs to player 2. The result is a *play* ρ which is an infinite path from v_0, i.e., $\rho = v_0 v_1 \ldots$, where every $(v_i, v_{i+1}) \in E$ for all $i \geq 0$. We denote by $Plays(\Gamma)$ the set of all plays of the game graph.

Strategies. Strategies are recipes to extend prefixes of plays by choosing the next vertex. Formally, a strategy for player-1 is a function $\sigma_1 : V^* \cdot V_1 \mapsto V$ such that $(v, \sigma_1(\rho \cdot v)) \in E$ for all $v \in V_1$ and all $\rho \in V^*$. We define strategies σ_2 for player 2 analogously. We denote by Σ_1 and Σ_2 the set of all strategies for player 1 and player 2, respectively. Given strategies σ_1 and σ_2 for player 1 and player 2, and a starting vertex v_0, there is a unique play $\rho = v_0 v_1 \ldots$ such that for all $i \geq 0$, (a) if $v_i \in V_1$ then $v_{i+1} = \sigma_1(v_0 \ldots v_i)$; and (b) if $v_i \in V_2$ then $v_{i+1} = \sigma_2(v_0 \ldots v_i)$. We denote the unique play as $outcome(v_0, \sigma_1, \sigma_2)$. A strategy is *memoryless* if it is independent of the past and depends only on the current vertex, and hence can be defined as a function $\sigma_1 : V_1 \mapsto V$ and $\sigma_2 : V_2 \mapsto V$, respectively.

Objectives and parity objectives. An objective for a game graph Γ is a subset of the possible plays, i.e., $\phi \subseteq Plays(\Gamma)$. Given a play ρ we denote by $Inf(\rho)$ the set of vertices that appear infinitely often in ρ. A *parity* objective is defined with a priority function p that maps every vertex to a non-negative integer priority, and a play satisfies the parity objective for player 1 if the minimum priority vertex that appear infinitely often is even. Formally, the parity objective is $Parity_\Gamma(p) = \{\rho \in Plays(\Gamma) \mid \min\{p(v) \mid v \in Inf(\rho)\} \text{ is even}\}$. The Büchi and coBüchi objectives are special cases of parity objectives with two priorities only. We have $p : V \mapsto \{0, 1\}$ for Büchi objectives and $p : V \mapsto \{1, 2\}$ for the coBüchi objectives.

Payoff functions. Consider a game graph Γ, and a weight function $w : E \mapsto \mathbb{Z}$ that maps every edge to an integer. The mean-payoff function maps every play to a real-number and is defined as follows: For a play $\rho = v_0 v_1 \ldots$ in $Plays(\Gamma)$ we have $MP(w, \rho) = \liminf_{n \mapsto \infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} w(v_i, v_{i+1})$. The mean-payoff parity function also maps every play to a real-number or $-\infty$ as follows: if the parity objective is satisfied, then the value is the mean-payoff value, else it is $-\infty$. Formally, for a play ρ, we have

$$MPP_\Gamma(w, p, \rho) = \begin{cases} MP_\Gamma(w, \rho) & \text{if } \rho \in Parity_\Gamma(p); \\ -\infty & \text{if } \rho \notin Parity_\Gamma(p). \end{cases}$$

Threshold mean-payoff parity objectives. Given a threshold $\nu \in \mathbb{Q}$, the *threshold mean-payoff objective* $MeanPayoff_\Gamma(\nu) = \{\rho \in Plays(\Gamma) \mid MP(\rho) \geq \nu\}$ requires that the mean-payoff value is at least ν. The *threshold mean-payoff parity objective* is a conjunction of a parity objective and a threshold mean-payoff objective, i.e., $Parity_\Gamma(p) \cap MeanPayoff_\Gamma(\nu)$.

Winning strategies. Given an objective (such as parity, threshold mean-payoff, or threshold mean-payoff parity) ϕ, a vertex v is winning for player 1, if there is a strategy σ_1 such that for all strategies σ_2 of player 2, the play $outcome(v, \sigma_1, \sigma_2) \in \phi$ (i.e., the play satisfies the objective). We denote by $W_1(\phi)$ the set of winning vertices (or the winning region) for player 1 for the objective ϕ. The notation $W_2(\overline{\phi})$ for complementary objectives $\overline{\phi}$ for player 2 is similar.

Value functions. Given a payoff function f (such as the mean-payoff function, or the mean-payoff parity function), the value for player 1 is the maximal payoff that she can guarantee against all strategies of player 2. Formally,

$$val_\Gamma(f)(v) = \sup_{\sigma_1 \in \Sigma_1} \inf_{\sigma_2 \in \Sigma_2} f(outcome(v, \sigma_1, \sigma_2)).$$

Attractors. The player-1 attractor $Attr_1(S)$ of a given set $S \subseteq V$ is the set of vertices from which player-1 can force to reach a vertex in S. It is defined as the limit of the sequence $A_0 = S; A_{i+1} = A_i \cup \{v \in V_1 \mid Out(v) \cap A_i \neq \emptyset\} \cup \{v \in V_2 \mid Out(v) \subseteq A_i\}$ for all $i \geq 0$. Th Player-2 attractor $Attr_2(S)$ is defined analogously exchanging the roles of player 1 and player 2. The complement of an attractor induces a game graph, as in the complement every vertex has an outgoing edge in the complement set.

Relevant parameters. In this work we will consider computing the winning region for threshold mean-payoff parity objectives, and the value function for mean-payoff parity objectives. We will consider the following relevant parameters: n denotes the number of vertices, m denotes the number of edges, d denotes the number of priorities of the parity function p, and W is the maximum absolute value of the weight function w.

3 Decremental Algorithm for Threshold Mean-Payoff Games

In this section we present a decremental algorithm for threshold mean-payoff games that supports deleting a sequence of sets of vertices along with their player-2 attractors. The overall running time of the algorithm is $\mathcal{O}(n \cdot m \cdot W)$.

Key idea. A static algorithm based on the notion of progress measure for mean-payoff games was presented in [5]. We show that the progress measure is monotonic wrt to the deletion of vertices and their player-2 attractors. We use an amortized analysis to obtain the running time of our algorithm.

Mean-payoff progress measure. Let Γ be a mean-payoff game with threshold ν. Progress measure is a function f which maps every vertex in Γ to an element of the set $C_\Gamma = \{i \in \mathbb{N} \mid i \leq nW\} \cup \{\top\}$, i.e., $f : V \mapsto C_\Gamma$. Let $(\preceq, C_\Gamma)$ be a total order, where $x \preceq y$ for $x, y \in C_\Gamma$ holds iff $x \leq y \leq nW$ or $y = \top$. We define the operation $\ominus : C_\Gamma \times \mathbb{Z} \mapsto C_\Gamma$ for all $a \in C_\Gamma$ and $b \in \mathbb{Z}$ as follows:

$$a \ominus b = \begin{cases} \max(0, a - b) & \text{if } a \neq \top \text{ and } a - b \leq nW, \\ \top & \text{otherwise.} \end{cases}$$

A player-1 vertex v is *consistent* if $f(v) \succeq f(v') \ominus w(v,v')$ for *any* $v' \in Out(v)$. A player-2 vertex v is *consistent* if $f(v) \succeq f(v') \ominus w(v,v')$ for *all* $v' \in Out(v)$. Let $v \in V$ then $\mathit{lift}(\cdot, v) : [V \mapsto C_\Gamma] \mapsto [V \mapsto C_\Gamma]$ is defined by $\mathit{lift}(f, v) = g$ where:

$$g(u) = \begin{cases} f(u) & \text{if } u \neq v, \\ \min\{f(v') \ominus w(v,v') \mid (v,v') \in E\} & \text{if } u = v \in V_1, \\ \max\{f(v') \ominus w(v,v') \mid (v,v') \in E\} & \text{if } u = v \in V_2. \end{cases}$$

Static Algorithm The static algorithm in [5] is an iterative algorithm which maintains and returns a progress measure f and a list L of vertices which are not consistent. The initial progress measure of every vertex is set to zero. Also, $w(e)$ is set to $w(e) - \nu$ for all edges e in E. The list L is initialized with the vertices which are not consistent considering the initial progress measure. Then the following steps are executed in a while-loop:

1. Take out a vertex v of L.
2. Perform the *lift*-operation on the vertex, i.e., $f \leftarrow \mathit{lift}(f, v)$.
3. If a vertex v' in $In(v)$ is not consistent, put v' into L.
4. If L is empty, return f else proceed to the next iteration.

If every vertex is consistent, i.e., the list L is empty, the winning region of player 1 is the set of vertices which are not set to $\top$ in f, i.e., $W_1(\nu) = \{v \in V \mid f(v) \neq \top\}$.

Decremental input/output. Let Γ be a mean-payoff game with threshold ν. The *input* to the decremental algorithm is a sequence of sets $A_1, A_2, \ldots, A_k$, such that each A_i is a player-2 attractor of a set X_i in the game $\Gamma_i = \Gamma \restriction (V \setminus \bigcup_{j<i} A_j)$. The *output requirement* is the player-1 winning set after the deletion of $\bigcup_{j<i} A_j$ for $i = 1, \ldots, k$, i.e., the output requirement is the sequence $Z_1, Z_2, \ldots, Z_k$, where $Z_i = W_1(\phi)$ in $\Gamma_i = \Gamma \restriction (V \setminus \bigcup_{j<i} A_j)$, where $\phi = \mathit{MeanPayoff}_{\Gamma_i}(\nu)$ is the threshold mean-payoff objective. In other words, we repeatedly delete a vertex set X_i along with its player-2 attractor A_i from the current game graph Γ_i, and require the winning set for player 1 as an output after each deletion.

Decremental algorithm. We maintain a progress measure f_i, $1 \leq i \leq k$, during the whole sequence of deletions. The initial progress measure f_1 for the mean-payoff game Γ with threshold mean-payoff objective ϕ is calculated using the static algorithm. For all edges e in E, we set $w(e) = w(e) - \nu$. In iteration i with input A_i, in the game Γ_i with its corresponding vertex set V_i the following steps are executed:

1. If a vertex in the set $\{v \in V_i \setminus A_i \mid \exists v' : \ v' \in Out(v) \wedge v' \in A_i\}$ is not consistent in f_i without the set A_i, put it in a list L_i.
2. Delete the set A_i from Γ_i to receive Γ_{i+1} (and thus V_{i+1}).
3. Execute steps (1)-(4) of the above described iterative algorithm from [5] initialized with Γ_{i+1}, L_i and f_i restricted to the vertices in V_{i+1}.
4. Finally the winning region of player 1 can be extracted from the obtained progress measure f_{i+1}, i.e., $W_1(\phi) = \{v \in V_{i+1} \mid f(v) \neq \top\}$.

Correctness. Let Γ be a game graph, ϕ a threshold objective and $A_1, A_2, \ldots, A_k$ a sequence of sets, such that each A_i is a player-2 attractor in the game $\Gamma_i = \Gamma \restriction (V \setminus \bigcup_{j<i} A_j)$. To show the correctness of the decremental algorithm we need to show that the condition that the list L contains all vertices which are not consistent is an invariant of the decremental algorithm at line 3. This property was proved for the static algorithm in [5].

▶ **Lemma 1.** *The condition that L_i contains all vertices which are not consistent with the progress measure f_i restricted to V_{i+1} in Γ_{i+1} is an invariant of the static algorithm called in step 3 of the decremental algorithm for $1 \leq i \leq k-1$.*

Proof. The fact that the static algorithm correctly returns a progress measure with only consistent vertices when the invariant holds was shown in [5]. It was also shown in [5] that the invariant is maintained in the loop. It remains to show that the condition holds when we call the static algorithm at step 3. For the base case, let $i = 1$. In the initial progress measure f_1 and the initial game graph Γ_1, every vertex is consistent. By the definition of a player-2 attractor, deleting the set A_1 potentially removes edges (v, v') where v is a player-1 vertex in $V \setminus A_1$ and v' is in A_1. (Note that v cannot be a player-2 vertex.) All of the vertices not consistent anymore are added to L_i in step 1 of the decremental algorithm. For the inductive step let $i = j$. By induction hypothesis, all vertices which were not consistent with the progress measures f_{h-1} restricted to V_h for $2 \leq h \leq j$ were added to the corresponding lists. Thus by the correctness of the static algorithm, it correctly computes the new progress measure f_h for the game graph Γ_h where every vertex is consistent. Thus also every vertex in the progress measure f_j restricted to V_j is consistent. Again the player-2 attractor is removed and vertices which are not consistent with progress measure f_j restricted to V_{j+1} are put into L_j by step 1 of the algorithm. ◀

Thus we proved that the static algorithm always correctly updates to the new progress measure in each iteration. The winning region of player-1 is obtained by the returned progress measure (step 4). The decremental algorithm thus correctly computes the sequence $Z_1, Z_2, \ldots Z_k$, where $Z_i = W_1(\phi)$ in Γ_i.

Running Time. The calculation of the initial progress measure for the mean-payoff game Γ with threshold ν is in time $\mathcal{O}(n \cdot m \cdot W)$. The vertices which are not consistent anymore after the deletion of A_i can be found in time $\mathcal{O}(m)$ (step 1). As at most n such sets A_i exist, the running time is $\mathcal{O}(mn)$. In step 3 the static algorithm is executed with our current progress measure f_i: Every time a vertex v is picked from the list L_i it costs $\mathcal{O}(|Out(v) + In(v)|)$ time to use *lift* on it and to look for vertices in $In(v)$ which are not consistent anymore (steps 1-3 in the static algorithm). This cost is charged to its incident edges. Note that deleting a set of vertices and their corresponding player-2 attractor will only potentially *increase* the progress measure of some player-1 vertices. As we can increase the progress measure of every vertex only nW times before it is set to $\top$ where it is always consistent, we get the desired bound of $\mathcal{O}(m \cdot n \cdot W)$.
Thus our decremental algorithm for threshold mean-payoff games works as desired and we obtain the following result:

▶ **Theorem 2.** *Given a game graph Γ, a threshold mean-payoff objective ϕ and a sequence of sets $A_1, A_2, \ldots, A_k$ such that each A_i is a player-2 attractor of a set X_i in the game $\Gamma_i = \Gamma \restriction (V \setminus \bigcup_{j<i} A_j)$, the sequence $Z_1, Z_2, \ldots, Z_k$, where $Z_i = W_1(\phi)$ in Γ_i can be computed in $\mathcal{O}(n \cdot m \cdot W)$ time.*

▶ Remark. Note that the running time analysis of our decremental algorithm crucially depends on the monotonicity property of the progress measure. If edges are both added and deleted, then the monotonicity property does not hold. Hence obtaining a fully dynamic algorithm that supports both addition/deletion of vertices/edges with running time $\mathcal{O}(n \cdot m \cdot W)$ is an interesting open problem. However, we will show that for solving mean-payoff parity games, the decremental algorithm plays a crucial part.

4 Threshold Mean-Payoff Parity Games

In this section we present algorithms for threshold mean-payoff parity games. Our most interesting contributions are for the base case of mean-payoff Büchi- and mean-payoff coBüchi objectives, and the general case follows a standard recursive argument.

4.1 Threshold Mean-Payoff Büchi Games

In this section we consider threshold mean-payoff Büchi games.

Algorithm for threshold mean-payoff Büchi games. The basic algorithm is an iterative algorithm that deletes player-2 attractors. The algorithm proceeds in iterations. In iteration i, let D_i be the set of vertices already deleted. Consider the subgame $\Gamma_i = \Gamma \restriction (V \setminus D_i)$. Then the following steps are executed:

1. Let $V^i = V \setminus D_i$ and B_i denote the set of Büchi vertices (or vertices with priority 0) in Γ_i. Compute $Y_i = Attr_1(B_i)$ the player-1 attractor to B_i in Γ_i.
2. Let $X_i = V^i \setminus Y_i$. If X_i is non-empty, remove $A_i = Attr_2(X_i)$ from the game graph, and proceed to the next iteration.
3. Else $V^i = Y_i$. Let $U_i = W_1(\phi)$ in Γ_i, where $\phi = \mathit{MeanPayoff}(\nu)$, be the winning region for the threshold mean-payoff objective in Γ_i. Let $X_i = V^i \setminus U_i$. If X_i is non-empty, remove $A_i = Attr_2(X_i)$ from the game graph, and proceed to the next iteration. If X_i is empty, then the algorithm stops and all the remaining vertices are winning for player 1 for the threshold mean-payoff Büchi objective.

Correctness. Since the correctness argument has been used before [11], we only present a brief sketch: The basic correctness argument is to show that all vertices removed over all iterations do not belong to the winning set for player 1. In the end, for the remaining vertices, player 1 can ensure to reach the Büchi vertices, and ensures the threshold mean-payoff objectives. A strategy that plays for the threshold mean-payoff objectives longer and longer, and in between visits the Büchi vertices, ensures that the threshold mean-payoff Büchi objective is satisfied.

Running time analysis. We observe that the total running time to compute all attractors is at most $\mathcal{O}(n \cdot m)$, since the algorithm runs for $\mathcal{O}(n)$ iterations and each attractor computation is linear time. In step 3, the algorithm needs to compute the winning region for threshold mean-payoff objective. The algorithm always removes a set X_i and its player-2 attractor A_i, and requires the winning set for player 1. Thus we can use the decremental algorithm from Section 3, which precisely supports these operations. Hence using Theorem 2 in the algorithm for threshold mean-payoff Büchi games, we obtain the following result.

▶ **Theorem 3.** *Given a game graph Γ and a threshold mean-payoff Büchi objective ϕ, the winning set $W_1(\phi)$ can be computed in $\mathcal{O}(m \cdot n \cdot W)$ time.*

4.2 Threshold Mean-Payoff coBüchi Games

In this section we will present an $\mathcal{O}(n \cdot m \cdot W)$-time algorithm for threshold mean-payoff coBüchi games. We start with the description of the basic algorithm for threshold mean-payoff coBüchi games.

Algorithm for threshold mean-payoff coBüchi games. The basic algorithm is an iterative algorithm that deletes player-1 attractors. The algorithm proceeds in iteration. In iteration i, let D_i be the set of vertices already deleted. Consider the subgame $\Gamma_i = \Gamma \restriction (V \setminus D_i)$. Then the following steps are executed:

1. Let $V^i = V \setminus D_i$ and C_i denote the set of coBüchi vertices (or vertices with priority 1) in Γ_i. Compute $Y_i = Attr_2(C_i)$ the player-2 attractor to C_i in Γ_i.

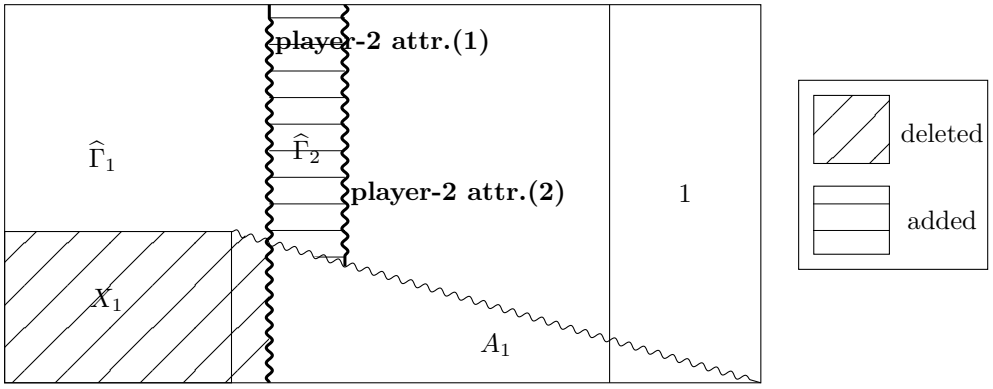


Figure 1 Pictorial illustration of threshold mean-payoff coBüchi games. The subgames $\widehat{\Gamma}_1$ and $\widehat{\Gamma}_2$ are shown. We observe that $\widehat{\Gamma}_2$ is obtained both by addition and deletion of game parts to $\widehat{\Gamma}_1$.

2. Let $X_i = V^i \setminus Y_i$. Consider the subgame $\widehat{\Gamma}_i = \Gamma_i \restriction X_i$. Compute the winning region $Z_i = W_1(\phi)$ for player 1 in $\widehat{\Gamma}_i$, where $\phi = \mathit{MeanPayoff}(\nu)$ is the threshold mean-payoff objective.
3. If Z_i is non-empty, remove $\mathit{Attr}_1(Z_i)$ from Γ_i, and proceed to the next iteration. Else if Z_i is empty, then all remaining vertices are winning for player 2.

Correctness argument. Consider the subgame Γ_i. In each subgame $\widehat{\Gamma}_i$ of Γ_i all edges of player 2 are intact, since it is obtained after removing a player-2 attractor Y_i. Moreover, there is no priority-1 vertex in $\widehat{\Gamma}_i$. Hence ensuring the threshold mean-payoff objective in $\widehat{\Gamma}_i$ for player 1 ensures satisfying the threshold mean-payoff coBüchi objective. Hence the set Z_i and its player-1 attractor belongs to the winning set of player 1 and can be removed. Thus all vertices removed are part of the winning region for player 1. Upon termination, in $\widehat{\Gamma}_i$, player 1 cannot satisfy the threshold mean-payoff condition from any vertex. Consider a player-2 strategy, where in $\widehat{\Gamma}_i$ player 2 falsifies the threshold mean-payoff condition, and in Y_i plays an attractor strategy to reach C_i (priority-1 vertices). Given such a strategy, either (a) Y_i is visited infinitely often, and then the coBüchi objective is violated; or (b) from some point on the play stays in $\widehat{\Gamma}_i$ forever, and then the threshold mean-payoff objective is violated. This shows the correctness of the algorithm. However, the running time of this algorithm is not $\mathcal{O}(n \cdot m \cdot W)$. We now present the key ideas to obtain an $\mathcal{O}(n \cdot m \cdot W)$-time algorithm.

First intuition. Our first intuition is as follows. In step 2 of the above algorithm, instead of obtaining the whole winning region $W_1(\phi)$ in $\widehat{\Gamma}_i$ it suffices to identify a subset X_i of the winning region (if it is non-empty) and remove its player-1 attractor. We call this the modified algorithm for threshold mean-payoff coBüchi games. We first describe why we cannot use the decremental approach in the following remark.

▶ Remark. Consider the subgames for which the threshold mean-payoff objective must be solved. Consider Figure 1. The first player-2 attractor removal induces subgame $\widehat{\Gamma}_1$. After identifying a winning region X_1 of $\widehat{\Gamma}_1$ we remove its player-1 attractor A_1. After removal of A_1, we consider the second player-2 attractor to the priority-1 vertices. The removal of this attractor induces $\widehat{\Gamma}_2$. We observe comparing $\widehat{\Gamma}_1$ and $\widehat{\Gamma}_2$ that certain vertices are removed, whereas other vertices are added. Thus the subgames to be solved for threshold mean-payoff objectives do not satisfy the condition of decremental or incremental algorithms (see Remark 3).

Second intuition. While we cannot use the decremental algorithm, we can solve the problem in $\mathcal{O}(n \cdot m \cdot W)$ time, if we have a modified static algorithm for threshold mean-payoff games, with the following property: (a) it identifies a subset of the winning region X for player 1, if the winning region is non-empty, in time $\mathcal{O}(|X| \cdot m \cdot W)$; (b) if the winning region is empty, it returns the empty set, and then it takes time $\mathcal{O}(n \cdot m \cdot W)$. With such an algorithm we analyze the running time of the above modified algorithm for threshold mean-payoff coBüchi games. The total time required for all attractor computations is again $\mathcal{O}(n \cdot m)$. Otherwise, we use the modified static algorithm to remove vertices of player-1 and to remove set of size X we take $\mathcal{O}(|X| \cdot m \cdot W)$ time, and thus we can charge each vertex $\mathcal{O}(m \cdot W)$ time. Hence the total time required is $\mathcal{O}(n \cdot m \cdot W)$. In the rest of the section we present this modified static algorithm for threshold mean-payoff games.

Problem Statement.

Input:	Mean-payoff game Γ with threshold ν.
Question:	If $W_1(\mathit{MeanPayoff}(\nu))$ is non-empty, return a nonempty set $X \subseteq W_1(\mathit{MeanPayoff}(\nu))$ in time $\mathcal{O}(\lvert X\rvert \cdot m \cdot W)$, else return $\emptyset$ in time $\mathcal{O}(n \cdot m \cdot W)$.

Modified static algorithm for threshold mean-payoff games. The basic algorithm for threshold mean-payoff games computes a progress measure, with a defined top element value $\top$. If the progress measure has the value $\top$ for a vertex, then the vertex is declared as winning for player 2. With value $\top = n \cdot W$, the correct winning region for both players can be identified. Moreover, for a given value α for $\top$, the progress measure algorithm requires $\mathcal{O}(\alpha \cdot m)$ time. Our modified static algorithm is based on the following idea:

1. Consider a value $\alpha \leq n \cdot W$ for the top element. With this reduced value for the top element, if a winning region is identified for player 1, then it is a subset of the whole winning region for player 1.
2. We will iteratively double the value for the top element.

Given the above ideas our algorithm is an iterative algorithm defined as follows: Initialize top value $\top_0 = W$. The i-th iteration is as follows:

1. Run the progress measure algorithm with top value $\top_i$.
2. If a winning region X for player is identified, return X.
3. Else $\top_{i+1} = 2 \cdot \top_i$ (i.e., the top value is doubled).
4. If $\top_{i+1} \geq 2 \cdot n \cdot W$, stop the algorithm and return $\emptyset$, else proceed to the next iteration.

Correctness and running time analysis. The key steps of the correctness argument and the running time analysis are as follows:

1. The above algorithm is correct, since if it returns a set X then it is a subset of the winning set for player 1.
2. If the algorithm returns a winning set with top value α, then the total running time till this iteration is $m \cdot (\alpha + \alpha/2 + \alpha/4 + \cdots)$, because the progress with top value α requires time $\mathcal{O}(\alpha \cdot m)$. Hence the total running time if a set X is returned with top value α is $\mathcal{O}(\alpha \cdot m)$.
3. Let Z be a set of vertices such that no player-2 vertex in Z has an edge out of Z, and the whole subgame $\Gamma \restriction Z$ is winning for player 1. Then a winning strategy in Z ensures that a progress measure with top value $|Z| \cdot W$ would identify the set Z as a winning set.
4. From above it follows that if the winning set X is identified at top value α, but no winning set was identified with top value $\alpha/2$, then the size of the winning set is at least $\alpha/(2W)$.

5. It follows from above that if a set X is identified, then the total running time to obtain set X is $\mathcal{O}(|X| \cdot m \cdot W)$.
6. Moreover, the total running time of the algorithm when no set X is identified is in $\mathcal{O}(n \cdot m \cdot W)$, and in this case, the winning region is empty.

Thus we solved the modified static algorithm for threshold mean-payoff games as desired and obtain the following result.

▶ **Theorem 4.** *Given a mean-payoff game Γ and a threshold ν, let $Z = W_1(MeanPayoff(\nu))$. If $Z \neq \emptyset$, then a non-empty set $X \subseteq Z$ can be computed in time $\mathcal{O}(|X| \cdot m \cdot W)$, else an empty set is returned if $Z = \emptyset$, which takes time $\mathcal{O}(n \cdot m \cdot W)$.*

Using the above algorithm to compute the winning set for player 1 in the subgames, we obtain an algorithm for threshold mean-payoff coBüchi games in time $\mathcal{O}(n \cdot m \cdot W)$.

▶ **Theorem 5.** *Given a game graph Γ and a threshold mean-payoff coBüchi objective ϕ, the winning set $W_1(\phi)$ can be computed in $\mathcal{O}(n \cdot m \cdot W)$ time.*

4.3 Threshold Mean-Payoff Parity Games

The algorithm for threshold mean-payoff parity games is the standard recursive algorithm [11] (classical parity game-style algorithm) that generalizes the Büchi and coBüchi cases (which are the base cases). The running time recurrence is as follows: $T(n, d, m, w) = n(T(n, d-1, m) + \mathcal{O}(m)) + \mathcal{O}(nmW)$. Using our approach we obtain the following result.

▶ **Theorem 6.** *Given a game graph Γ and a threshold mean-payoff parity objective ϕ, the winning set $W_1(\phi)$ can be computed in $\mathcal{O}(n^{d-1} \cdot m \cdot W)$ time.*

5 Optimal Values for Mean-payoff Parity Games

In this section we present an algorithm which computes the value function for mean-payoff parity games. For mean-payoff games a dichotomic search approach was presented in [5]. We show that such an approach can be generalized to mean-payoff parity games.

Range of Values for the Dichotomic Search. To describe the algorithm we recall a lemma about the possible range of optimal values of a mean-payoff parity game. The lemma is an easy consequence of the characterization of [11] that the mean-payoff parity value coincide with the mean-payoff value, and the possible range of value for mean-payoff games.

▶ **Lemma 7** ([11, 15, 16])**.** *Let Γ be a mean-payoff parity game. For each vertex $v \in V$, the optimal value $val_\Gamma(MPP)(v)$ is a rational number $\frac{y}{z}$ such that $1 \leq z \leq n$ and $|y| \leq z \cdot W$.*

By Lemma 7 the value of each vertex $v \in V$, is contained in the following set of rationals

$$S^\Gamma = \left\{ \frac{y}{z} \;\middle|\; y, z \in \mathbb{Z}, 1 \leq z \leq n \wedge -z \cdot W \leq y \leq z \cdot W \right\}.$$

▶ **Definition 8.** Let Γ be a mean-payoff parity game. We denote the set of vertices $v \in V$ such that $val_\Gamma(MPP)(v) \circ \mu$ where $\circ \in \{<, \leq, =, \geq, >\}$ with $V_\Gamma^{\circ\mu}$.

Key Observation. Let $\Gamma = (V, E, \langle V_1, V_2 \rangle, w, p)$ be a mean-payoff parity game. Let $\mu \in [-W, W]$. The sets $V_\Gamma^{>\mu}, V_\Gamma^{=\mu}$ and $V_\Gamma^{<\mu}$ can be computed using any algorithm for threshold mean-payoff parity games twice (for example using Theorem 6). To calculate $V_\Gamma^{\geq\mu}$ and $V_\Gamma^{<\mu}$ use the algorithm on Γ with the mean-payoff parity objective $\phi = Parity_\Gamma(p) \cap$

$\mathit{MeanPayoff}_\Gamma(\mu)$. Consider $\Gamma' = (V, E, \langle V_2, V_1\rangle, w', p)$, where $w'(e) = -w(e)$ for all edges $e \in E$ and player-1 and player-2 vertices are swapped. To calculate $V_\Gamma^{\leq\mu}$ and $V_\Gamma^{>\mu}$ use the algorithm on Γ' with mean-payoff parity objective $\phi = \mathit{Parity}_{\Gamma'}(p) \cap \mathit{MeanPayoff}_{\Gamma'}(-\mu)$. Given the sets $V_\Gamma^{\leq\mu}$, $V_\Gamma^{>\mu}, V_\Gamma^{\geq\mu}$ and $V_\Gamma^{<\mu}$ we can extract the sets $V_\Gamma^{>\mu}, V_\Gamma^{=\mu}$ and $V_\Gamma^{<\mu}$.
All values μ' in S^Γ are of the form $\frac{y}{z}$. For those values we can determine whether $v \in V_\Gamma^{\geq\mu'}$ by applying the algorithm for threshold mean-payoff parity games on $\Gamma' = (V, E, \langle V_2, V_1\rangle, w', p)$ where $w'(e) = w(e) \cdot z$ for all $e \in E$ with the mean-payoff parity objectives $\phi = \mathit{Parity}_\Gamma(p) \cap \mathit{MeanPayoff}_\Gamma(y)$. Note that in the worst case, the weight function w' of Γ' is in $\mathcal{O}(nW)$.

Dichotomic Search. Let Γ be a mean-payoff parity game. The dichotomic search algorithm is recursive algorithm initialized with $\Gamma_0 = \Gamma$ and $S_0 = S^\Gamma$. In recursive call i the following steps are executed:

1. Let $r_i = \min(S_i)$ and $s_i = max(S_i)$.
2. Determine a_1, the largest element in S_i less than or equal to $\frac{r_i+s_i}{2}$ and a_2, the smallest element in S_i greater than or equal to $\frac{r_i+s_i}{2}$.
3. Determine the partitions $V_{\Gamma_i}^{<a_1}$, $V_{\Gamma_i}^{=a_1}$, $V_{\Gamma_i}^{=a_2}$, $V_{\Gamma_i}^{>a_2}$ using the key observation.
4. For all $v \in V_{\Gamma_i}^{=a_1}$ set the value to a_1, for all $v \in V_{\Gamma_i}^{=a_2}$ set the value to a_2 and set the value to $-\infty$ for all vertices v which are not in any set calculated in step 3.
5. Recurse upon $\Gamma_i \restriction V_{\Gamma_i}^{<a_1}$ and $\Gamma_i \restriction V_{\Gamma_i}^{>a_2}$.

Correctness. Let Γ be a mean-payoff parity game. We prove that the dichotomic search algorithm correctly calculates $val_\Gamma(MPP)(v)$ for all $v \in V$. The algorithm is initialized with Γ and S^Γ. By Lemma 7 the values of the vertices $v \in V$ are in the set S^Γ. Because we perform a binary search over the set S^Γ we can guarantee the termination of the algorithm. Notice that we need to show that the values calculated in the subgames constructed in step 4 are identical to the values in the original game. Then correctness follows immediately by our key observation and because we perform a binary search over the set S^Γ.

▶ **Lemma 9.** *Given a mean-payoff parity game Γ and $\mu \in \mathbb{Q}$, let $\Gamma' = \Gamma \restriction V_\Gamma^{>\mu}$ and $\Gamma'' = \Gamma \restriction V_\Gamma^{<\mu}$. For all $v \in V_\Gamma^{>\mu}$, we have $val_{\Gamma'}(MPP)(v) = val_\Gamma(MPP)(v)$ and for all $v \in V_\Gamma^{<\mu}$, we have $val_{\Gamma''}(MPP)(v) = val_\Gamma(MPP)(v)$.*

Proof. Let $v \in V_\Gamma^{>\mu}$ be arbitrary. We will prove $val_{\Gamma'}(MPP)(v) = val_\Gamma(MPP)(v)$ by showing the following two cases:

- $val_{\Gamma'}(MPP)(v) \leq val_\Gamma(MPP)(v)$: Note that there can be no player-2 vertex in $V_\Gamma^{>\mu}$ with an edge to $V_\Gamma^{\leq\mu}$. Thus we cut away only edges of player-1 vertices in Γ'. Consequently player-1 has less choices in Γ' than in Γ at each of her vertices. Thus $val_{\Gamma'}(MPP)(v) \leq val_\Gamma(MPP)(v)$ holds.
- $val_{\Gamma'}(MPP)(v) \geq val_\Gamma(MPP)(v)$: Let σ_1 be an optimal strategy for player 1 and let σ_2 be an optimal strategy for player 2 which both exist by [11]. We will show that σ_1 produces plays with vertices in $V_\Gamma^{>\mu}$ only, if it starts from v. For the sake of contradiction assume that a play $\rho = \mathit{outcome}(v, \sigma_1, \sigma_2)$ contains a vertex $v^* \in V_\Gamma^{\leq\mu}$. Notice that there are no player-2 vertices in $V_\Gamma^{>\mu}$ with edges to $V_\Gamma^{\leq\mu}$. Thus σ_1 chose a successor vertex in $V_\Gamma^{\leq\mu}$. But when ρ ends up in $V_\Gamma^{\leq\mu}$ the optimal player-2 strategy σ_2 can guarantee that $MPP_\Gamma(w, p, \rho) \leq \mu$ by the definition of $V_\Gamma^{\leq\mu}$. There is a strategy to keep the value of the play starting at v greater than μ by the definition of $V_\Gamma^{>\mu}$. Thus any play ρ leading to $V_\Gamma^{\leq\mu}$ by σ_1 is not optimal which is a contradiction to our assumption. Consequently $val_{\Gamma'}(MPP)(v) \geq val_\Gamma(MPP)(v)$ follows.

The fact that for all $v \in V_\Gamma^{<\mu}$, we have $val_{\Gamma''}(MPP)(v) = val_\Gamma(MPP)(v)$ follows by a symmetric argument. ◀

Running Time. The running time of the dichotomic search is $\mathcal{O}(n \cdot \log(nW) \cdot \mathsf{TH})$ where TH is the running time of an algorithm for the threshold mean-payoff parity problem. The additional factor n comes from rescaling the weights of the mean-payoff parity game Γ which is described in the key observation. The factor $\mathcal{O}(\log(nW))$ is from using binary search on S as $|S| = \mathcal{O}(n^2 \cdot W)$.

▶ **Theorem 10.** *Given a game graph Γ and an algorithm that solves the threshold mean-payoff parity problem in $\mathcal{O}(\mathsf{TH})$, the value function of Γ can be computed in time $\mathcal{O}(n \cdot \log(nW) \cdot \mathsf{TH})$.*

As a corollary of the above theorem and Theorem 6, the value function for mean-payoff parity games can be computed in $\mathcal{O}(n^d \cdot m \cdot W \cdot \log(nW))$ time.

6 Conclusion

In this paper we present faster algorithms for mean-payoff parity games. Our most interesting results are for mean-payoff Büchi and mean-payoff coBüchi games, which are the base cases. For threshold mean-payoff Büchi and mean-payoff coBüchi games, our bound $\mathcal{O}(n \cdot m \cdot W)$ matches the current best-known bound for mean-payoff games. For the value problem, we show the dichotomic search approach of [5] for mean-payoff games can be generalized to mean-payoff parity games. This gives an additional multiplicative factor of $n \cdot \log(nW)$ as compared to the threshold problem. A recent work [14] shows that the value problem for mean-payoff objective can be solved with a multiplicative factor n compared to the threshold objective (i.e., it shaves of the log factor). An interesting question is whether the approach of [14] can be generalized to mean-payoff parity games.

References

1. R. Bloem, K. Chatterjee, T. A. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *Proc. of CAV*, LNCS 5643, pages 140–156. Springer, 2009.
2. Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, Georg Hofferek, Barbara Jobstmann, Bettina Könighofer, and Robert Könighofer. Synthesizing robust systems. *Acta Inf.*, 51(3-4):193–220, 2014.
3. P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In *Proc. of FORMATS*, LNCS 5215, pages 33–47. Springer, 2008.
4. P. Bouyer, N. Markey, J. Olschewski, and M. Ummels. Measuring permissiveness in parity games: Mean-payoff parity games revisited. In *Proc. of ATVA*, LNCS 6996, pages 135–149. Springer, 2011.
5. L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J. F. Raskin. Faster algorithms for mean-payoff games. *Form. Methods Syst. Des.*, 38(2):97–118, April 2011. doi:10.1007/s10703-010-0105-x.
6. J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the AMS*, 138:295–311, 1969.
7. P. Cerný, K. Chatterjee, T. A. Henzinger, A. Radhakrishna, and R. Singh. Quantitative synthesis for concurrent programs. In *Proc. of CAV*, LNCS 6806, pages 243–259. Springer, 2011.
8. A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource interfaces. In *Proc. of EMSOFT*, LNCS 2855, pages 117–133. Springer, 2003.
9. K. Chatterjee and L. Doyen. Energy parity games. In *Proc. of ICALP: Automata, Languages and Programming (B)*, LNCS 6199, pages 599–610. Springer, 2010.
10. K. Chatterjee and L. Doyen. Energy and mean-payoff parity Markov decision processes. In *Proc. of MFCS*, LNCS 6907, pages 206–218. Springer, 2011.

11 K. Chatterjee, T. A. Henzinger, and M. Jurdziński. Mean-payoff parity games. In *Proc. of LICS*, pages 178–187. IEEE Computer Society, 2005.

12 Krishnendu Chatterjee and Laurent Doyen. Games and markov decision processes with mean-payoff parity and energy parity objectives. In *MEMICS*, pages 37–46, 2011.

13 Krishnendu Chatterjee, Laurent Doyen, Hugo Gimbert, and Youssouf Oualhadj. Perfect-information stochastic mean-payoff parity games. In *FOSSACS*, pages 210–225, 2014.

14 Carlo Comin and Romeo Rizzi. Improved pseudo-polynomial bound for the value problem and optimal strategy synthesis in mean payoff games. *Algorithmica*, 77(4):995–1021, 2017. doi:10.1007/s00453-016-0123-1.

15 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979. doi:10.1007/BF01768705.

16 Y. M. Lifshits and D. S. Pavlov. Potential theory for mean payoff games. *Journal of Mathematical Sciences*, 145(3):4967–4974, 2007. doi:10.1007/s10958-007-0331-y.

17 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of POPL*, pages 179–190. ACM Press, 1989.

18 P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

19 W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3, Beyond Words, chapter 7, pages 389–455. Springer, 1997.

Attainable Values of Reset Thresholds

Michalina Dżyga[1], Robert Ferens[*2], Vladimir V. Gusev[†3], and Marek Szykuła[‡4]

1 **Institute of Computer Science, University of Wrocław, Joliot-Curie 15, Wrocław, Poland**
misia.sieradzka@interia.pl

2 **Institute of Computer Science, University of Wrocław, Joliot-Curie 15, Wrocław, Poland**
robert.ferens@interia.pl

3 **ICTEAM Institute, Université catholique de Louvain, Louvain-la-Neuve, Belgium, and Institute of Natural Sciences and Mathematics, Ural Federal University, Ekaterinburg, Russia**
vl.gusev@gmail.com

4 **Institute of Computer Science, University of Wrocław, Joliot-Curie 15, Wrocław, Poland**
msz@cs.uni.wroc.pl

Abstract

An automaton is synchronizing if there exists a word that sends all states of the automaton to a single state. The reset threshold is the length of the shortest such word. We study the set RT_n of attainable reset thresholds by automata with n states. Relying on constructions of digraphs with known local exponents we show that the intervals $\left[1,(n^2-3n+4)/2\right]$ and $[(p-1)(q-1),p(q-2)+n-q+1]$, where $2\le p<q\le n, p+q>n, \gcd(p,q)=1$, belong to RT_n, even if restrict our attention to strongly connected automata. Moreover, we prove that in this case the smallest value that does not belong to RT_n is at least $n^2-O(n^{1.7625}\log n/\log\log n)$. This value is increased further assuming certain conjectures about the gaps between consecutive prime numbers. We also show that any value smaller than $\frac{n(n-1)}{2}$ is attainable by an automaton with a sink state and any value smaller than $n^2-O(n^{1.5})$ is attainable in general case. Furthermore, we solve the problem of existence of slowly synchronizing automata over an arbitrarily large alphabet, by presenting for every fixed size of the alphabet an infinite series of irreducibly synchronizing automata with the reset threshold $n^2-O(n)$.

1998 ACM Subject Classification F.1.1 Models of Computation, G.2.2 Graph Theory

Keywords and phrases Černý conjecture, exponent, primitive digraph, reset word, synchronizing automaton

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.40

[*] R. Ferens was supported in part by the National Science Centre, Poland under project number 2014/15/B/ST6/00615.

[†] V. Gusev was supported by the French Community of Belgium, by the IAP network DYSCO, RFBR grant no. 16-01-00795, Russian Ministry of Education and Science project no. 1.3253.2017, and the Competitiveness Enhancement Program of Ural Federal University.

[‡] M. Szykuła was supported in part by the National Science Centre, Poland under project number 2014/15/B/ST6/00615.

1 Introduction

Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a *deterministic complete finite (semi)automaton* with the set of states Q, the alphabet Σ, and the transition function $\delta\colon Q \times \Sigma \to Q$. We denote the image of a state q under the action of a word w as $q \cdot w$. Automaton $\mathscr{A}$ is called *synchronizing* if there exist a word w and a state f such that for every state $q \in Q$ we have $q \cdot w = f$. Every such word w is called *synchronizing* (or *reset*) word for $\mathscr{A}$. The length of the shortest synchronizing word of $\mathscr{A}$ is called the *reset threshold* and is denoted by $\mathrm{rt}(\mathscr{A})$. This notion can be extended to subsets: for a subset $S \subseteq Q$, $\mathrm{rt}(\mathscr{A}, S)$ is the length of the shortest word w such that $|\delta(S, w)| = 1$.

Synchronizing automata constitute a well-studied class of automata with applications to group theory [4], coding theory [6, Chapter 10], industrial automation, matrix and control theories [7], etc. A brief survey of the theory of synchronizing automata can be found in [18, 33].

The reset threshold is one of the most important characteristics of a synchronizing automaton. One can compute a synchronizing word following a greedy strategy in polynomial-time, but finding the shortest such word is hard. The problem of deciding whether $\mathrm{rt}(\mathscr{A}) = k$ for a given automaton $\mathscr{A}$ and an integer k encoded in binary is DP-complete [23]. Moreover, unless P = NP, there is no polynomial-time algorithm computing the reset threshold with the approximation error $O(n^{1-\varepsilon})$ for a fixed $\varepsilon > 0$ [13]. Furthermore, widely used greedy algorithms have approximation error $\Omega(n)$, where n is the number of states in a given automaton [2].

One of the most famous open problems in automata theory is to determine general bounds on the reset thresholds of automata with n states. *The Černý conjecture* states that the reset threshold of an automaton is at most $(n-1)^2$ [10, 11]. Furthermore, this bound is reached by the Černý automaton $\mathscr{C}_n$ with n states, see [33, p. 18]. Despite intensive efforts of researchers and confirmation of this conjecture in various special classes of automata the best upper bound obtained so far is $(15617n^3 + 7500n^2 + 9375n - 31250)/93750$ [29].

The difficulty of the Černý conjecture led to a large number of attempts to disprove it by means of a counterexample obtained via computational experiments [31, 3, 20]. Although, no counterexample was ever found, several interesting observations were made. For example, it was noted that there are no synchronizing automata with n states and two letters with the reset threshold in the range $[n^2 - 3n + 5, (n-1)^2 - 1]$ for $n = 7, \ldots, 12$. Afterwards, a few more potential "gaps" were identified. It leads us in turn to the following general question that we address in our paper:

> What is the set RT_n of possible reset thresholds
> for synchronizing automata with n states?

Clearly, a complete answer to this question is out of reach for the state of the art techniques as the Černý conjecture is merely about an upper bound on RT_n. Nevertheless, it is possible to describe certain values belonging to RT_n by presenting synchronizing automata with known reset thresholds. This problem is also not trivial as one has to prove that a constructed automaton is not only synchronized by a word of the claimed length, but also demonstrate that any shorter word is not synchronizing.

Additional interest to results of this kind comes from the following reasons. Concrete examples of synchronizing automata with known reset thresholds shed light on the phenomenon of synchronization giving hints on potential proof of the Černý conjecture. Furthermore, the availability of hands on examples is important for the evaluation of new ideas and algorithms.

Synchronizing automata with the reset threshold close to $(n-1)^2$, so called slowly synchronizing automata, are especially important as they almost never appear in random samples of synchronizing automata. Moreover, the ideas used in construction of synchronizing automata with known reset thresholds could also lead to interesting connections between different fields. For example, a new line of research devoted to the interplay between synchronizing automata and primitive matrices was started in [3], see [14] for recent results. Part of our current work can be seen as a continuation of this line of research.

Over the years a large number of synchronizing automata with known reset thresholds were presented in the literature. It can be either sporadic examples possessing interesting properties [31, 33] or infinite series of automata designed to give a lower bound on the largest reset threshold among synchronizing automata belonging to a special class. For example, constructions of automata with the sink state are given in [1, 24, 26], Eulerian automata in [15, 30], automata with the reset threshold close to $(n-1)^2$ [3]. Multi-parametric series of automata with the aim of covering RT_n were presented in [16].

Our contributions. In the present paper we significantly expand the list of values known to belong to RT_n. Moreover, we present non-trivial lower bounds on the smallest value that does not belong to RT_n. Our approach can be summarized as follows. Recall that a subset of states $\mathcal{C}$ of the automaton $\mathscr{A}$ is a *sink* if $\delta(q,\ell) \in \mathcal{C}$ for every $q \in \mathcal{C}$ and $\ell \in \Sigma$. A classical observation states that a synchronizing automaton has the unique strongly connected sink component $\mathcal{C}$. Furthermore, the process of synchronization can be performed in the following manner: initially, one applies a word u such that $Q \cdot u \subseteq \mathcal{C}$, i.e. the automaton is brought to $\mathcal{C}$, afterwards, one applies v such that $|\mathcal{C} \cdot v| = 1$. Therefore, we consider three natural cases.

Case (i) strongly connected automata. In Section 2 we construct synchronizing automata belonging to this class with known reset thresholds based on examples of digraphs with known local exponents [28]. We show that the intervals $\left[1, (n^2-3n+4)/2\right]$ and $[(p-1)(q-1), p(q-2)+n-q+1]$, where $2 \leq p < q \leq n, p+q > n, \gcd(p,q) = 1$, belong to the set of attainable reset thresholds by n-state automata. The method that we use to convert digraphs into synchronizing automata is based on techniques recently introduced in [7, 14].

Let $\mathrm{gt}_{sc}(n)$ be the smallest value that does not serve as the reset threshold of a strongly connected automaton with n states. In Section 3 we show that $\mathrm{gt}_{sc}(n)$ is at least $n^2 - \tilde{O}(n^{1.7625})$, where $\tilde{O}(f(n))$ is the shorthand for $O(f(n)\log^k(f(n))$ for some k. Moreover, we strengthen this bound conditioning on the validity of classical conjectures related to the distribution of prime numbers. If the Riemann hypothesis is true, then $\mathrm{gt}_{sc}(n) \geq n^2 - \tilde{O}(n^{1.75})$. If the Cramer's conjecture is true, then $\mathrm{gt}_{sc}(n) \geq n^2 - \tilde{O}(n^{1.5})$. Our proofs are based on rigorous analysis of the overlaps of the aforementioned intervals. Similar intervals often appear in index set problems about digraphs and Boolean matrices, so our techniques can be applied to them as well [22].

Case (ii) automata with the sink state. The reset threshold of automata belonging to this class is bounded by $\frac{n(n-1)}{2}$, moreover, there exists a series of n-state automata reaching this bound [26]. In Section 4 we generalize this series to prove that for every $1 \leq \ell \leq \frac{n(n-1)}{2}$ there exists an n-state automaton with sink and the reset threshold equal to ℓ.

Case (iii) automata without restrictions. In Section 5 we utilize ideas of the previous cases to show that for every $1 \leq \ell \leq n^2 - O(n^{1.5})$ there exists an n-state synchronizing automaton with reset threshold equal to ℓ. This result does not depend on the number theoretic conjectures.

The number of letters of automata constructed in the previous cases grows with the number of states. In Section 6 we aim to get a better understanding on how the number of letters influences the set of possible reset thresholds. To avoid trivial cases we focus on *irreducibly synchronizing* automata, i.e. automata that become non-synchronizing after the removal of any letter. We resolve an open problem asking whether for each fixed size of the alphabet there is a series of irreducibly synchronizing n-state automata with the reset threshold $n^2 - O(n)$. Previously, such automata were known only over 2-letter and 3-letter alphabets [3, 21]. Namely, we construct infinite (in n and k) series of automata $\mathscr{M}_{n,k}$, $\mathscr{M}'_{n,k}$ with n states and k letters such that $rt(\mathscr{M}_{n,k}) = n^2 - (k+3)n + 2k + 3$ and $rt(\mathscr{M}'_{n,k}) = n^2 - (k+3)n + 2k + 4$. These examples can be also seen as a formal bound to the following common empirical statement: synchronizing automata with large number of letters have relatively small reset thresholds (due to a large number of possibilities at every step of synchronization).

2 Strongly connected automata

A digraph G is *primitive* if there exists a positive integer t such that for every pair of vertices u, v of G there exists a walk from u to v of length exactly t. The smallest such t is called the exponent of G and denoted by $\exp(G)$. A survey of results about this classical notion can be found in [9, Chapter 3.5].

The notion of local exponent was introduced in [8]. The *local exponent* of G at a vertex u, denoted by $\exp_G(u)$ or $\exp(u)$, is the smallest t such that for every vertex v of G there is a walk from u to v of length exactly t. Let $V = \{1, 2, \ldots, n\}$. We will always assume that the vertices are reordered so that $\exp_G(1) \leq \exp_G(2) \leq \ldots \leq \exp_G(n)$.

The behavior of the exponents and the local exponents of digraphs with n vertices gained a lot of attention in literature. Let $\mathrm{ES}_n(1)$ be the set of possible first local exponents of all digraphs with n vertices, i.e. $\mathrm{ES}_n(1) = \{\exp_G(1) \mid G = (V, E), |V| = n, G \text{ is primitive}\}$.

▶ **Theorem 1** ([28, Theorem 9])**.**

$$\mathrm{ES}_n(1) = \left[1, \frac{n^2 - 3n + 4}{2}\right] \bigcup \bigcup_{(p,q)\in L(n)} [(p-1)(q-1), p(q-2) + n - q + 1],$$

where $L(n) = \{(p, q) : 2 \leq p < q \leq n, p + q > n, \gcd(p, q) = 1\}$.

We will rely on Theorem 1 to construct synchronizing automata with known reset thresholds. The proof of the following proposition is based on the "determinization" procedure appearing in [7, 14, 17].

▶ **Proposition 2.** *Let $G(V, E)$ be a primitive n-vertex digraph. Then there exists a synchronizing n-state automaton $\mathscr{A}$ such that* $\mathrm{rt}(\mathscr{A}) = \exp_G(1)$.

Proof. The automaton $\mathscr{A}$ is constructed as follows. The set of states is equal to $V = \{1, 2, \ldots, n\}$. For every choice of states $s_1, s_2, \ldots, s_n \in V$ such that $(s_1, 1), (s_2, 2), \ldots, (s_n, n) \in E$ we add the letter $(s_1, \ldots, s_n)$ with the action $\delta(j, (s_1, \ldots, s_n)) = s_j$ for every $j \in V$.

We need to show that $\mathscr{A}$ is synchronizing and $\mathrm{rt}(\mathscr{A}) = \exp_G(1)$. Since G is primitive and every vertex is reachable from 1 in $\exp_G(1)$ steps there exists a sequence of n-tuples $(v_1^{(1)} = 1, v_2^{(1)} = 1, \ldots, v_n^{(1)} = 1), (v_1^{(2)}, v_2^{(2)}, \ldots, v_n^{(2)}), \ldots, (v_1^{(t)} = 1, v_2^{(t)} = 2, \ldots, v_n^{(t)} = n)$ of length $t = \exp_G(1)$ and such that for every $2 \leq i \leq t$, $1 \leq j \leq n$ we have $(v_j^{(i-1)}, v_j^{(i)}) \in E$. Furthermore, we can assume that if $v_j^{(i)} = v_k^{(i)}$ for some i, j, k then $v_j^{(\ell)} = v_k^{(\ell)}$ for all $\ell \leq i$.

Indeed, by substituting the value of $v_j^{(\ell)}$ to $v_k^{(\ell)}$ for all $\ell \leq i$ we will obtain a sequence satisfying all of the aforementioned properties.

Observe now that for every $i \geq 2$ there is a letter of $\mathscr{A}$ that maps the tuple $(v_1^{(i)}, v_2^{(i)}, \ldots, v_n^{(i)})$ to $(v_1^{(i-1)}, v_2^{(i-1)}, \ldots, v_n^{(i-1)})$, namely, any letter $(s_1, \ldots, s_n)$ satisfying $s_{v_j^{(i)}} = v_j^{(i-1)}$ for all $1 \leq j \leq n$. Thus, the sequence of tuples can be seen as an application of a word w mapping the set $\{1, 2, \ldots n\}$ to $\{1\}$. In other words, w is a synchronizing word of length $\exp_G(1)$.

It remains to note that $\mathrm{rt}(\mathscr{A}) \geq \exp_G(1)$. Indeed, every synchronizing word w mapping V to $\{f\}$ labels walks leading from every state to f; moreover, the edges of $\mathscr{A}$ are the inverted edges of G. Thus, every vertex of G is reachable from f in $|w|$ steps. Since $\exp_G(1)$ is the smallest number with this property, we have $|w| \geq \exp_G(1)$. ◀

By combining Theorem 1 and Proposition 2 we obtain the main result of this section:

▶ **Theorem 3.** *For every n, $\mathrm{ES}_n(1) \subset \mathrm{RT}_n$. Furthermore, it remains true even in the case of strongly connected automata.*

▶ **Remark 4.** *Clearly, $\mathrm{RT}_n \not\subset \mathrm{ES}_n(1)$, since the largest local 1-exponent is at most n^2-3n+3, by Theorem 1 (originally [27, Theorem 2.1]), while the Černý series of automata has the reset threshold equal to $(n-1)^2$.*

Proposition 2 is also tightly connected to the Hybrid Černý-Road Coloring conjecture [3, Conjecture 2]. Let G be a digraph with the set of edges E, and Σ be a finite alphabet. A *coloring* of G is an arbitrary deterministic finite state automaton obtained by distributing letters of Σ over the edges E. Note that G typically has a large number of colorings. The celebrated Road Coloring Theorem states that every primitive digraph with out-degree k has a synchronizing coloring with k letters [32]. The Hybrid Černý-Road Coloring conjecture states that such synchronizing coloring can always be found with the reset threshold at most $n^2 - 3n + 3$.

▶ **Corollary 5.** *Let G be a primitive digraph with n vertices. There exists an alphabet Σ and a coloring $\mathscr{A}$ of G with Σ such that $\mathscr{A}$ is a synchronizing automaton and $\mathrm{rt}(\mathscr{A}) \leq n^2 - 3n + 3$. In other words, the Hybrid Černý-Road Coloring conjecture holds true if we are allowed to use an alphabet of arbitrary size.*

Proof. The proof of Proposition 2 describes a procedure to derive a synchronizing coloring of $\overleftarrow{G}$ – the digraph obtained by reversing all the edges of G, with the reset threshold equal to $\exp_G(1)$. Since $\exp_G(1)$ is bounded by $n^2 - 3n + 3$ by Theorem 1, the corollary follows. ◀

3 Lower bounds on the smallest unattainable value

In this section we will derive a lower bound on the smallest value that is not a reset threshold of a strongly connected n-state automaton. The proof of main theorem is based on the following number theoretic result.

Let $g(x)$ be the maximal difference (*prime gap*) between any prime number $p \leq x$ and the next prime number. It is known that $g(x) \leq x^{0.525}$ when x is large enough [5].

Let $\omega(x)$ be the maximal number of distinct prime divisors of any number $i \leq x$. It is known that $\omega(x) \leq 1.38402 \log x / \log\log x$ for $x \geq 3$ [25].

▶ **Theorem 6.** *For n large enough, the function*

$$f(n) = 6n + 4n \cdot g(3\sqrt{n}) \cdot \omega(n) \cdot \big(g(3\sqrt{n}) \cdot \omega(n) + 6\sqrt{n}\big)$$

satisfies the following equation:

$$\left[\left\lceil\frac{n^2}{3}\right\rceil, \lfloor n^2 - f(n)\rfloor\right] \subseteq \bigcup_{(p,q)\in L(n)} [(p-1)(q-1), (p-1)(q-1)+n-p],$$

where $L(n) = \{(p,q) \mid 2 \le p < q \le n, p+q > n, \gcd(p,q) = 1\}$.

▶ **Corollary 7.** *Using the known upper bound* $g(x) \le x^{0.525}$ [5] *we obtain*

$$f(n) = O(n^{1.7625}\log n/\log\log n).$$

Moreover, if we assume the Riemann hypothesis, which implies $g(x) \in O(\sqrt{x}\log x)$ [12]*, we get*

$$f(n) = O(n^{1.75}\log^2 n/\log\log n).$$

If we assume the Cramer's conjecture $g(x) \in O(\log^2 x)$ [12]*, we get*

$$f(n) = O(n^{1.5}\log^3 n/\log\log n).$$

Before proving Theorem 6 we will derive the main result of this section:

▶ **Theorem 8.** *Let* $\mathrm{gt}_{sc}(n) \ge 1$ *be the smallest unattainable value by the reset thresholds of* n*-state strongly connected synchronizing automata. Then* $\mathrm{gt}_{sc}(n)$ *grows at least as fast as* $n^2 - O(f(n))$*, where* $f(n)$ *is the function from Corollary 7. In particular,*

$$\lim_{n\to\infty} \mathrm{gt}_{sc}(n)/n^2 = 1.$$

Proof. By Theorems 1 and 3 and Proposition 2 we know that for every integer in $[1, (n^2-3n+4)/2]$ there exists an automaton with the reset threshold equal to it. Also, from Theorem 6 and Corollary 7 we obtain the same result for the interval $[\left\lceil\frac{n^2}{3}\right\rceil, n^2 - O(n^{1.7625}\log n/\log\log n)]$. In other words, $\mathrm{gt}_{sc}(n)$ is at least $n^2 - O(f(n))$. Since $O(n^{1.7625}\log n/\log\log n)$ grows strictly slower than n^2, we have $\mathrm{gt}_{sc}(n)/n^2$ tending to 1 when $n \to \infty$. ◀

▶ **Remark 9.** *Theorem 6 establishes a lower bound on the least value that does not belong to* ES_n *as well.*

Now we are going to prove Theorem 6. Let n be large enough. Let x be any integer from $[\lceil\frac{n^2}{3}\rceil, \lfloor n^2 - f(n)\rfloor]$. We will show that x falls in the interval $[(p-1)(q-1), (p-1)(q-1)+n-p]$ of some $(p,q) \in L(n)$. Let k be the smallest odd integer such that $x \le k^2$.

$$(k-2)^2 < x \le k^2.$$

We define

$$S = \{p \in [3, n-k-1] \mid (p \text{ is prime}) \wedge p \nmid (k+1)\}.$$

▶ **Lemma 10.** *For every* $s \in S$*, each pair* $(p,q) = (k+1-s, k+1+s)$ *is in* $L(n)$*.*

Proof. It is enough to check all conditions for (p,q) in $L(n)$. For the first condition $p = k+1-s \ge k+1-(n-k-1) = 2(k+1)-n \ge 2\sqrt{x}-n$. Because $3x \ge n^2$ (so $\sqrt{x} \ge n/\sqrt{3}$), we have $2\sqrt{x}-n \ge (2/\sqrt{3}-1)n$. So for $n \ge 13$, we have $p \ge 2$.

The condition $p < q$ is obvious, and $q = k+1+s \le k+1+n-k-1 = n$.

The next condition states that $p+q > n$ and it is clear, since $p+q = 2(k+1) \geq 2\sqrt{x} \geq 2n/\sqrt{3} > n$.

Now we show that p and q are coprime. Let d be a non-trivial common divisor of p and q. Then also d divides $q-p = 2s$. Since s is prime, d is either 2 or s. But $q = k+1+s$ is odd, since $k+1$ is even and s is odd. Also s cannot divide $q = k+1+s$, since by definition of s, s is not a divisor of $k+1$. Thus, p and q are coprime. ◀

Let $\text{next}_S(i)$ be the smallest number in S greater than i (if it exists). Let $\text{next}_P(i)$ be similarly defined for the set of prime numbers. For all i, we have $\text{next}_P(i) - i \leq g(i)$.

To simplify formulas we define:

$$b_k(s) = (k-s)(k+s) = k^2 - s^2$$
$$e_k(s) = (k-s)(k+s) + n - (k+1) + s = k^2 - s^2 + n - (k+1) + s$$
$$I_k(s) = [b_k(s), e_k(s)] = [k^2 - s^2, k^2 - s^2 + n - (k+1) + s]$$

Notice that $I_k(s)$ is the interval from Theorem 6 with $(p,q) = (k+1-s, k+1+s)$, which according to Lemma 10 is a pair from $L(n)$ for every $s \in S$.

Our plan is as follows. We will show the existence of $s_{max} \in S$ such that $b_k(s_{max}) \leq (k-2)^2$ and for every two consecutive elements $s, \text{next}_S(s) \in S \cap [3, s_{max}]$, the intervals $I_k(s)$ and $I_k(\text{next}_S(s))$ overlap. Since $b_k(s_{max}) \leq (k-2)^2$ and $k^2 \leq e_k(\text{next}_S(0))$, it will prove that the intervals

$$I_k(\text{next}_S(0)), \ldots, I_k(s), I_k(\text{next}_S(s)), \ldots, I_k(s_{max})$$

cover all integers from $[(k-2)^2, k^2]$. So in particular x will be covered.

▶ **Lemma 11.** *For every $\varepsilon \in [0,1)$, when n is large enough, there is $s_{max} \in S$ satisfying $b_k(s_{max}) \leq (k-2)^2$ and $2\sqrt{k-1} \leq s_{max} \leq (2+\varepsilon)\sqrt{k-1}$.*

Proof. Let s_{max} be the smallest prime number $\geq 2\sqrt{k-1}$ and such that $s_{max} \nmid k+1$.

We show that $s_{max} \in S$. Let p be the first prime number $\geq 2\sqrt{k-1}$. If $p \neq s_{max}$, then $p \mid k+1$. But since $p \geq 2\sqrt{k-1}$ there is no other prime number $p' > p$ such that $p' \mid (k+1)$, as otherwise $p' \cdot p \geq 4(k-1) > k+1$ (for $k \geq 2$). Hence, $s_{max} = \text{next}_P(p) > p$. So for both cases we get the upper bound:

$$\begin{aligned} s_{max} &\leq \text{next}_P(\text{next}_P(2\sqrt{k-1})) \\ &\leq 2\sqrt{k-1} + 2g(2\sqrt{k-1} + g(2\sqrt{k-1})). \end{aligned}$$

According to the bound $g(n) < n^\theta$ for $\theta = 0.525$ [5],

$$\begin{aligned} s_{max} &\leq 2\sqrt{k-1} + 2(4\sqrt{k-1})^\theta \\ &\leq 2\sqrt{k-1} + 2 \cdot 4^\theta \cdot (k-1)^{\theta/2} \\ &\leq 2\sqrt{k-1}(1 + 4^\theta \cdot (k-1)^{(\theta-1)/2}). \end{aligned}$$

Since $k \geq n/\sqrt{3}$, we have

$$0 \leq \lim_{k\to\infty} 4^\theta (k-1)^{0.5(\theta-1)} \leq \lim_{n\to\infty} 4^\theta (n/\sqrt{3} - 1)^{0.5(\theta-1)} = 0.$$

So for n large enough, we obtain the upper bound:

$$s_{max} \leq (2+\varepsilon)\sqrt{k-1}.$$

Observe that:

$$\begin{aligned} k^2 &= (k-2)^2 + 4k - 4 \\ &< x + 6k \\ &< n^2 - f(n) + 6n \\ &< n^2 - 24n\sqrt{n} \\ &< n^2 - 24n\sqrt{n} + (12\sqrt{n})^2, \end{aligned}$$

and so $k < n - 12\sqrt{n}$. From this we obtain:

$$\begin{aligned} s_{max} &\leq (2+\varepsilon)\sqrt{k-1} \\ &< 3\sqrt{n-1} \\ &< 12\sqrt{n} - 1 \\ &\leq n - (n - 12\sqrt{n}) - 1 \\ &\leq n - k - 1. \end{aligned}$$

Thus, $s_{max} \in [3, n-k-1]$ and therefore is in S.
Finally, we have:

$$\begin{aligned} s_{max} &\geq 2\sqrt{k-1} \\ s_{max}^2 &\geq 4(k-1) \\ k^2 - s_{max}^2 &\leq k^2 - 4k + 4 \\ b_k(s_{max}) &\leq (k-2)^2. \end{aligned}$$

◀

▶ **Lemma 12.** *For every $s \in S \cap [3, s_{max} - 1]$, the intervals $I_k(s)$ and $I_k(\mathrm{next}_S(s))$ overlap, that is, $b_k(s) \leq e_k(\mathrm{next}_S(s))$. Moreover $k^2 \leq e_k(\mathrm{next}_S(0))$.*

Proof. We have $\mathrm{next}_S(s) \leq s_{max}$, because $s_{max} \in S$. Notice that

$$\mathrm{next}_S(s) \leq s + g(s_{max}) \cdot \big(\omega(k+1) + 1\big), \tag{1}$$

because the number of distinct prime odd divisors of $k+1$ is at most $\omega(k+1)$, and the gap between every two consecutive of the prime divisors is bounded from above by $g(s_{max})$.

Observe that:

$$\begin{aligned} (k+1)^2 &< (k-2)^2 + 6k - 3 \\ &< x + 6k \\ &< n^2 - f(n) + 6n \\ &\leq n^2 - 4n \cdot g(3\sqrt{n}) \cdot \omega(n) \cdot \big(g(3\sqrt{n}) \cdot \omega(n) + 6\sqrt{n}\big) \\ &< n^2 - 4n \cdot g(3\sqrt{n}) \cdot \omega(n) \cdot \big(g(3\sqrt{n}) \cdot \omega(n) + 6\sqrt{n}\big) + \\ &\quad + \Big(2 \cdot g(3\sqrt{n}) \cdot \omega(n) \cdot \big(g(3\sqrt{n}) \cdot \omega(n) + 6\sqrt{n}\big)\Big)^2, \end{aligned}$$

and so

$$k + 1 < n - 2 \cdot g(3\sqrt{n}) \cdot \omega(n) \cdot \big(g(3\sqrt{n}) \cdot \omega(n) + 6\sqrt{n}\big). \tag{2}$$

Then we have:

$$\begin{aligned} e_k\big(\mathrm{next}_S(s)\big) - b_k(s) &= -\mathrm{next}_S(s)^2 + n - (k+1) + \mathrm{next}_S(s) + s^2 \\ &> n - (k+1) + s^2 - \mathrm{next}_S(s)^2. \end{aligned} \tag{3}$$

Using (1) we obtain:

$$\begin{aligned} e_k\big(\mathrm{next}_S(s)\big) - b_k(s) &> n - (k+1) + s^2 - \Big(s + \big(\omega(k+1)+1\big)g(s_{max})\Big)^2 \\ &= n - (k+1) - g(s_{max}) \cdot \big(\omega(k+1)+1\big)\Big(g(s_{max}) \cdot \big(\omega(k+1)+1\big) + 2s\Big) \end{aligned}$$

using (2) we obtain:

$$\begin{aligned} &\geq n - \Big(n - 2 \cdot g(3\sqrt{n}) \cdot \omega(n) \cdot \big(g(3\sqrt{n}) \cdot \omega(n) + 6\sqrt{n}\big)\Big) \\ &\quad - g(s_{max}) \cdot \big(\omega(k+1)+1\big) \cdot \Big(g(s_{max}) \cdot \big(\omega(k+1)+1\big) + 2s\Big) \\ &\geq g(3\sqrt{n}) \cdot \sqrt{2} \cdot \omega(n) \cdot \big(g(3\sqrt{n}) \cdot \sqrt{2} \cdot \omega(n) + 6\sqrt{n}\big) \\ &\quad - g(s_{max}) \cdot \big(\omega(k+1)+1\big) \cdot \Big(g(s_{max}) \cdot \big(\omega(k+1)+1\big) + 2s\Big) \end{aligned} \tag{4}$$

Observe that $3\sqrt{n} > (2+\varepsilon)\sqrt{k-1} \geq s_{max}$ from Lemma 11. Also, since $n > k+1$ (by (2)), for $n \geq 30$ we have $\omega(n) \geq 3$ and so $\sqrt{2} \cdot \omega(n) \geq \omega(k+1)+1$. Because functions g and ω are monotonic, we finally obtain that (4) ≥ 0.

Consider $e_k\big(\mathrm{next}_S(0)\big) - k^2$. Then it is equal to (3) with $s = 0$. All the above inequalities remains unchanged and then $k^2 \leq e_k(\mathrm{next}_S(0))$. ◀

Summarizing, for each $x \in [n^2/3, n^2 - f(n)]$ we find some k such that $x \in [(k-2)^2, k^2]$. From Lemma 11 we know that there exists $s_{max} \in S$ such that $b_k(s_{max}) \leq (k-2)^2$. From Lemma 12 we also get $k^2 \leq e_k\big(\mathrm{next}_S(0)\big)$, and the intervals for consecutive values from S between $\mathrm{next}_S(0)$ and s_{max} overlap. Hence, in particular, x belongs to some interval $I_k(s_x)$ where $s_x \in S$, $\mathrm{next}_S(0) \leq s_x \leq s_{max}$. Lemma 10 ensures that the pair $(k+1-s_x, k+1+s_x)$ belongs to $L(n)$ from the Theorem 6, so x is covered.

4 Automata with a sink state

It is known that if a synchronizing automaton $\mathscr{A}_n$ has a sink state, then $\mathrm{rt}(\mathscr{A}_n) \leq \frac{(n-1)n}{2}$ [26]. We show that reset thresholds of automata with a sink state cover all values in $[1, \ldots, \frac{(n-1)n}{2}]$, at least if the size of the alphabet can be quadratic in n.

We construct a class of automata as follows. Let $T_n(V, E)$ be an undirected tree with $n = |V|$ vertices and the root r. We construct $\mathscr{A}(T_n) = (V, E, \delta)$, and the action of every letter $\{v_1, v_2\}$ is defined as follows:

$$\delta(v, \{v_1, v_2\}) = \begin{cases} v_1 & \text{if } v = v_2 \text{ and } r \notin \{v_1, v_2\}, \\ v_2 & \text{if } v = v_1 \text{ and } r \notin \{v_1, v_2\}, \\ r & \text{if } v, r \in \{v_1, v_2\}, \\ v & \text{otherwise.} \end{cases}$$

For $v \in V$ let $d(v)$ denote the distance in T_n from v to the root; hence $d(r) = 0$. For a subset $S \subseteq V$ we define:

$$U(T_n, S) = \sum_{q \in S} d(q).$$

▶ **Lemma 13.** *For every rooted tree $T_n(V, E)$ and every subset $S \subseteq V$ we have:* $\mathrm{rt}(\mathscr{A}(T_n), S) = U(T_n, S)$.

Proof. The proof follows easily by induction on $U(T_n, S)$. The case $U(T_n, S) = 0$ is trivial, since it must be that $S = \{r\}$. Let $U(T_n, S) \geq 1$ and assume that the claim holds for all S' such that $U(T_n, S') < U(T_n, S)$. Then observe that the action of any letter $\{v_1, v_2\} \in E$ increases the value of $U(T_n, S)$ by 1, decreases the $U(T_n, S)$ by 1, or does not change it. Moreover, we can always find such $\{v_1, v_2\}$ that decreases the value by 1, i.e. when $v_1 \in S$ and $v_2 \notin S$, or $v_1 = r$ and $v_2 \in S$. Thus the claim follows inductively and the automaton is synchronizing. ◀

▶ **Proposition 14.** *Given $n \geq 2$, for every $k \in [1, \ldots, \frac{(n-1)n}{2}]$ there exists an automaton $\mathscr{A}_n$ with a sink state and* $\mathrm{rt}(\mathscr{A}) = k$.

Proof. First, we consider the case of $k \geq n-1$. For each such k we construct a tree $T_n(V, E)$ with $U(T_n, V) = k$ and the claimed result follows by Lemma 13. We proceed by induction:

1. the *star* ($E = \{\{v, r\} \mid v \in V\}$) is suitable for the case of $n-1$;
2. let $T_n(V, E)$ be a tree such that $U(T_n, V) = k$. If the height of T_n is equal to $n-1$, then T_n is a *path* graph and $U(T_n, V) = \frac{(n-1)n}{2}$. Thus, we can assume that the height of T_n is smaller than $n-1$. We construct a new tree $T'_n(V', E)$ such that $U(T_n, V') = k+1$ in the following manner. Observe, that there exist two vertices v_1, v_2, both distinct from the root, that are leaves; without loss of generality let $d(v_1) \leq d(v_2)$. We will remove v_1, which does not change $d(v_2)$. Afterwards, we can find a vertex v_3 such that $d(v_3) = d(v_1)$, and attach v'_1 to v_3 as a leaf; thus $d(v'_1) = d(v_2) + 1$ and $U(T_n, V') = k+1$.

Hence, we can construct the tree for any k starting from $n-1$.

Finally, for $k \in \{1, \ldots, n-2\}$ let $\mathscr{A}_n(V, E, \delta)$ be the automaton with $V = \{r, v_1, \ldots, v_{n-1}\}$, $E = \{e_1, \ldots, e_k\}$ and the transitions defined by

$$\delta(v_i, e_j) = \begin{cases} r & \text{if } i = j \text{ or } i > k, \\ v_i & \text{otherwise.} \end{cases}$$

Obviously, to synchronize $\mathscr{A}_n$ it is sufficient and necessary to use a word with every letter from E. ◀

5 General case

Let $\mathscr{C}_m(Q, \Sigma, \delta_{\mathscr{C}})$ be the Černý automaton with m states [10]: $Q = \{q_0, \ldots, q_{m-1}\}$, $\Sigma = \{a, b\}$, $\delta_{\mathscr{C}}(q_{m-1}, a) = q_0$ and $\delta_{\mathscr{C}}(q_i, a) = q_i$ for $i \leq m-2$, and $\delta_{\mathscr{C}}(q_i, b) = (q_i + 1) \bmod m$.

Let $T_{m'}(V, E)$ be a tree with m' vertices and let $\mathscr{A}(T_{m'}) = (V, E, \delta_{\mathscr{A}})$ be the automaton from Section 4.

Given $\mathscr{C}_m$ and $\mathscr{A}(T_{m'})$ we construct the joint automaton $\mathscr{B}_n$ with $n = m + m' - 1$ states. This is done by union of both automata, while identifying the sink state r of $\mathscr{A}(T_{m'})$ with state q_0 of $\mathscr{C}_m$ and extending the actions of the letters for the states of the other automaton to identity.

Formally, assume that the alphabets Σ and E are disjoint, r is the sink state of $\mathscr{A}(T_{m'})$, and let $\mathscr{B}_n = (Q \cup V \setminus \{r\}, \Sigma \cup E, \delta)$, where δ is defined as follows:

$$\delta(q, a) = \begin{cases} \delta_{\mathscr{C}}(q, a) & \text{if } q \in Q \text{ and } a \in \Sigma, \\ \delta_{\mathscr{A}}(q, a) & \text{if } q \in V \setminus \{r\},\ a \in E, \text{ and } \delta_{\mathscr{A}}(q, a) \neq r, \\ q_0 & \text{if } q \in V \setminus \{r\},\ a \in E, \text{ and } \delta_{\mathscr{A}}(q, a) = r, \\ q & \text{if } q \in Q \text{ and } a \in E, \text{ or } q \in V \setminus \{r\} \text{ and } a \in \Sigma. \end{cases}$$

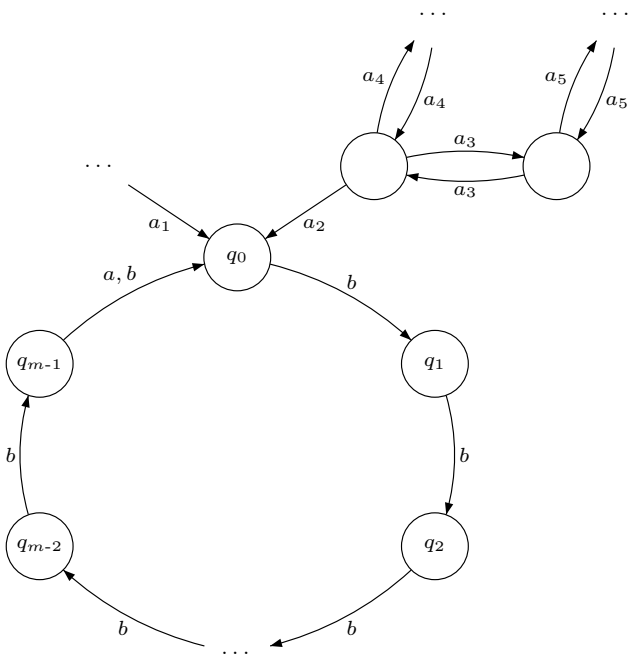


Figure 1 The automaton $\mathscr{B}_n$ from Section 5 formed from $\mathscr{C}_m$ and some $\mathscr{A}(T_{m'})$. The omitted transitions are self-loops.

The scheme of this construction is illustrated in Fig. 1.

▶ **Lemma 15.** *For all $m, m' \geq 1$ we have*

$$\mathrm{rt}(\mathscr{B}_n) = \mathrm{rt}(\mathscr{C}_m) + \mathrm{rt}(\mathscr{A}(T_{m'})).$$

Proof. Note that in the degenerated case $m' = 1$ this trivially holds, and suppose $m' \geq 2$.

To synchronize $\mathrm{rt}(\mathscr{B}_n)$ is it enough to apply the word $w'w$, where w' is a word synchronizing $\mathscr{A}(T_{m'})$ and w is a word synchronizing $\mathscr{C}_m$.

To show that this is a shortest possibility, let w be a synchronizing word for $\mathscr{B}_n$. Then w contain two disjoint subsequences of letters from Σ and from E, respectively. These subsequences synchronizes respectively $\mathscr{C}_m$ and $\mathscr{A}(T_{m'})$, which proves the lower bound. ◀

Arbitrary choice of m and m' with the constraint $n = m + m' - 1$ provides enough flexibility that finally leads to

▶ **Theorem 16.** *For every $k \leq n^2 - O(n^{3/2})$ there exists a synchronizing automaton with reset threshold k.*

Proof. Since $\mathrm{rt}(\mathscr{C}_m) = (m-1)^2$ and $\mathrm{rt}(\mathscr{A}(T_{m'})) \in \left[1, \ldots, \frac{(m'-1)m'}{2}\right]$ by Proposition 14, given m and m' we can construct an automaton with any value of reset threshold in $\left[(m-1)^2 + 1, (m-1)^2 + \frac{(m'-1)m'}{2}\right]$.

Since $n = m + m' - 1$ ($m' = n - m + 1$) we can cover all intervals

$$\left[(m-1)^2 + 1, (m-1)^2 + \frac{(n-m)(n-m+1)}{2}\right]$$

for all $1 \leq m \leq n$. Let g, $2 \leq g \leq (n-1)^2$, be the smallest number such that is not in the interval for some m. Consider the interval lying just before g ($g \geq 2$ so it exists), that is, let m

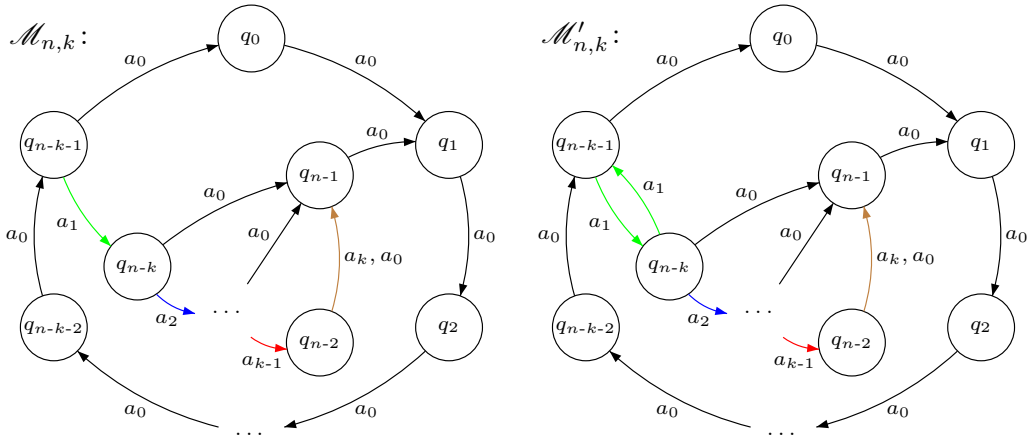


Figure 2 Automata $\mathscr{M}_{n,k}$ and $\mathscr{M}'_{n,k}$. The omitted transitions are self-loops.

be the largest number such that $g > (m-1)^2+((n-m)(n-m+1))/2$. Then the interval for $m+1$ must begin after g, so $g < m^2+1$. Hence $(m-1)^2+((n-m)(n-m+1))/2+1 \leq m^2$, which solved yields

$$m \geq \left(2n - \sqrt{16n+9} + 5\right)/2 = n - O(\sqrt{n}).$$

So $(m-1)^2 + (n-m)(n-m+1)/2 = n^2 - O(n^{3/2})$. ◀

6 Irreducibly synchronizing automata with large reset thresholds

Let $k \geq 1$ and $n \geq k+3$. Let $Q_n = q_0, \ldots, q_{n-1}$, and $\Sigma_k = a_0, \ldots, a_k$. We define the automaton $\mathscr{M}_{n,k} = (Q_n, \Sigma_k, \delta_{n,k})$, illustrated in Fig. 2 (left), with the transition function $\delta_{n,k}$ defined as follows:

$$\delta_{n,k}(q_i, a_0) = \begin{cases} q_{i+1} & \text{if } i \leq n-k-2 \\ q_0 & \text{if } i = n-k-1 \\ q_{n-1} & \text{if } n-k \leq i \leq n-2 \\ q_1 & \text{if } i = n-1, \end{cases}$$

and for $j \geq 1$

$$\delta_{n,k}(q_i, a_j) = \begin{cases} q_{i+1} & \text{if } i = n-k-2+j \\ q_i & \text{otherwise.} \end{cases}$$

We also define the variation $\mathscr{M}'_{n,k}(Q_n, \Sigma_k, \delta_{n,k})$, illustrated in Fig. 2 (right), of $\mathscr{M}_{n,k}$, where $\delta'_{n,k}$ is defined as follows:

$$\delta'_{n,k}(q_i, a_j) = \begin{cases} q_{n-k-1} & \text{if } i = n-k \text{ and } j = 1 \\ \delta_{n,k}(q_i, a_j) & \text{otherwise.} \end{cases}$$

▶ **Theorem 17.** *The automaton $\mathscr{M}_{n,k}$ is irreducibly synchronizing and has reset threshold*

$$n^2 - (k+3)n + 2k + 3.$$

Proof (sketch). The word $a_1 a_2 \cdots a_k (a_0^{n-k-1} a_1 \cdots a_k)^{n-k-2} a_0$ synchronizes $\mathscr{M}_{n,k}$ to the state q_1 and has length $n^2 - (k+3)n + 2k + 3$.

To show that this is the reset threshold, we use the backward tracing technique (cf. [19, Lemma 2], [21, 30]). The idea is to keep track, for $i = 1, 2, \ldots$, of families L_i of subsets of Q_n

that are preimages of a singleton under the action of a word of length i. Hence, L_i contains in particular all subsets that are compressible to a singleton by a word of length i. The smallest i such that $Q_n \in L_i$ is the length of the shortest reset words that synchronize to the singletons from L_0. Further, from each L_i we can exclude *visited* subsets, which are those being a proper subset of another subset from L_i or being a subset (non necessarily proper) of a subset from some $L_0, \ldots, L_{i-1}$ ([19, Lemma 2]). ◀

▶ **Theorem 18.** *The automaton $\mathscr{M}'_{n,k}$ is irreducibly synchronizing and has reset threshold*

$$n^2 - (k+3)n + 2k + 4.$$

References

1 D. Ananichev. A new lower bound for reset threshold of synchronizing automata with sink state. *ArXiv e-prints*, January 2017. `arXiv:1701.07954`.

2 D. S. Ananichev and V. V. Gusev. Approximation of reset thresholds with greedy algorithms. *Fundam. Inform.*, 145(3):221–227, 2016.

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 J. Araújo, P. J. Cameron, and B. Steinberg. Between primitive and 2-transitive: Synchronization and its friends. *ArXiv e-prints*, November 2015. `arXiv:1511.03184`.

5 R. C. Baker, G. Harman, and J. Pintz. The difference between consecutive primes, II. *Proceedings of the London Mathematical Society*, 83(3):532–562, 2001.

6 J. Berstel, D. Perrin, and C. Reutenauer. *Codes and Automata*. Encyclopedia of Mathematics an. Cambridge University Press, 2010.

7 V. D. Blondel, R. M. Jungers, and A. Olshevsky. On primitivity of sets of matrices. *Automatica*, 61(C):80–88, 2015.

8 R. A. Brualdi and Bolian Liu. Generalized exponents of primitive directed graphs. *Journal of Graph Theory*, 14(4):483–499, 1990.

9 R. A. Brualdi and H. Ryser. *Combinatorial Matrix Theory*. Cambridge University Press, 1991.

10 J. Černý. Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied*, 14(3):208–216, 1964. In Slovak.

11 J. Černý, A. Pirická, and B. Rosenauerová. On directable automata. *Kybernetica*, 7:289–298, 1971.

12 H. Crámer. On the order of magnitude of the difference between consecutive prime numbers. *Acta Arithmetica*, 2:23–46, 1936.

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 B. Gerencsér, V. V. Gusev, and R. M. Jungers. Primitive sets of nonnegative matrices and synchronizing automata. `https://arxiv.org/abs/1602.07556`, 2016.

15 V. Gusev. Lower Bounds for the Length of Reset Words in Eulerian Automata. *International Journal of Foundations of Computer Science*, 24(2), 2013.

16 V. V. Gusev and E. V. Pribavkina. Reset thresholds of automata with two cycle lengths. *International Journal of Foundations of Computer Science*, 26(07):953–966, 2015.

17 M. Ito and K. Shikishima-Tsuji. Some results on directable automata. In J. Karhumäki, H. Maurer, G. Păun, and G. Rozenberg, editors, *Theory Is Forever: Essays Dedicated to Arto Salomaa on the Occasion of His 70th Birthday*, pages 125–133. Springer, 2004.

18 J. Kari and M. V. Volkov. Černý's conjecture and the road coloring problem. In *Handbook of Automata*. European Science Foundation, 2013.

19 A. Kisielewicz, J. Kowalski, and M. Szykuła. Computing the shortest reset words of synchronizing automata. *Journal of Combinatorial Optimization*, 29(1):88–124, 2015.

20 A. Kisielewicz, J. Kowalski, and M. Szykuła. Experiments with Synchronizing Automata. In *Implementation and Application of Automata*, volume 9705 of *LNCS*, pages 176–188. Springer, 2016.

21 A. Kisielewicz and M. Szykuła. Synchronizing Automata with Extremal Properties. In *Mathematical Foundations of Computer Science*, volume 9234 of *LNCS*, pages 331–343. Springer, 2015.

22 Q. Li and J. Shao. The index set problem for boolean (or nonnegative) matrices. *Discrete Mathematics*, 123(1):75–92, 1993.

23 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.

24 E. V. Pribavkina. Slowly synchronizing automata with zero and noncomplete sets. *Mathematical Notes*, 90(3):411, 2011.

25 G. Robin. Estimation de la fonction de Tchebychef ϕ sur le k-iéme nombre premier et grandes valeurs de la fonction $\omega(n)$ nombre de diviseurs premiers de n. *Acta Arithmetica*, 42(4):367–389, 1983.

26 I. K. Rystsov. Reset words for commutative and solvable automata. *Theoretical Computer Science*, 172(1-2):273–279, 1997.

27 J. Shao, J. Wang, and G. Li. Generalized primitive exponents with the complete characterizations of the extreme digraphs. *Chinese J. Contemp. Math.*, 15(4):317–324, 1994.

28 J. Shen and S. Neufeld. Local exponents of primitive digraphs. *Linear Algebra and its Applications*, 268:117–129, 1998.

29 M. Szykuła. Improving the upper bound on the length of the shortest reset words. *ArXiv e-prints*, February 2017. `arXiv:1702.05455`.

30 M. Szykuła and V. Vorel. An Extremal Series of Eulerian Synchronizing Automata. In *Developments in Language Theory*, LNCS, pages 380–392. Springer, 2016.

31 A. N. Trahtman. An efficient algorithm finds noticeable trends and examples concerning the Černý conjecture. In *Mathematical Foundations of Computer Science*, volume 4162 of *LNCS*, pages 789–800. Springer, 2006.

32 A. N. Trahtman. The Road Coloring Problem. *Israel Journal of Mathematics*, 172(1):51–60, 2009.

33 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.

Lower Bounds and PIT for Non-Commutative Arithmetic Circuits with Restricted Parse Trees*

Guillaume Lagarde[1], Nutan Limaye[2], and Srikanth Srinivasan[3]

1 **Univ Paris Diderot, Sorbonne Paris Cité, IRIF, UMR 7089 CNRS, Paris, France**
guillaume.lagarde@irif.fr

2 **Department of Computer Science and Engineering, IIT Bombay, Mumbai, India**
nutan@cse.iitb.ac.in

3 **Department of Mathematics, IIT Bombay, Mumbai, India**
srikanth@math.iitb.ac.in

Abstract

We investigate the power of *Non-commutative Arithmetic Circuits*, which compute polynomials over the free non-commutative polynomial ring $\mathbb{F}\langle x_1, \ldots, x_N \rangle$, where variables do not commute. We consider circuits that are restricted in the ways in which they can compute monomials: this can be seen as restricting the families of *parse trees* that appear in the circuit. Such restrictions capture essentially all non-commutative circuit models for which lower bounds are known. We prove several results about such circuits.

1. We show explicit exponential lower bounds for circuits with up to an exponential number of parse trees, strengthening the work of Lagarde, Malod, and Perifel (ECCC 2016), who prove such a result for *Unique Parse Tree* (UPT) circuits which have a single parse tree.
2. We show explicit exponential lower bounds for circuits whose parse trees are rotations of a single tree. This simultaneously generalizes recent lower bounds of Limaye, Malod, and Srinivasan (Theory of Computing 2016) and the above lower bounds of Lagarde et al., which are known to be incomparable.
3. We make progress on a question of Nisan (STOC 1991) regarding separating the power of Algebraic Branching Programs (ABPs) and Formulas in the non-commutative setting by showing a tight lower bound of $n^{\Omega(\log d)}$ for any UPT formula computing the product of d $n \times n$ matrices.

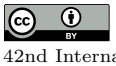
   When $d \leq \log n$, we can also prove superpolynomial lower bounds for formulas with up to $2^{o(d)}$ many parse trees (for computing the same polynomial). Improving this bound to allow for $2^{O(d)}$ trees would yield an unconditional separation between ABPs and Formulas.
4. We give deterministic white-box PIT algorithms for UPT circuits over any field (strengthening a result of Lagarde et al. (2016)) and also for sums of a constant number of UPT circuits with different parse trees.

1998 ACM Subject Classification F.1.2 Modes of Computation

Keywords and phrases Non-commutative Arithemetic circuits, Partial derivatives, restrictions

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.41

* A full version of the paper is available at https://eccc.weizmann.ac.il/report/2017/077/.



1 Introduction

In this paper, we study questions related to Arithmetic Circuits, which are computational devices that use arithmetic operations (such as $+$ and $\times$) to compute multivariate polynomials over a field $\mathbb{F}$. While the more standard work in this area deals with the commutative polynomial ring $\mathbb{F}[x_1, \ldots, x_N]$, there is also a line of research, initiated by Hyafil [12] and Nisan [21], that studies the complexity of computing polynomials from the *non-commutative* polynomial ring $\mathbb{F}\langle x_1, \ldots, x_N \rangle$, where monomials are simply strings over the alphabet $X = \{x_1, \ldots, x_N\}$. The motivation for this is twofold: firstly, the study of polynomial computations over non-commutative algebras (e.g. the algebra of matrices over $\mathbb{F}$) naturally leads to such questions [7, 6], and secondly, computing, say, the Permanent non-commutatively[1] is at least as hard as computing it in the commutative setting, and thus the lower bound question should be easier to tackle in the non-commutative setting.

In an influential result, Nisan [21] justified this by proving exponential lower bounds for non-commutative *formulas*, and more generally *Algebraic Branching Programs* (ABPs), computing the Determinant and Permanent (and also other polynomials). The method used by Nisan to prove this lower bound can also be seen as a precursor to the method of Partial derivatives in Arithmetic circuit complexity (introduced by Nisan and Wigderson [22]), variants of which have been used to prove a large body of lower bound results in the area [22, 25, 9, 14, 16].

While lower bounds for general non-commutative circuits remain elusive, we do have other lower bounds that strengthen Nisan's result. Recently, Malod, along with two of the authors of this paper showed [19] that Nisan's method can be extended to prove lower bounds for *skew circuits*, which are circuits where every $\times$-gate has at most one non-variable input. Also, the first author, Malod and Perifel [18] proved lower bounds for another variant of non-commutative circuits that they defined to be *unambiguous* circuits (which we describe below). While these two results both strengthen Nisan's result, they are incomparable to each other, as shown in [18].

In this paper, we build on the above work to prove lower bounds that generalize these results significantly and also make progress on other problems related to non-commutative circuits. The circuits we consider are restricted in the ways they are allowed to compute monomials. We do this by restricting the "parse trees" that are allowed to appear in the circuits. Informally, the polynomial computed by any arithmetic circuit C can be written down as an exponentially-large sum of subcircuits, each of which contains only multiplication gates and hence computes a single monomial[2]; each such subcircuit gives rise to a tree, which we call a *parse tree of* C (see [18] and references therein for the background of parse trees), that tells us how the monomial was computed. For example, for the circuit C in Figure 1, the monomial $x_1x_2x_3x_4$ may be computed in C as $(x_1 \cdot x_2) \cdot (x_3 \cdot x_4)$ or as $(x_1 \cdot (x_2 \cdot x_3) \cdot x_4)$, each of which comes from a parse tree of C.

All the non-commutative circuit classes for which we know lower bounds can be defined by restrictions on the parse trees that appear in them. ABPs are circuits where all parse trees are left combs (i.e. a tree where every internal node has two children and the left child is always a leaf); skew circuits are equivalent in power to circuits where the parse trees are *twisted* combs (i.e. a tree where every internal node has two children and at least one of the

[1] We can define the Permanent in the non-commutative polynomial ring by ordering the variables in each monomial in the commutative permanent, say, in increasing order of the rows in which they appear.

[2] different subcircuits could compute the same monomial

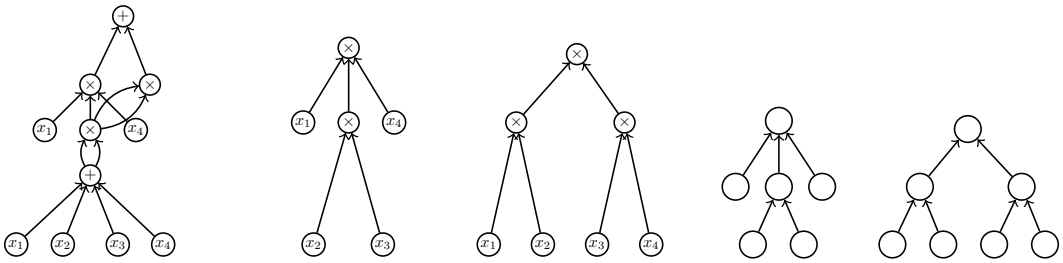


Figure 1 From left to right: a non-commutative arithmetic circuit C; two ways in which the monomial $x_1x_2x_3x_4$ is computed in the circuit; the corresponding parse trees.

two children is always a leaf); and unambiguous circuits (that we will call *Unique Parse Tree* (UPT) circuits below) are defined to be circuits that have only one parse tree. It is thus natural to consider other restrictions on the structure of the parse trees that appear in a circuit. We prove the following results about such circuits.

- We prove lower bounds for circuits that only contain a few different parse trees. The motivation for this is the lower bound of [18] for the case of circuits with a single parse tree and a construction in [19] that shows that poly(N, d)-sized circuits with $\exp(\Omega(d))$ many parse trees[3] evade all currently known techniques for proving lower bounds for non-commutative circuits. We show explicit exponential lower bound for circuits with up to $\exp(d^{\Omega(1)})$ parse trees (Theorem 13).
- We also consider structural restrictions on the collections of parse trees that appear in our circuits. As mentioned above, skew circuits are circuits where all parse trees are twisted combs, which can be seen as trees obtained by starting with a left comb (which defines an ABP) and successively applying rotations to the internal nodes that swap the children. We say that a circuit C is *rotation Unique Parse tree* (rotUPT) if there is a single tree T such that all the parse trees of C can be obtained as rotations of T.[4]
  We show an explicit exponential lower bound for rotUPT circuits (Theorem 17). Note that this result simultaneously generalizes the skew circuit lower bound of Limaye et al. [19] as well as the UPT circuit lower bound of Lagarde et al. [18].
- We consider the problem of separating ABPs from formulas, which was posed by Nisan [21] and is the non-commutative arithmetic analogue of separating NL from NC^1. Equivalently, this is the question of whether (an entry of) the product of d $n \times n$ matrices, all of whose entries are distinct variables, can be computed by a poly(n, d)-sized non-commutative formula. The standard divide-and-conquer approach yields, for every even Δ, a non-commutative formula of depth Δ and size $n^{O(\Delta d^{2/\Delta})}$ computing this polynomial and a size $n^{O(\log d)}$ formula in general. Further, these formulas can be seen to have a *unique* parse tree (i.e. they are UPT).
  We show that this upper bound is nearly tight for UPT formulas and every choice of Δ by showing a lower bound of $n^{\Omega(\Delta d^{1/\lfloor \Delta/2 \rfloor})}$.[5] (Theorem 18). In particular, our result implies that any UPT formula for this polynomial must have size $n^{\Omega(\log d)}$. We can extend this (Theorem 19) to prove a superpolynomial lower bound even in the case that the formula has at most k parse trees, where $k = 2^{o(d)}$ (however, for this result, we need the assumption that $d \leq \log n$).

[3] A close look at the circuits in [19] indicates that just about all parse trees of fan-in 2 appear in these circuits.
[4] There can be $\exp(\Omega(d))$ of these, as in the case of skew circuits.
[5] Our bounds are actually better stated in terms of the $\times$-*depth* of the formula.

- Finally, we consider the Polynomial Identity Testing (PIT) problem for non-commutative circuits with restricted parse trees. Lagarde et al. [18] show that deterministic PIT algorithms for UPT circuits can be obtained by adapting a PIT algorithm for ABPs due to Arvind, Joglekar and Srinivasan [3]. However, this technique only works over fields of characteristic zero. Here, we give a straightforward adaptation of an older PIT algorithm of Raz and Shpilka [24] (also for non-commutative ABPs) to show that PIT for UPT circuits can be solved in deterministic polynomial time over all fields (Theorem 20). We also consider circuits that are sums of UPT circuits (with possibly different parse trees). By using ideas from the work of Gurjar, Korwar, Saxena and Thierauf [10], we show that PIT for a sum of constant number of UPT circuits can be solved in deterministic polynomial time over any field (Theorem 21).

For lack of space, many of the proofs of the above statements have been omitted from this extended abstract. We refer the reader to the full version [17] for detailed proofs.

Related work. Hrubeš, Wigderson and Yehudayoff [11] initiated a study of the asymptotics of the classical *sum-of-squares* problem in mathematics and showed that a suitable result in this direction would yield strong lower bounds against general non-commutative circuits. While this line of work is currently the only feasible attack on the problem of general circuit lower bounds, we do not yet have any lower bounds using this technique.

Nisan and Wigderson [22] prove results that imply[6] some lower bounds for UPT formulas computing iterated matrix product. For depth-3 formulas, they prove an optimal n^d bound on computing the product of d $n \times n$ matrices. For depths $\Delta > 3$ though, the lower bound is only $\exp(\Theta(d^{1/\Delta}))$ and thus does not yield anything non-trivial when Δ approaches $\log d$. Indeed, the proof method of this result in [22] is not sensitive to the value of n and holds for any $n \geq 2$. Such a method cannot yield non-trivial lower bounds for general formulas since we do have $\mathrm{poly}(d)$-sized formulas in the setting when $n = O(1)$.

The results of Kayal, Saha, and Saptharishi [15] and Fournier, Limaye, Malod, and Srinivasan [8] together also prove a superpolynomial lower bound on the size of *regular* formulas (defined by [15]) computing the product of d $n \times n$ matrices in the commutative setting. While these formulas (in the non-commutative setting) are definitely UPT, the converse is not true.

Arvind, Mukhopadhyay and Raja [4] and Arvind, Joglekar, Mukhopadhyay and Raja [2] have some recent work on PIT algorithms for general non-commutative circuits that run in time *polylogarithmic* in the degree of the circuit and polynomial in the size of the circuit. Our results are incomparable with theirs, since our algorithms run in time polynomial in *both* degree and size but are deterministic, whereas the algorithms of [4, 2] are faster (especially in terms of degree) but *randomized*.

An earlier manuscript of Arvind and Raja [5] contains a claim that the PIT problem for non-commutative skew circuits has a deterministic polynomial time algorithm, but the proof is unfortunately flawed.[7]

Techniques. The techniques used to prove the lower bounds in this paper are generalizations of the techniques of Hyafil [12] and Nisan [21]. Given a homogeneous polynomial $f \in \mathbb{F}\langle X \rangle$ of degree d, we associate with it an $N^{d/2} \times N^{d/2}$ matrix whose rows and columns are labelled

[6] The results of [22] in fact hold in the stronger commutative *set-multilinear* setting.
[7] Private communication with the authors.

by monomials (i.e. strings over X) m of degree $d/2$ each. Nisan [21] considers the matrix $M[f]$ where the (m_1, m_2)th entry is the coefficient of the monomial $m_1 m_2$ in f. In [19, 18], along with our co-authors, we considered the more general family of matrices $M_Y[f]$ where $Y \subseteq [d]$ is of size $d/2$ and the (m_1, m_2)th entry of $M_Y[f]$ is the coefficient of the monomial m such that the projection of m to the locations in Y gives m_1 and the locations outside Y give m_2.

This is the general technique we use in this paper as well, though choosing the right Y requires some work. In the lower bound for circuits with few parse trees, it is chosen at random (in a similar spirit to a multilinear lower bound of Raz [23]). In the lower bound for rotUPT circuits, it is chosen in a way that depends on the structure of the parse trees in the circuit (combining the approaches of [19, 18]). In the separation of UPT formulas from ABPs, it is applied (after a suitable restriction) in a way that keeps the iterated matrix product polynomial high rank but reduces the rank of the UPT formula.

For the PIT algorithm for sums of UPT circuits, we use an observation of Gurjar et al. [10] (also see [21, 24]) that any polynomial P that has a small ABP has a small set of *characterizing identitites* such that $Q = P$ iff Q satisfies these identities. We are able to show (using a suitable decomposition lemma of [18]) that a similar fact is also true more generally in the case that P has a small UPT circuit. If Q also has a small UPT circuit, then checking these identities for Q reduces to a PIT circuit for a single UPT circuit, for which we already have algorithms. In this way, given two UPT circuits (with different parse trees) computing P, Q, we can check if $P - Q = 0$. Extending this idea exactly as in [10], we can efficiently check if the sum of any small number of UPT circuits is 0.

2 Preliminaries

We refer the reader to the survey [26] for standard definitions regarding arithmetic circuits.

2.1 Non-commutative polynomials

Throughout, we use $X = \{x_1, \ldots, x_N\}$ to denote the set of variables. We work over the *non-commutative* ring of polynomials $\mathbb{F}\langle X\rangle$ where monomials are *strings* over the alphabet X: for example, x_1x_2 and x_2x_1 are distinct monomials in this ring. For $d \in \mathbb{N}$, we use $\mathcal{M}_d(X)$ to denote the set of monomials (i.e. strings) over the variables in X of degree exactly d.

For $i, j \in \mathbb{N}$, we define $[i, j]$ to be the set $\{i, i+1, \ldots, j\}$ (the set is empty if $i > j$). We also use the standard notation $[i]$ to denote the set $[1, i]$.

Given homogeneous polynomials $g, h \in \mathbb{F}\langle X\rangle$ of degrees d_g and d_h respectively and an integer $j \in [0, d_h]$, we define the *j-product of g and h* – denoted $g \times_j h$ – as follows:

- When g and h are monomials, then we can factor h uniquely as a product of two monomials h_1h_2 such that $\deg(h_1) = j$ and $\deg(h_2) = d_h - j$. In this case, we define $g \times_j h$ to be $h_1 \cdot g \cdot h_2$.
- The map is extended bilinearly to general homogeneous polynomials g, h. Formally, let g, h be general homogeneous polynomials, where $g = \sum_\ell g_\ell$, $h = \sum_i h_i$ and g_ℓ, h_i are monomials of g, h respectively. For $j \in [0, d_h]$, each h_i can be factored uniquely into h_{i_1}, h_{i_2} such that $\deg(h_{i_1}) = j$ and $\deg(h_{i_2}) = d_h - j$. And $g \times_j h$ is defined to be $\sum_i \sum_\ell h_{i_1} g_\ell h_{i_2}$.

Note that $g \times_0 h$ and $g \times_{d_h} h$ are just the products $g \cdot h$ and $h \cdot g$ respectively.

2.2 The partial derivative matrix

Here we recall some definitions from [21] and [19]. Let Π denote a partition of $[d]$ given by an ordered pair (Y, Z), where $Y \subseteq [d]$ and $Z = [d] \setminus Y$. In what follows we only use ordered partitions of sets into two parts. We say that such a Π is *balanced* if $|Y| = |Z| = d/2$.

Given a monomial m of degree d and a set $W \subseteq [d]$, we use m_W to denote the monomial of degree $|W|$ obtained by keeping exactly the variables in the locations indexed by W.

▶ **Definition 1** (Partial Derivative matrix)**.** Let $f \in \mathbb{F}\langle X\rangle$ be a homogeneous polynomial of degree d. Given a partition $\Pi = (Y, Z)$ of $[d]$, we define an $N^{|Y|} \times N^{|Z|}$ matrix $M[f, \Pi]$ with entries from $\mathbb{F}$ as follows: the rows of $M[f, \Pi]$ are labelled by monomials from $\mathcal{M}_{|Y|}(X)$ and the columns by elements of $\mathcal{M}_{|Z|}(X)$. Let $m' \in \mathcal{M}_{|Y|}(X)$ and $m'' \in \mathcal{M}_{|Z|}(X)$; the (m', m'')th entry of $M[f, \Pi]$ is the coefficient in the polynomial f of the unique monomial m such that $m_Y = m'$ and $m_Z = m''$.

We will use the rank of the matrix $M[f, \Pi]$ – denoted $\text{rank}(f, \Pi)$ – as a measure of the complexity of f . Note that since the rank of the matrix is at most the number of rows, we have for any $f \in \mathbb{F}\langle X\rangle$ $\text{rank}(f, \Pi) \leq N^{|Y|}$.

▶ **Definition 2** (Relative Rank)**.** Let $f \in \mathbb{F}\langle X\rangle$ be a homogeneous polynomial of degree d. For any $Y \subseteq [d]$, we define the *relative rank of f w.r.t.* $\Pi = (Y, Z)$ – denoted $\text{rel-rank}(f, \Pi)$ – to be

$$\text{rel-rank}(f, \Pi) := \frac{\text{rank}(M[f, \Pi])}{N^{|Y|}}.$$

Fix a partition $\Pi = (Y, Z)$ of $[d]$ and two homogeneous polynomials g, h of degrees d_g and d_h respectively. Let $f = g \times_j h$ for some $j \in [0, d_h]$. This induces naturally defined partitions Π_g of $[d_g]$ and Π_h of $[d_h]$ respectively in the following way. Let $I_g = [j + 1, j + d_g]$ and $I_h = [d] \setminus I_g$. We define $\Pi_g = (Y_g, Z_g)$ such that $Y_g = \{j \in [d_g] \mid Y \text{ contains the } j\text{th smallest element of } I_g\}$; $\Pi_h = (Y_h, Z_h)$ is defined similarly with respect to I_h. Let $|Y_g|, |Z_g|, |Y_h|, |Z_h|$ be denoted d'_g, d''_g, d'_h, d''_h respectively.

In the above setting, we have a simple description of the matrix $M[f, \Pi]$ in terms of $M[g, \Pi_g]$ and $M[h, \Pi_h]$. We use the observation that monomials of degree $|Y| = d'_g + d'_h$ are in one-to-one correspondence with pairs (m'_g, m'_h) of degrees d'_g and d'_h respectively (and similarly for monomials of degree $|Z|$). The following appears in [19].

▶ **Lemma 3** (Tensor Lemma)**.** *Say $f = g \times_j h$ as above. Then, $M[f, \Pi] = M[g, \Pi_g] \otimes M[h, \Pi_h]$.*

▶ **Corollary 4.** *Say $f = g \times_j h$ as above. We have $\text{rank}(f, \Pi) = \text{rank}(g, \Pi_g) \cdot \text{rank}(h, \Pi_h)$. In the special case that one of Y_g, Z_g, Y_h, or Z_h is empty, the tensor product is an outer product of two vectors and hence $\text{rank}(f, \Pi) \leq 1$.*

We associate any partition $\Pi = (Y, Z)$ with the string in $\{-1, 1\}^d$ that contains a -1 in exactly the locations indexed by Y. Given partitions $\Pi_1, \Pi_2 \in \{-1, 1\}^d$, we now define $\Delta(\Pi_1, \Pi_2)$ to be the Hamming distance between the two strings or equivalently as $|Y_1 \Delta Y_2|$ where $\Pi_1 = (Y_1, Z_1)$ and $\Pi_2 = (Y_2, Z_2)$.

▶ **Proposition 5.** *Let $f \in \mathbb{F}\langle X\rangle$ be homogeneous of degree d and say $\Pi \in \{-1, 1\}^d$. Then, $\text{rank}(f, \Pi) = \text{rank}(f, -\Pi)$.*

Proof. Follows from the fact that $M[f, -\Pi]$ is the transpose of $M[f, \Pi]$. ◀

▶ **Lemma 6** (Distance lemma)**.** *Let $f \in \mathbb{F}\langle X\rangle$ be homogeneous of degree d and say $\Pi_1, \Pi_2 \in \{-1, 1\}^d$. Then, $\text{rank}(f, \Pi_2) \leq \text{rank}(f, \Pi_1) \cdot N^{\Delta(\Pi_1, \Pi_2)}$.*

Proof. See the full version [17]. ◀

2.3 Standard definitions related to non-commutative circuits

We consider noncommutative arithmetic circuits that compute polynomials over the ring $\mathbb{F}\langle X\rangle$. These are arithmetic circuits where the children of each $\times$ gate are ordered and the polynomial computed by a $\times$ gate is the product of the polynomials computed by its children, where the product is computed in the given order. Further, unless mentioned otherwise, we allow both $+$ and $\times$ gates to have unbounded fan-in and the $+$ gates to compute arbitrary linear combinations of its inputs (the input wires to the $+$ gate are labelled by the coefficients of the linear combination). A noncommutative formula is a circuit where the underlying directed acyclic graph is a rooted tree. The size of an arithmetic circuit or formula is the number of edges or wires in the circuit (which can be assumed to be at least the number of gates in the circuit).

We always assume that the output gate of the circuit is a $+$ gate (possibly of fan-in 1) and that input gates feed into $+$ gates. We also assume that $+$ and $\times$ gates alternate on any path from the output gate to an input gate (some of these gates can have fan-in 1). Any circuit can be converted to one of this form with at most a constant blow-up in size.

Throughout, our circuits and formulas will be *homogeneous* in the following sense. Define the formal degree of a gate as follows: the formal degree of an input gate is 1, the formal degree of a $+$ gate is the maximum of the formal degrees of its children, and that of a $\times$ gate is the sum of the formal degrees of its children. We say that a circuit is homogeneous if each gate computes a homogeneous polynomial and any gate computing a non-zero polynomial computes one of degree equal to the formal degree of the gate. Note, in particular, that every input node is labelled by a variable only (and not by constants from $\mathbb{F}$).

Homogeneity is *not* a strong assumption on the circuit: it is a standard fact that any homogeneous polynomial of degree d computed by a non-commutative circuit of size s can be computed by a homogeneous circuit of size $O(sd^2)$ [11].

We also consider homogeneous Algebraic Branching Programs (ABPs), defined by Nisan [21] in the non-commutative context. We give here a slightly different definition that is equivalent up to polynomial factors.

Assume that $N = n^2 \cdot d$ for positive $n, d \in \mathbb{N}$ and let $\mathrm{IMM}_{n,d}(X)$ denote the following polynomial in N variables (see, e.g. [22]). Assume X is partitioned into d sets of variables $X_1, \ldots, X_d$ of size n^2 each and let $M_1, \ldots, M_d$ be $n \times n$ matrices such that the entries of M_i ($i \in [d]$) are distinct variables in X_i. Let $M = M_1 \cdot M_2 \cdots M_d$; each entry of M is a homogeneous polynomial of degree d from $\mathbb{F}\langle X\rangle$. We define the polynomial $\mathrm{IMM}_{n,d}$ to be the sum of the diagonal entries of M.

A homogeneous ABP for a homogenous polynomial $f \in \mathbb{F}\langle X\rangle$ of degree d is a pair (n_1, ρ) where $n_1 \in \mathbb{N}$ and ρ is a map from $X' = \{x'_1, \ldots, x'_{n_1^2 d}\}$ to homogeneous linear functions from $\mathbb{F}\langle X\rangle$ such that f can be obtained by substituting $\rho(x'_i)$ for each x'_i in the polynomial $\mathrm{IMM}_{n_1,d}(X')$. The parameter n_1 is called the *width* of the ABP.

2.4 Non-commutative circuits with restricted parse trees

In this paper, we study restricted forms of non-commutative arithmetic circuits. The restrictions are defined by the way the circuits are allowed to multiply variables to compute a monomial. To make this precise we need the notion of a parse tree of a circuit, which has been considered in many previous works [13, 1, 20, 18].

Fix a homogeneous non-commutative circuit C. A *parse formula* of C is a *formula* C' obtained by making copies of gates in C as follows:

- Corresponding to the output $+$ gate of C, we add an output $+$ gate to C',

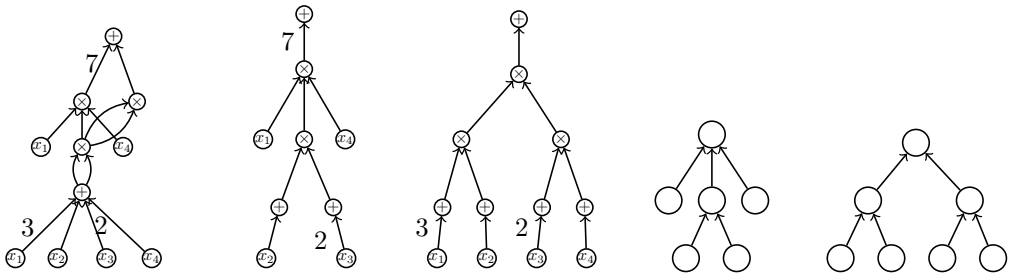


Figure 2 From left to right: a non-commutative arithmetic circuit; two parse formulas in the circuit; the corresponding parse trees. (To simplify the picture, we have not depicted the edges that carry the constant 1. Also we have not introduced + gates between the two layers of × gates; the reader should assume that the edges between the two layers carry + gates of fan-in 1.)

- For every + gate Φ' added to C' corresponding to a + gate Φ in C, we choose exactly one child Ψ of Φ in C and add a copy Ψ' to C' as a child of Φ'. The constant along the wire from Ψ' to Φ' remains the same as in C.
- For every × gate Φ' added to C' corresponding to a × gate Φ in C and *every wire* from a child Ψ to Φ in C, we make a copy of Ψ' to C' and make it a child of Φ.

Any such parse formula C' computes a *monomial* (with a suitable coefficient) and the polynomial computed by C is the sum of all monomials computed by parse formulas C' of C. We define $val(C')$ to be the monomial computed by C'.

A parse tree of C is a rooted, ordered tree obtained by taking a parse formula C' of C, "short circuiting" the + nodes (i.e. we remove the + nodes and connect the edges that were connected to it directly), and deleting all labels of the nodes and the edges of the tree. See Figure 2 for an example. Note that in a homogeneous circuit C, each such tree has exactly d leaves. We say that the tree T is the *shape* of the parse formula C'.

The process that converts the parse formula C' to T associates each internal node of T with a multiplication gate of C' and each leaf of T with an input gate of C'.

Let T be a parse tree of a homogeneous circuit C with d leaves. Given a node $v \in V(T)$, we define the $\deg(v)$ to be the number of leaves in the subtree rooted at v and $\mathrm{pos}(v) :=$ (1+the number of leaves preceding v in an in-order traversal of T). The type of v is defined to be $\mathrm{type}(v) := (\deg(v), \mathrm{pos}(v))$. (The reason for this definition is that in any parse formula C' of shape T, the monomial computed by the multiplication gate or input gate corresponding to v in C' computes a monomial of degree $\deg(v)$ which sits at position $\mathrm{pos}(v)$ w.r.t. the monomial computed by the circuit C'.) We also use $\mathcal{I}(T)$ to denote the set of internal nodes of T and $\mathcal{L}(T)$ to denote the set of leaves of T.

We use $\mathcal{T}(C)$ to denote the set of parse trees that can be obtained from parse formulas of C. We say that a homogeneous non-commutative arithmetic circuit is a *Unique Parse Tree circuit* (or *UPT circuit*) if $|\mathcal{T}(C)| = 1$. More generally if $|\mathcal{T}(C)| \leq k$, we say that C is *k-PT*. Finally, if $\mathcal{T}(C) \subseteq \mathcal{T}$ for some family $\mathcal{T}$ of trees, we say that C is *$\mathcal{T}$-PT*. Similarly, we also define UPT formulas, k-PT formulas and $\mathcal{T}$-PT formulas. If C be a UPT circuit with $\mathcal{T}(C) = \{T\}$, we say that T is the *shape* of the circuit C.

We say that a UPT circuit C is in *normal form* if we can associate with each gate Φ of the circuit a node $v(\Phi) \in V(T)$ such that the following holds: if Φ is an input gate, then $v(\Phi)$ is a leaf; if Φ is a × gate with children $\Psi_1, \ldots, \Psi_t$ (in that order), then the nodes $v(\Psi_1), \ldots, v(\Psi_t)$ are the children of $v(\Phi)$ (in that order); and finally, if Φ is a + gate with children $\Psi_1, \ldots, \Psi_t$ (which are all × or input gates since we assume that + and × gates are

alternating along each input to output path), then $v(\Phi) = v(\Psi_1) = \cdots = v(\Psi_t)$. (Intuitively, what this means is that in any unravelling of a parse formula containing a (multiplication or input) gate Φ to get the parse tree T, the gate Φ always takes the position of node $v(\Phi)$.)

We state below some simple structural facts about UPT circuits. (See [17] for proof.)

▶ **Proposition 7.**
1. *Let C be a UPT* formula. *Then C is in normal form.*
2. *For any UPT circuit C of size s and shape T, there is another UPT circuit C' of size $O(s^2)$ and shape T in normal form computing the same polynomial as C. Further, given C and T, such a C' can be constructed in time* $\mathrm{poly}(s)$.

Let C be either a UPT formula or a UPT circuit of shape T in normal form. We say that a $+$ gate Φ in C is a $(v, +)$ gate if $v(\Phi) = v$. Similarly, we refer to a $\times$ gate Φ in C as a $(v, \times)$ gate if $v(\Phi) = v$. For simplicity of notation, we also refer to an *input* gate Φ as a $(v, \times)$ gate if $v(\Phi) = v$. Note that the output gate is a $(v_0, +)$ gate where v_0 is the root of T.

We now observe that any UPT formula or circuit in normal form can be converted to another (of a possibly different shape) where each multiplication gate has fan-in at most 2.

▶ **Lemma 8.** *Let C be a normal form UPT circuit (resp. formula) of size s and shape T. Then there is a tree T' and normal form UPT circuit (resp. formula) C' of size $O(s)$ and shape T' such that C' computes the same polynomial as C and every multiplication gate in C' has fan-in at most 2. (This implies that every internal node of T' also has fan-in at most 2.) Further, there is a deterministic polynomial-time algorithm, which when given C, computes C' as above.*

Proof. See the full version [17]. ◀

Let C be a UPT circuit of shape T computing a homogeneous polynomial f of degree d. Given any node $u \in V(T)$, we define partition Π_u of $[d]$ so that $\Pi_u = (Y_u, Z_u)$ where $Y_u = \{\mathrm{pos}(v) \mid v \text{ a leaf and descendant of } u\}$.

We will need the following lemma of Lagarde et al. [18].

▶ **Lemma 9** ([18]). *Let C be a normal form UPT circuit of size s computing a homogeneous polynomial $f \in \mathbb{F}\langle X\rangle$ of degree d. Assume that the fan-in of each multiplication gate is bounded by 2. Then, for any $u \in V(T)$, $\mathrm{rank}(f, \Pi_u) \leq s$, where Π_u is as defined above.*

2.5 A polynomial that is full rank w.r.t. all partitions

The following was shown in [19].

▶ **Theorem 10.** *For any even d and any positive $N \in \mathbb{N}$, there is a $q_0(N, d)$ such that the following holds over any field of size at least $q_0(N, d)$. There is an explicit homogeneous polynomial $F_{N,d} \in \mathbb{F}\langle X\rangle$ of degree d such that for any balanced partition $\Pi = (Y, Z)$ of $[d]$, $\mathrm{rank}(f, \Pi) = N^{d/2}$ (equivalently, $\text{rel-rank}(f, \Pi) = 1$). Further, $F_{N,d}$ can be computed by an explicit homogeneous non-commutative arithmetic circuit of size* $\mathrm{poly}(N, d)$.

3 Lower bounds for k-PT circuits

In this section, we show that any k-PT circuit computing a polynomial of degree d where k is subexponential in d cannot compute the polynomial $F_{N,d}$ from Theorem 10. We will show that if both k and the size of the circuit are subexponential in d, then there is a Π such that $\text{rel-rank}(f, \Pi) < 1$.

Our proof is based on the following lemmas.

▶ **Lemma 11.** *Let C be a k-PT circuit (resp. formula) of size s with $\mathcal{T}(C) = \{T_1, \ldots, T_k\}$ computing $f \in \mathbb{F}\langle X\rangle$. Then there exist normal form UPT circuits (resp. formulas) $C_1, \ldots, C_k$ of size at most s^2 each such that $\mathcal{T}(C_i) = \{T_i\}$ and $f = \sum_{i=1}^{k} f_i$, where f_i the polynomial computed by C_i.*

Proof. See the full version [17]. ◀

▶ **Lemma 12.** *Let C be a UPT circuit in normal form over $\mathbb{F}\langle X\rangle$ of size $s = N^c$ and f a homogeneous polynomial of degree d computed by C. Let Π be a uniformly random partition of the variables of $[d]$ into two sets. Then for any parameter $b \in \mathbb{N}$,*

$$\Pr_{\Pi}\left[\operatorname{rank}(f,\Pi) \geq N^{d/2-b}\right] \leq \exp(-\Omega(d/(b+c)^2)).$$

Before proving Lemma 12, let us see that the above lemmas imply the following lower bound for homogeneous non-commutative circuits with few parse trees. Note that when the field $\mathbb{F}$ is large enough, this proves a lower bound for $F_{N,d}$ from Theorem 10.

▶ **Theorem 13.** *Assume that $N \geq 2$ is any constant and d an even integer parameter that is growing. Let $F \in \mathbb{F}\langle X\rangle$ be any polynomial such that for each balanced partition Π, $\operatorname{rank}(F,\Pi) = N^{d/2}$. Then, for any constant $\varepsilon \in (0,1)$, any circuit that computes F and satisfies $|\mathcal{T}(C)| = k \leq 2^{d^{\frac{1}{3}-\varepsilon}}$ must have size at least $2^{d^{\frac{1}{3}-\frac{\varepsilon}{2}}}$.*

Proof. Let C be any circuit of size $s \leq N^c$ for $c = d^{1/3-\varepsilon/2}$ with $|\mathcal{T}(C)| = k \leq 2^{d^{1/3-\varepsilon}}$ and computing $f \in \mathbb{F}\langle X\rangle$. We show that there is a balanced partition Π such that $\operatorname{rank}(f,\Pi) < N^{d/2}$. This will prove the theorem.

To show this, we proceed as follows. Using Lemma 11, we can write $f = \sum_{i\in[k]} f_i$ where each $f_i \in \mathbb{F}\langle X\rangle$ is computed by a normal form UPT circuit C_i of size at most $s^2 \leq N^{2c}$.

Fix any $i \in [k]$. By Lemma 12, the number of partitions Π for which $\operatorname{rank}(f_i,\Pi) \geq N^{\frac{d}{2}-c}$ is at most $2^d \cdot \exp(-\Omega(d/c^2))$. In particular, since the number of balanced partitions is $\binom{d}{d/2} = \Theta(\frac{2^d}{\sqrt{d}})$, we see that for a random *balanced* partition Π,

$$\Pr_{\Pi \text{ balanced}}\left[\operatorname{rank}(f_i,\Pi) \geq N^{d/2-c}\right] \leq \sqrt{d}\cdot\exp(-\Omega(d/c^2)) \leq \exp(-d^{1/3}).$$

Say f_i is good for Π if $\operatorname{rank}(f_i,\Pi) \geq N^{d/2-c}$. By the above, we have

$$\Pr_{\Pi \text{ balanced}}[\exists i \in [k] \text{ s.t. } f_i \text{ good for } \Pi] \leq k\cdot\exp(-d^{1/3}) \leq 2^{d^{1/3-\varepsilon}}\cdot\exp(-d^{1/3}) < 1.$$

In particular, there is a balanced Π such that no f_i is good for Π. Fix such a balanced partition Π. By the subadditivity of rank, we have

$$\begin{aligned}\operatorname{rank}(f,\Pi) &\leq \sum_{i\in[k]}\operatorname{rank}(f_i,\Pi) \leq k\cdot N^{d/2-c} \leq 2^{d^{1/3-\varepsilon}}\cdot N^{d/2-c}\\ &= N^{d/2}\cdot\exp(O(d^{1/3-\varepsilon}) - \Omega(d^{1/3-\varepsilon/2})) < N^{d/2}.\end{aligned}$$

This proves the theorem. ◀

Proof of Lemma 12 . Recall from Section 2.2 that we identify each partition Π with an element of $\{-1,1\}^d$. Given partitions $\Pi_1, \Pi_2 \in \{-1,1\}^d$ we use $\langle \Pi_1, \Pi_2\rangle$ to denote their inner product: i.e., $\langle \Pi_1, \Pi_2\rangle := \sum_{i\in[d]} \Pi_1(i)\Pi_2(i)$. Note that the Hamming distance $\Delta(\Pi_1,\Pi_2)$ is

$$\Delta(\Pi_1,\Pi_2) = \frac{d}{2} - \frac{1}{2}\langle \Pi_1, \Pi_2\rangle. \tag{1}$$

Let $\mathcal{T}(C) = \{T\}$. Recall that $|\mathcal{L}(T)| = d$ and by Lemma 8, we can assume that the fan-in of each internal node of T is bounded by 2. For any $u \in \mathcal{I}(T)$ (recall $\mathcal{I}(T)$ is the set of internal nodes of T), let $\mathcal{L}(u)$ denote the set of leaves of the subtree rooted at u. We identify each leaf $\ell \in V(T)$ with $\mathrm{pos}(\ell) \in [d]$. For each $u \in \mathcal{I}(T)$, we can define the partition Π_u from Section 2.4 by $\Pi_u(\ell) = -1$ iff $\ell \in \mathcal{L}(u)$.

For $\gamma > 0$, define a partition Π to be *γ-correlated to T* if for each $u \in \mathcal{I}(T)$, we have $\left|\sum_{\ell \in \mathcal{L}(u)} \Pi(\ell)\right| \leq \gamma$.

Lemma 12 immediately follows from Claims 14 and 15, stated below.

▶ **Claim 14.** *Let Π be any partition of $[d]$ such that $\mathrm{rank}(f, \Pi) \geq N^{d/2-b}$. Then Π is $O(b+c)$-correlated to T.*

Proof. We know from Lemma 9 and Proposition 5 that for each $u \in \mathcal{I}(T)$, $\mathrm{rank}(f, \Pi_u)$ and $\mathrm{rank}(f, -\Pi_u)$ are at most N^c. If Π is a partition such that either $\Delta(\Pi, \Pi_u)$ or $\Delta(\Pi, -\Pi_u)$ is strictly smaller than $\frac{d}{2} - (b+c)$ for *some* $u \in \mathcal{I}(T)$, then by Lemma 6 we would have $\mathrm{rank}(f, \Pi) < N^{d/2-b}$.

Thus, if $\mathrm{rank}(f, \Pi) \geq N^{d/2-b}$, we must have $\min\{\Delta(\Pi, \Pi_u), \Delta(\Pi, -\Pi_u)\} \geq \frac{d}{2} - (b+c)$ for each $u \in \mathcal{I}(T)$. By (1), this means that for each $u \in \mathcal{I}(T)$, $|\langle \Pi, \Pi_u \rangle| \leq \gamma$ for some $\gamma = O(b+c)$.

Let v be the root of T. Note that $\Pi_v \in \{-1, 1\}^d$ is the vector with all its entries being -1. Hence, we have for any $u \in \mathcal{I}(T)$,

$$\left|\sum_{\ell \in \mathcal{L}(u)} \Pi(\ell)\right| = \left|\langle \Pi, \frac{-(\Pi_u + \Pi_v)}{2} \rangle\right| \leq \frac{1}{2}(|\langle \Pi, \Pi_u \rangle| + |\langle \Pi, \Pi_v \rangle|) \leq O(\gamma).$$

This proves the claim. ◀

▶ **Claim 15.** *Say $\Pi \in \{-1, 1\}^d$ is chosen uniformly at random and $\gamma \leq \sqrt{d}$. Then $\Pr_\Pi[\Pi$ is γ-correlated to $T] \leq \exp(-\Omega(\frac{d}{\gamma^2}))$.*

The following technical subclaim is useful for proving Claim 15. (See [17] for proof.)

▶ **Subclaim 16.** *Assume that $r, t \in \mathbb{N}$ such that $rt \leq d/4$. Then we can find a sequence $u_1, \ldots, u_r \in \mathcal{I}(T)$ such that for each $i \in [r]$ we have $|\mathcal{L}(u_i) \setminus \bigcup_{j=1}^{i-1} \mathcal{L}(u_j)| \geq t$.*

Proof of Claim 15. We apply Subclaim 16 with $t = \Theta(\gamma^2)$ and $r = \Theta(d/\gamma^2)$ to get a sequence $u_1, \ldots, u_r \in \mathcal{I}(T)$ such that for each $i \in [r]$, we have $|\mathcal{L}(u_i) \setminus \bigcup_{j=1}^{i-1} \mathcal{L}(u_j)| \geq t$.

By the definition of γ-correlation, we have

$$\begin{aligned}\Pr_\Pi[\Pi \text{ } \gamma\text{-correlated to } T] &\leq \Pr_\Pi\left[\forall i \in [r], \left|\sum_{\ell \in \mathcal{L}(u_i)} \Pi(\ell)\right| \leq \gamma\right] \\ &\leq \prod_{i \in [r]} \Pr_\Pi\left[\left|\sum_{\ell \in \mathcal{L}(u_i)} \Pi(\ell)\right| \leq \gamma \;\middle|\; \{\Pi(\ell) \mid \ell \in \bigcup_{j<i} \mathcal{L}(u_j)\}\right] \qquad (2)\end{aligned}$$

Fix any $i \in [r]$ and $\Pi(\ell)$ for each $\ell \in \mathcal{L}_{<i} := \bigcup_{j<i} \mathcal{L}(u_j)$. The event $|\sum_{\ell \in \mathcal{L}(u_i)} \Pi(\ell)| \leq \gamma$ is equivalent to $\sum_{\ell \in \mathcal{L}(u_i) \setminus \mathcal{L}_{<i}} \Pi(\ell) \in I$ for some interval I of length $2\gamma = O(\sqrt{t})$. This is the probability that the sum of at least t $\{-1, 1\}$-valued random variables chosen i.u.a.r. lies in an interval of length $O(\sqrt{t})$. By the Central Limit theorem, this is at most $1 - \Omega(1)$. By (2), we get $\Pr_\Pi[\Pi$ γ-correlated to $T] \leq \exp\{-\Omega(r)\}$, which proves the claim. ◀

4 Other results

We refer the reader to the full version of the paper [17] for proofs of the results stated below.

Lower bounds for circuits with rotations of one parse tree. Given two parse trees T_1 and T_2 with the same number of leaves, we say that T_1 is a *rotation* of T_2, denoted $T_1 \sim T_2$, if T_1 can be obtained from T_2 by repeatedly reordering the children of various nodes in T_2. Clearly, $\sim$ is an equivalence relation. We use $[\![T]\!]$ to denote the equivalence class of tree T. We say that a homogeneous circuit C is *rotation UPT or rotUPT* if there is a tree T such that $\mathcal{T}(C) \subseteq [\![T]\!]$. We can show the following result.

▶ **Theorem 17.** *Let $N, d \in \mathbb{N}$ be parameters with d even. Let C be a rotUPT circuit of size s computing a polynomial $f \in \mathbb{F}\langle X\rangle$ of degree d over N variables, then there exists a partition $\Pi = \Pi_C$ s.t.* rel-rank(f, Π) *is at most* $\mathrm{poly}(s) \cdot N^{-\Omega(d)}$. *In particular, if $|\mathbb{F}| > q_0(N, d)$ where $q_0(N, d)$ is as in Theorem 10, any rotUPT circuit for $F_{N,d}$ has size $N^{\Omega(d)}$.*

Separation between Few PT formulas and ABPs. We now state two lower bounds for formulas against $\mathrm{IMM}_{n,d}$, yelding separations with ABPs.

We define the *$\times$-depth* of a formula to be the maximum number of $\times$-gates that one can meet on a path from the root to a leaf. Note that if a formula has alternating $+$ and $\times$ gates on each path and has depth Δ' and $\times$-depth Δ, then $\Delta' \geq \Delta \geq \lceil \frac{\Delta'}{2} \rceil$. We will state our first lower bound in terms of $\times$-depth.

▶ **Theorem 18.** *Let F be a UPT formula of $\times$-depth Δ, size s, computing* $\mathrm{IMM}_{n,d} \in \mathbb{F}\langle X\rangle$. *Then, $s \geq n^{\Omega(\Delta d^{1/\Delta})}$. In particular, any UPT formula for* $\mathrm{IMM}_{n,d}$ *must have size $n^{\Omega(\log d)}$.*

This lower bound is actually tight for every $\times$-depth Δ, since the standard divide and conquer approach to computing $\mathrm{IMM}_{n,d}$ gives in fact a UPT formula of size $n^{O(\Delta d^{1/\Delta})}$ and $\times$-depth Δ, for any $\Delta \leq \log d$.

We can also prove a lower bound on the size of k-PT formulas computing $\mathrm{IMM}_{n,d}$ as long as k is significantly smaller than 2^d and $d \leq \log n$.

▶ **Theorem 19.** *Let n, d be growing parameters with $d \leq \log n$. Then, any k-PT formula F computing* $\mathrm{IMM}_{n,d}$ *has size at least n^{ℓ} where $\ell = \Omega(\lg d - \lg\lg k)$. In particular, if $k = 2^{o(d)}$, the* $\mathrm{size}(F) \geq n^{\omega(1)}$ *and if $k = 2^{d^{1-\Omega(1)}}$, then* $\mathrm{size}(F) \geq n^{\Omega(\log d)}$.

Deterministic PIT for UPT and k-PT circuits. We now state two results regarding deterministic whitebox PIT algorithms for UPT and k-PT circuits. The first result was already known in characteristic 0 via the result of Lagarde et al. [18]. However, the algorithm we give, adapting the work of Raz and Shpilka [24], works over fields of any characteristic and runs in time polynomial in the size of the circuit. The second result uses additionally the ideas of Gurjar et al. [10] to extend the above algorithm to a deterministic PIT for sums of k UPT circuits; the algorithm runs in polynomial time as long as k is a constant.

▶ **Theorem 20** (PIT for UPT circuits)**.** *Let $N, s \in \mathbb{N}$ be parameters. There is a deterministic algorithm running in time* $\mathrm{poly}(s)$ *which, on input a UPT circuit C of size at most s over N variables, checks if C computes the zero polynomial or not.*

▶ **Theorem 21** (PIT for sums of k UPT circuits)**.** *Let $N, s, k \in \mathbb{N}$ be parameters. There is a deterministic algorithm running in time $s^{O(2^k)}$ which, on input k UPT circuits $C_1, \ldots, C_k$ (of possibly differing shapes) each of of size at most s over N variables, checks if $\sum_{i=0}^{k} C_i$ computes the zero polynomial or not.*

References

1 Eric Allender, Jia Jiao, Meena Mahajan, and V. Vinay. Non-commutative arithmetic circuits: Depth reduction and size lower bounds. *Theor. Comput. Sci.*, 209(1-2):47–86, 1998. doi:10.1016/S0304-3975(97)00227-2.

2 Vikraman Arvind, Pushkar S. Joglekar, Partha Mukhopadhyay, and S Raja. Identity testing for +-regular noncommutative arithmetic circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:193, 2016. URL: http://eccc.hpi-web.de/report/2016/193.

3 Vikraman Arvind, Pushkar S. Joglekar, and Srikanth Srinivasan. Arithmetic circuits and the hadamard product of polynomials. In Ravi Kannan and K. Narayan Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, volume 4 of *LIPIcs*, pages 25–36. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2304.

4 Vikraman Arvind, Partha Mukhopadhyay, and S Raja. Randomized polynomial time identity testing for noncommutative circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:89, 2016. URL: http://eccc.hpi-web.de/report/2016/089.

5 Vikraman Arvind and S. Raja. The complexity of two register and skew arithmetic computation. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:28, 2014. URL: http://eccc.hpi-web.de/report/2014/028.

6 Steve Chien, Lars Eilstrup Rasmussen, and Alistair Sinclair. Clifford algebras and approximating the permanent. *J. Comput. Syst. Sci.*, 67(2):263–290, 2003. doi:10.1016/S0022-0000(03)00010-2.

7 Steve Chien and Alistair Sinclair. Algebras with polynomial identities and computing the determinant. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 352–361, 2004. doi:10.1109/FOCS.2004.9.

8 Hervé Fournier, Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower bounds for depth-4 formulas computing iterated matrix multiplication. *SIAM J. Comput.*, 44(5):1173–1201, 2015. doi:10.1137/140990280.

9 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Approaching the chasm at depth four. In *Proceedings of the Conference on Computational Complexity (CCC)*, 2013.

10 Rohit Gurjar, Arpita Korwar, Nitin Saxena, and Thomas Thierauf. Deterministic identity testing for sum of read-once oblivious arithmetic branching programs. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 323–346, 2015. doi:10.4230/LIPIcs.CCC.2015.323.

11 Pavel Hrubeš, Avi Wigderson, and Amir Yehudayoff. Non-commutative circuits and the sum-of-squares problem. *Journal of the American Mathematical Society*, 24(3):871–898, 2011.

12 Laurent Hyafil. The power of commutativity. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 171–174. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.31.

13 Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *J. ACM*, 29(3):874–897, 1982. doi:10.1145/322326.322341.

14 Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. An exponential lower bound for homogeneous depth four arithmetic formulas. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 61–70, 2014. doi:10.1109/FOCS.2014.15.

15 Neeraj Kayal, Chandan Saha, and Ramprasad Saptharishi. A super-polynomial lower bound for regular arithmetic formulas. In David B. Shmoys, editor, *Symposium on Theory*

of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014, pages 146–153. ACM, 2014. doi:10.1145/2591796.2591847.

16 Mrinal Kumar and Shubhangi Saraf. On the power of homogeneous depth 4 arithmetic circuits. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 364–373, 2014. doi:10.1109/FOCS.2014.46.

17 Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:77, 2017. URL: https://eccc.weizmann.ac.il/report/2017/077.

18 Guillaume Lagarde, Guillaume Malod, and Sylvain Perifel. Non-commutative computations: lower bounds and polynomial identity testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:94, 2016. URL: http://eccc.hpi-web.de/report/2016/094.

19 Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower bounds for non-commutative skew circuits. *Theory of Computing*, 12(1):1–38, 2016. doi:10.4086/toc.2016.v012a012.

20 Guillaume Malod and Natacha Portier. Characterizing valiant's algebraic complexity classes. *J. Complexity*, 24(1):16–38, 2008. doi:10.1016/j.jco.2006.09.006.

21 Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 410–418. ACM, 1991. doi:10.1145/103418.103462.

22 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. doi:10.1007/BF01294256.

23 Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2):8:1–8:17, 2009. doi:10.1145/1502793.1502797.

24 Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005. doi:10.1007/s00037-005-0188-8.

25 Amir Shpilka and Avi Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Computational Complexity*, 10(1):1–27, 2001. doi:10.1007/PL00001609.

26 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010. doi:10.1561/0400000039.

Approximation and Parameterized Algorithms for Geometric Independent Set with Shrinking*

Michał Pilipczuk[†1], **Erik Jan van Leeuwen**[2], **and Andreas Wiese**[3]

1 **Institute of Informatics, University of Warsaw, Poland**
michal.pilipczuk@mimuw.edu.pl
2 **Department of Information and Computing Sciences, Utrecht University, The Netherlands**
e.j.vanleeuwen@uu.nl
3 **Department of Industrial Engineering and Center for Mathematical Modeling, Universidad de Chile, Chile**
awiese@dii.uchile.cl

Abstract

Consider the Maximum Weight Independent Set problem for rectangles: given a family of weighted axis-parallel rectangles in the plane, find a maximum-weight subset of non-overlapping rectangles. The problem is notoriously hard both in the approximation and in the parameterized setting. The best known polynomial-time approximation algorithms achieve super-constant approximation ratios [5, 7], even though there is a $(1+\epsilon)$-approximation running in quasi-polynomial time [2, 8]. When parameterized by the target size of the solution, the problem is W[1]-hard even in the unweighted setting [12].

To achieve tractability, we study the following *shrinking model*: one is allowed to shrink each input rectangle by a multiplicative factor $1-\delta$ for some fixed $\delta > 0$, but the performance is still compared against the optimal solution for the original, non-shrunk instance. We prove that in this regime, the problem admits an EPTAS with running time $f(\epsilon,\delta)\cdot n^{\mathcal{O}(1)}$, and an FPT algorithm with running time $f(k,\delta)\cdot n^{\mathcal{O}(1)}$, in the setting where a maximum-weight solution of size at most k is to be computed. This improves and significantly simplifies a PTAS given earlier for this problem [1], and provides the first parameterized results for the shrinking model. Furthermore, we explore kernelization in the shrinking model, by giving efficient kernelization procedures for several variants of the problem when the input rectangles are squares.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Combinatorial optimization, Approximation algorithms, Fixed-parameter algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.42

1 Introduction

In Maximum (Weight) Independent Set, given a graph, the goal is to select a set of pairwise non-adjacent vertices with maximum cardinality or total weight. In its full generality, the problem is NP-hard and intractable both in the approximation and in the parameterized

* A full version of the paper is available at https://arxiv.org/abs/1611.06501.
† The research of Mi. Pilipczuk is supported by Polish National Science Centre grant UMO-2013/11/D/ST6/03073. Mi. Pilipczuk is also supported by the Foundation for Polish Science (FNP) via the START stipend programme.

setting: it is NP-hard to approximate within ratio $n^{1-\epsilon}$ for any $\epsilon > 0$ [16], and it is W[1]-hard when parameterized by the solution size [9]. Therefore, many restricted settings were studied.

One well-studied case is to consider a geometric setting where the input consists of a family of geometric objects, and the goal is to select a maximum-weight subfamily of pairwise non-overlapping objects. This case reduces to the graph setting by considering the intersection graph of the objects. These graphs are highly structured, which gives hope for better results than for general graphs.

This paper concentrates on the variant in which the input objects are axis-parallel rectangles in the two-dimensional plane. In this variant, Maximum Weight Independent Set admits much smaller approximation ratios than on general graphs. While no polynomial-time constant-factor approximation algorithm is known in general, there is an $\mathcal{O}(\log \log n)$-approximation algorithm for unweighted rectangles [5], an $\mathcal{O}(\log n / \log \log n)$-approximation algorithm for weighted rectangles [7], and a PTAS for squares [6, 10]. If one allows quasi-polynomial running time, then there is a $(1+\epsilon)$-approximation algorithm (a *QPTAS*) [3, 11]. It remains open whether this can be improved to a PTAS.

From the parameterized perspective, the problem remains W[1]-hard when parameterized by the size k of the solution, even for unweighted unit squares [12]. Therefore, the existence of an FPT algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$ for a computable f is unlikely under standard assumptions from parameterized complexity; this also excludes the existence of an EPTAS for the problem [12]. However, the problem admits a faster-than-brute-force parameterized algorithm with running time $n^{\mathcal{O}(\sqrt{k})}$, which is optimal under the Exponential Time Hypothesis [13]. This algorithm works in the general setting of finding a maximum-weight independent set of size k in a family of polygons in the plane.

Shrinking model. In order to circumvent some of the many challenges that arise when designing approximation or parameterized algorithms for geometric Maximum Weight Independent Set, we investigate the *shrinking model* introduced by Adamaszek et al. [1]. In this model, one is allowed to shrink each input object by a multiplicative factor $1-\delta$ for some fixed $\delta > 0$, but the weight of the computed solution is still compared to the optimum for the original, non-shrunk instance; we give a formal definition later. It is known that the shrinking model allows for substantially better approximation algorithms than the general setting: Adamaszek et al. [1] gave a PTAS for axis-parallel rectangles, which was later generalized by Wiese to arbitrary convex polygons [15]. However, it has not been studied so far whether shrinking also helps to design parameterized algorithms. One concrete question would be whether Independent Set for axis-parallel rectangles remains W[1]-hard in the shrinking model.

Our results. This paper addresses the parameterized complexity of Maximum Weight Independent Set of Rectangles in the shrinking model, and answers the above questions. On the way to our two main parameterized contributions, we also improve the PTAS by Adamaszek et al. [1] to an EPTAS.

Our first main contribution is that Maximum Weight Independent Set of Rectangles is fixed-parameter tractable (FPT) in the shrinking model. Formally, for a shrinking parameter δ, we can decide in (deterministic) time $f(k,\delta) \cdot (nN)^{\mathcal{O}(1)}$ whether there is an independent set of k (shrunk) rectangles, or the original family has no independent subfamily of size k. Here, N is the total bit size of the input and f is some computable function. The algorithm also works in the weighted setting, where we look for a maximum-weight subset of at most k non-overlapping rectangles. The reason why we are able to circumvent

the W[1]-hardness for the standard model (i.e., without shrinking) is that the reduction of Marx [12] relies on tiny differences in the coordinates of the rectangles. However, as Adamaszek et al. [1] and this paper show, this aspect vanishes in the shrinking model.

The parameterized algorithm is actually a consequence of an EPTAS that we present for Maximum Weight Independent Set of Rectangles. That is, we give an algorithm with running time $f(\epsilon,\delta)\cdot(nN)^{\mathcal{O}(1)}$ that finds a subset of rectangles that do not overlap after shrinking by factor $1-\delta$, and whose total weight is at least $1-\epsilon$ times the optimum without shrinking. Recall that the standard model does not admit an EPTAS, unless FPT = W[1] [12].

Our EPTAS is based on the same principles as the PTAS of Adamaszek et al. [1]. The idea is to assemble an optimum solution using a bottom-up dynamic-programming approach pioneered by Erlebach et al. [10]. Each subproblem solved in the dynamic program corresponds to the maximum weight of an independent set contained in a "box", and the computation of the optimum for each such box boils down to enumerating a limited number of carefully chosen partitions of the box into smaller boxes. Intuitively, the ability to shrink is used to make sure that rectangles fit nicely into the different boxes. The main challenge is to ensure that these boxes can be assumed to be simple, and therefore only a limited number of subproblems is necessary to assemble a near-optimum solution.

The crucial contribution in our approximation algorithm is that we show that rectangular boxes suffice. In [1], most rectangles were shrunk in *only one direction* and therefore, the boxes were axis-parallel polygons with at most $g(\epsilon,\delta)$ sides each, for some function g. This makes the dynamic program very complex, and yields a running time of $(nN)^{g(\epsilon,\delta)}$ due to the sheer number of subproblems solved. In this paper, we fully exploit the properties of the shrinking model and shrink *each* rectangle in *two directions*. This changes the analysis, but the main advantage is that we only need to consider boxes that are rectangles (i.e., with only four sides) in our dynamic program. This greatly simplifies the dynamic program, and we show that we need to consider only $f(\epsilon,\delta)\cdot(nN)^{\mathcal{O}(1)}$ different subproblems. Hence, our EPTAS is both substantially faster and significantly simpler than previous work.

Our second main contribution is showing that several important subcases of Maximum Weight Independent Set of Rectangles with δ-shrinking admit polynomial kernels when parameterized by k and δ. Intuitively, such a kernel is a polynomial-time computable subfamily of the input rectangles of size bounded by a polynomial of k and δ that retains an optimum solution after δ-shrinking; a formal definition is given in Section 4. For unit squares of non-uniform weight, we construct a kernel of size $\mathcal{O}(k/\delta^2)$, while for squares of non-uniform size, but of uniform weight, we construct a kernel of size $\mathcal{O}(k^2\cdot\frac{\log(1/\delta)}{\delta^3})$. As a direct consequence, we obtain FPT algorithms for the considered variants with running time $(k/\delta)^{\mathcal{O}(\sqrt{k})}\cdot(nN)^{\mathcal{O}(1)}$ by applying the $n^{\mathcal{O}(\sqrt{k})}$-time algorithm of Marx and Pilipczuk [13] on the kernels. This *subexponential* running time is far better than the running time of our FPT algorithm for the general case.

Organization. In this extended abstract we sketch our EPTAS and present the main ideas behind the kernelization results. A much broader discussion, including complete proofs of all results, can be found in the full version available on the arXiv [14]. The proofs of claims marked with ♠ appear in the full version [14].

2 Preliminaries

We essentially adopt the notation of Adamaszek et al. [1]. Suppose that $\mathcal{R}=\{R_1,R_2,\ldots,R_n\}$ is a family of axis-parallel rectangles given in the input. Each rectangle R_i is described as $R_i=\{(a,b)\,:\,x_i^{(1)}<a<x_i^{(2)}$ and $y_i^{(1)}<b<y_i^{(2)}\}$, where $x_i^{(1)}<x_i^{(2)}$ and $y_i^{(1)}<y_i^{(2)}$

are integers. Thus, the input rectangles are assumed to be open, and their vertices are at integral points. We assume that the family $\mathcal{R}$ is given in the input with all the coordinates $x_i^{(1)}, x_i^{(2)}, y_i^{(1)}, y_i^{(2)}$ encoded in binary; thus, the coordinates are at most exponential in the total bit size of the input, denoted by N. For a rectangle R_i, we define its *width* $g_i = x_i^{(2)} - x_i^{(1)}$ and *height* $h_i = y_i^{(2)} - y_i^{(1)}$. Moreover, each rectangle R_i has an associated *weight* w_i, which is a nonnegative real. For a subset $\mathcal{S} \subseteq \mathcal{R}$, we denote $w(\mathcal{S}) = \sum_{R_i \in \mathcal{S}} w_i$.

Fix a constant δ with $0 < \delta < 1$. For a rectangle R_i, its *δ-shrinking* $R_i^{-\delta}$ is the rectangle with x-coordinates $x_i^{(1)} + \frac{\delta}{2} g_i$ and $x_i^{(2)} - \frac{\delta}{2} g_i$, and y-coordinates $y_i^{(1)} + \frac{\delta}{2} h_i$ and $y_i^{(2)} - \frac{\delta}{2} h_i$. The δ-shrinking retains the weight w_i of the original rectangle. For a subset $\mathcal{S} \subseteq \mathcal{R}$, we denote $\mathcal{S}^{-\delta} = \{R_i^{-\delta} \colon R_i \in \mathcal{S}\}$ to be the family of δ-shrinkings of rectangles from $\mathcal{S}$.

A family of rectangles is *independent* (or is an *independent set*) if the rectangles are pairwise non-overlapping. In the Maximum Weight Independent Set of Rectangles problem (MWISR) we are given a family of axis-parallel rectangles $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$, and the goal is to find a subfamily of $\mathcal{R}$ that is independent and has maximum total weight. This maximum weight will be denoted by $\mathsf{OPT}(\mathcal{R})$. In the parameterized setting, we are additionally given an integer parameter k, and we look for a subfamily of $\mathcal{R}$ that has size at most k, is independent, and has maximum possible weight subject to these conditions. This maximum weight will be denoted by $\mathsf{OPT}_k(\mathcal{R})$.

In the δ-shrinking setting, we relax the requirement of independence to just requiring the disjointness of δ-shrinkings, but we still compare the weight of the output of our algorithm with $\mathsf{OPT}(\mathcal{R})$, respectively with $\mathsf{OPT}_k(\mathcal{R})$.

3 Main results

With the above definitions in mind, we can state formally our main results.

▶ **Theorem 1** (FPT for MWISR with δ-shrinking). *There is a deterministic algorithm that, given a weighted family $\mathcal{R}$ of n axis-parallel rectangles with total encoding size N and parameters k and δ, runs in time $f(k,\delta) \cdot (nN)^c$ for some computable function f and constant c, and outputs a subfamily $\mathcal{S} \subseteq \mathcal{R}$ such that $|\mathcal{S}| \leq k$, $\mathcal{S}^{-\delta}$ is independent, and $w(\mathcal{S}) \geq \mathsf{OPT}_k(\mathcal{R})$.*

▶ **Theorem 2** (EPTAS for MWISR with δ-shrinking). *There is a deterministic algorithm that, given a weighted family $\mathcal{R}$ of n axis-parallel rectangles with total encoding size N and parameters δ, ϵ, runs in time $f(\epsilon,\delta) \cdot (nN)^c$ for some computable function f and constant c, and outputs a subfamily $\mathcal{S} \subseteq \mathcal{R}$ such that $\mathcal{S}^{-\delta}$ is independent, and $w(\mathcal{S}) \geq (1-\epsilon)\mathsf{OPT}(\mathcal{R})$.*

In this section we sketch the proof of Theorem 2. Theorem 1 follows by a simple adjustment of the reasoning, as explained in the full version of the paper [14].

Throughout the proof we fix the input family $\mathcal{R} = \{R_1, \ldots, R_n\}$, and we denote $\mathsf{OPT}(\mathcal{R})$ by OPT. We also fix the constants δ and ϵ, and w.l.o.g. we assume that $1/\delta$ and $1/\epsilon$ are even integers larger than 4. For convenience, throughout the proof we aim at finding a solution $\mathcal{S}$ with $w(\mathcal{S}) \geq (1 - d \cdot \epsilon)\mathsf{OPT}$ for some constant d, for at the end we may rescale the parameter ϵ to ϵ/d. By shifting all the rectangles, we may assume without loss of generality that they all fit into the square $[1, L] \times [1, L]$, where $L = (1/\delta\epsilon)^\ell$ for some integer $\ell = \mathcal{O}(N)$. That is, all the coordinates $x_i^{(1)}, x_i^{(2)}, y_i^{(1)}, y_i^{(2)}$ are between 1 and L, so in particular the width and the height of each rectangle is smaller than L.

We divide our reasoning into two steps. First, like in the PTAS of Adamaszek et al. [1], in Section 3.1 we describe how to remove some rectangles from $\mathcal{R}$ using standard shifting arguments so that OPT decreases only by an $\mathcal{O}(\epsilon)$-fraction, but the resulting family admits

some useful properties. Then, we shrink the rectangles in a similar way as in [1]; however, in contrast to [1] we will shrink *each* rectangle in both directions which will be important in our analysis. Second, we show that the properties of the obtained family enable us to compute an optimum solution using dynamic programming; this algorithm is presented in Section 3.2.

3.1 Sparsifying the family

Intuitively, we will apply shifting techniques to extract some structure in the input family $\mathcal{R}$ while losing only an $\mathcal{O}(\epsilon)$-fraction of OPT. The first goal is to classify the rectangles according to their widths (respectively, heights) such that rectangles in the same class have similar widths (respectively, heights), but between the classes the dimensions differ significantly.

▶ **Definition 3.** A subfamily $\mathcal{R}' \subseteq \mathcal{R}$ is *well-separated* if there exist two partitions $(\mathcal{R}^{\mathsf{V}}_1, \ldots, \mathcal{R}^{\mathsf{V}}_p)$ and $(\mathcal{R}^{\mathsf{H}}_1, \ldots, \mathcal{R}^{\mathsf{H}}_p)$ of $\mathcal{R}'$, with $p \leq \ell$, as well as reals ν_t, μ_t for $t = 1, 2, \ldots, p$, with the following properties satisfied for each $t \in \{1, 2, \ldots, p\}$:

- $\nu_t \leq g_i < \mu_t$ for each $R_i \in \mathcal{R}^{\mathsf{V}}_t$;
- $\nu_t \leq h_i < \mu_t$ for each $R_i \in \mathcal{R}^{\mathsf{H}}_t$;
- $\nu_t / \mu_{t-1} = 1/\delta\epsilon$ (except for $t = 1$) and $\mu_t / \nu_t = (1/\delta\epsilon)^{(1/\epsilon)-1}$; and
- $\nu_1 \leq 1$, $\mu_p \geq L$, and all numbers ν_t and μ_t apart from ν_1 are integers.

The partitions $(\mathcal{R}^{\mathsf{V}}_t)_{t=1,\ldots,p}$ and $(\mathcal{R}^{\mathsf{H}}_t)_{t=1,\ldots,p}$ are called the *vertical* and *horizontal levels*, respectively, whereas the parameters $(\nu_t)_{t=1,\ldots,p}$ and $(\mu_t)_{t=1,\ldots,p}$ are the *lower* and *upper limits* of the corresponding levels. Note that vertical levels partition $\mathcal{R}'$ by width, while the horizontal levels partition $\mathcal{R}'$ by height.

We now prove that we can find a well-separated subfamily that loses only an $\mathcal{O}(\epsilon)$-fraction of OPT using a standard shifting technique. Essentially the same step is used in the PTAS of Adamaszek et al. [1] (see Lemma 6 therein). Henceforth we will use the notation $[q] = \{0, 1, \ldots, q-1\}$ for any positive integer q.

▶ **Lemma 4.** *We can compute a collection of $1/\epsilon$ subfamilies $\mathcal{R}'_0, \ldots, \mathcal{R}'_{1/\epsilon-1} \subseteq \mathcal{R}$ in polynomial time such that each subfamily is well-separated, and there exists a $b^* \in [1/\epsilon]$ for which $\mathsf{OPT}(\mathcal{R}'_{b^*}) \geq (1 - 2\epsilon)\mathsf{OPT}$.*

Proof. Recall that the widths and heights of the rectangles from $\mathcal{R}$ are integers between 1 and $L-1$, where $L = (1/\delta\epsilon)^\ell$. First, create a partition of the rectangles into *vertical layers* $\mathcal{L}^{\mathsf{V}}_j$ for $j = 1, 2, \ldots, \ell$, where layer $\mathcal{L}^{\mathsf{V}}_j$ consists of rectangles R_i for which $(1/\delta\epsilon)^{j-1} \leq g_i < (1/\delta\epsilon)^j$. In a symmetric manner, partition $\mathcal{R}$ into *horizontal layers* $\mathcal{L}^{\mathsf{H}}_j$ for $j = 1, 2, \ldots, \ell$, where layer $\mathcal{L}^{\mathsf{H}}_j$ consists of rectangles R_i for which $(1/\delta\epsilon)^{j-1} \leq h_i < (1/\delta\epsilon)^j$.

For each offset $b \in [1/\epsilon]$ we construct a subfamily $\mathcal{R}'_b$ from $\mathcal{R}$ by removing all rectangles contained in those vertical layers $\mathcal{L}^{\mathsf{V}}_j$ and those horizontal layers $\mathcal{L}^{\mathsf{H}}_j$, for which $j \equiv b \mod (1/\epsilon)$. It is easy to see that each subfamily $\mathcal{R}'_b$ constructed in this manner is well-separated: each vertical level $\mathcal{R}^{\mathsf{V}}_t \subseteq \mathcal{R}'_b$ consists of $(1/\epsilon) - 1$ consecutive vertical layers between two removed ones, with the exception of the first and the last level, for which the start/end of the sequence of layers delimits the level. A symmetric analysis yields the partition into horizontal levels. It is straightforward to compute in polynomial time the partition into horizontal/vertical levels, as well as to choose their lower and upper limits.

Suppose that we choose b uniformly at random from the set $[1/\epsilon]$. Fix any optimum solution $\mathcal{S}$ in $\mathcal{R}$, that is, an independent set of rectangles such that $w(\mathcal{S}) = \mathsf{OPT}$. Observe that for any rectangle $R_i \in \mathcal{S}$, the probability that the vertical layer it belongs to is removed during the construction of $\mathcal{R}'_b$, is equal to ϵ. Similarly, the probability that the horizontal layer to which R_i belongs is removed when constructing $\mathcal{R}'_b$, is also ϵ. Hence, R_i is not

included in $\mathcal{R}'_b$ with probability at most 2ϵ. This means that the expected value of $w(\mathcal{S} \setminus \mathcal{R}'_b)$, the total weight of rectangles from $\mathcal{S}$ that did not survive in $\mathcal{R}'_b$, is at most $2\epsilon \cdot \mathsf{OPT}$. Hence, in expectation we have that $w(\mathcal{S} \setminus \mathcal{R}'_b) \leq 2\epsilon \cdot \mathsf{OPT}$. Therefore, there exists a subfamily $\mathcal{R}'_{b^*}$ with $b^* \in [1/\epsilon]$ such that $\mathsf{OPT}(\mathcal{R}'_{b^*}) \geq (1 - 2\epsilon)\mathsf{OPT}$. ◀

We now execute the rest of the algorithm on $\mathcal{R}'_b$ for each $b \in [1/\epsilon]$, and output the best solution obtained overall. This increases the running time of the algorithm by a factor $1/\epsilon$. From Lemma 4, however, we know that $\mathsf{OPT}(\mathcal{R}'_b) \geq (1 - 2\epsilon)\mathsf{OPT}$ for $b = b^*$, and thus we lose at most a factor $(1 - 2\epsilon)$ in this way. From now on, let $\mathcal{R}' = \mathcal{R}'_b$ for some $b \in [1/\epsilon]$.

Hierarchical grid structure. For any integer a, we describe a hierarchical grid structure and remove rectangles along it. We then execute the rest of the algorithm for a bounded number of values of a, losing at most an additional factor $(1 - 6\epsilon)$ in this way. Given a value of a, the grid structure is constructed as follows. We first divide the horizontal lines into p levels (p as in Definition 3) corresponding to the horizontal levels $\mathcal{R}^{\mathsf{H}}_t$. For level t, define the *level-t unit* as $u_t = \delta\nu_t/2$. Thus, for $t > 1$, we have $u_t = \mu_{t-1}/(2\epsilon)$, and hence u_t is an integer for $t > 1$, since $1/\epsilon$ is even. For each level $t \in \{1, 2, \ldots, p\}$ we define a set of horizontal grid lines G^{H}_t, consisting of the horizontal lines with y-coordinates from the set $\{a + b \cdot u_t \colon b \in \mathbb{Z}\}$. In other words, we take horizontal lines that are u_t apart from each other, and we shift them so that there is a line with y-coordinate a. We define vertical grid lines G^{V}_t of levels $t = 1, 2, \ldots, p$ in a symmetric manner, using the same shift parameter a and the same units for all levels. Define the *grid* of level t to be $G_t = G^{\mathsf{H}}_t \cup G^{\mathsf{V}}_t$. Note that any line of G_t is also a line of $G_{t'}$ for all $t' < t$. Thus, the grid of each level t' refines the grid of each larger level t.

Before we proceed, we describe the intuition of the next step; this step is also present in the PTAS of Adamaszek et al. [1] (see Lemma 7 therein). Rectangles belonging to $\mathcal{R}^{\mathsf{V}}_{t'}$ for $t' \geq t$ have width not smaller than ν_t. On the other hand, the lines of G^{V}_t are spaced at distance $u_t = \delta\nu_t/2$ apart, which means that there are $\Omega(1/\delta)$ vertical grid lines of G^{V}_t crossing each rectangle of vertical level t or larger. Intuitively, G^{V}_t provides a fine grid for those vertical levels, so that their rectangles can be snapped to the lines of G^{V}_t via shrinking by a multiplicative factor of at most $1 - \delta$. On the other hand, the rectangles of vertical levels $t - 1$ or smaller have widths not larger than $\mu_{t+1} = \nu_t \cdot (\delta\epsilon)$. Hence, the grid lines of G^{V}_t are at much larger distance to each other than the maximum possible width of such rectangles; more precisely, larger by a multiplicative factor at least $1/(2\epsilon)$. Consequently, if we were to choose $a \in [L]$ uniformly at random, then the probability that a rectangle R_i will be crossed by a vertical line of level larger than its vertical level, or a horizontal line of level larger than its horizontal level, will be $\mathcal{O}(\epsilon)$. If we exclude such rectangles, then we lose only an $\mathcal{O}(\epsilon)$-fraction of OPT in expectation. We now formalize the above intuition and use it to derive an existential statement and a deterministic algorithm.

We need the following definition. Let $R_i \in \mathcal{R}'$, and suppose that $R_i \in \mathcal{R}^{\mathsf{V}}_s \cap \mathcal{R}^{\mathsf{H}}_t$. We say that R_i is *abusive* if R_i is crossed by a vertical line of level larger than s, or R_i is crossed by a horizontal line of level larger than t; see Figure 1. A family of rectangles without abusive rectangles (with respect to the hierarchical grid structure for a) is called *well-behaved (for a)*.

▶ **Lemma 5.** *Let $U := \sum_{t=1}^{p} u_t$. We can compute a collection of $(1/\delta\epsilon)^{1/\epsilon}$ subfamilies $\mathcal{R}''_0, \ldots, \mathcal{R}''_{(1/\delta\epsilon)^{1/\epsilon}-1} \subseteq \mathcal{R}'$ in $(1/\delta\epsilon)^{1/\epsilon} \cdot (nN)^{\mathcal{O}(1)}$ time such that, for each $c \in [(1/\delta\epsilon)^{1/\epsilon}]$, the subfamily $\mathcal{R}''_c$ is well-separated and well-behaved for $c \cdot U$, and there exists a $c^* \in [(1/\delta\epsilon)^{1/\epsilon}]$ for which $\mathsf{OPT}(\mathcal{R}''_{c^*}) \geq (1 - 6\epsilon)\mathsf{OPT}(\mathcal{R}')$.*

Proof. For each $c \in [(1/\delta\epsilon)^{1/\epsilon}]$, the family $\mathcal{R}''_c$ is obtained from $\mathcal{R}'$ by removing all rectangles that are abusive with respect to the hierarchical grid structure for $a = c \cdot U$. Hence, $\mathcal{R}''_c$ is well-behaved for $c \cdot U$. Since we are only removing rectangles, $\mathcal{R}''_c$ is still well-separated.

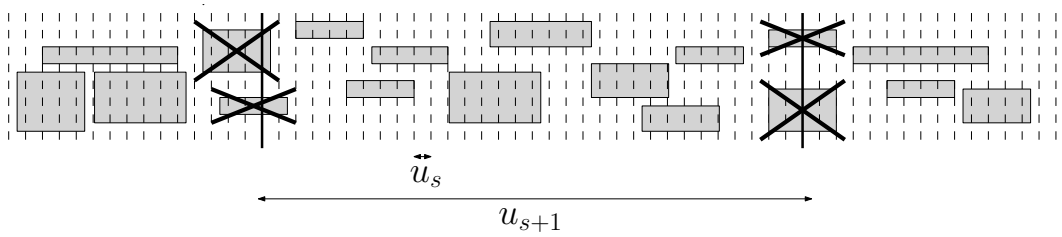


Figure 1 The vertical grid. The dashed vertical lines are the vertical grid lines of G_s^{V}, the bold vertical lines are the lines in the set G_{s+1}^{V}. All shown rectangles are from level $\mathcal{R}_s^{\mathsf{V}}$. The crossed out rectangles are abusive since they intersect lines from G_{s+1}^{V}.

It remains to show the existence of c^*. Let $R_i \in \mathcal{R}_s^{\mathsf{V}}$. As $R_i \in \mathcal{R}_s^{\mathsf{V}}$, we have that $g_i < \mu_s = \nu_{s+1} \cdot (\delta\epsilon) = u_{s+1} \cdot 2\epsilon$. Hence,

$$\frac{g_i}{\sum_{r=1}^{s} u_r} \leq \frac{g_i}{u_s} < \frac{u_{s+1} \cdot 2\epsilon}{u_s} = 2\epsilon \cdot (1/\delta\epsilon)^{1/\epsilon}. \tag{1}$$

Note that this inequality holds regardless of the choice of a for the construction of the hierarchical grid structure. Now consider the hierarchical grid structure for $a = c \cdot U$ for some $c \in [(1/\delta\epsilon)^{1/\epsilon}]$. Rectangle R_i is crossed by a vertical line of level larger than s if and only if it is crossed by a vertical line of level $s+1$. Lines of G_{s+1}^{V} are spaced at distance u_{s+1} from each other, which means that R_i is crossed by a line of G_{s+1}^{V} if and only if the remainder of $c \cdot U$ modulo u_{s+1} is among a set Γ_i of $g_i - 1$ consecutive remainders from $[u_{s+1}]$, being the remainders of the x-coordinates of vertical lines that cross R_i. By (1), Γ_i contains at most $2\epsilon \cdot (1/\delta\epsilon)^{1/\epsilon} + 2 \leq 3\epsilon \cdot (1/\delta\epsilon)^{1/\epsilon}$ multiples of $\sum_{r=1}^{s} u_r$. On the other hand, observe that $0 \leq c \cdot \sum_{r=1}^{s} u_r < u_{s+1}$ for all $c \in [(1/\delta\epsilon)^{1/\epsilon}]$, and that u_r divides u_{r+1} for each r. Hence, $c \cdot U$ gives remainder $c \cdot \sum_{r=1}^{s} u_r$ modulo u_{s+1}, which is always a multiple of $\sum_{r=1}^{s} u_r$. In particular, it follows that the multiples of $\sum_{r=1}^{s} u_r$ contained in Γ_i constitute at most a 3ϵ-fraction of all the remainders modulo u_{s+1} that $c \cdot U$ attains for $c \in [(1/\delta\epsilon)^{1/\epsilon}]$.

Suppose now that $c \in [(1/\delta\epsilon)^{1/\epsilon}]$ is chosen uniformly at random. Let $R_i \in \mathcal{R}_s^{\mathsf{V}} \cap \mathcal{R}_t^{\mathsf{H}}$. By the previous observation, R_i is crossed by a vertical line of level larger than s with probability at most 3ϵ. A similar analysis shows that R_i is crossed by a horizontal line of level larger than t with probability at most 3ϵ. Therefore, R_i is abusive with probability at most 6ϵ, and the total expected weight of the abusive rectangles in $\mathsf{OPT}(\mathcal{R}')$ with respect to $a = c \cdot U$ is bounded by $6\epsilon \cdot \mathsf{OPT}(\mathcal{R}')$. Hence, the value c^*, as claimed in the lemma statement, exists. ◀

We now execute the rest of the algorithm on $\mathcal{R}_c''$ for each $c \in [(1/\delta\epsilon)^{1/\epsilon}]$, and output the best solution obtained overall. This increases the running time of the algorithm by a factor $(1/\delta\epsilon)^{1/\epsilon}$. From Lemma 5, we know that $\mathsf{OPT}(\mathcal{R}_c'') \geq (1 - 6\epsilon)\mathsf{OPT}(\mathcal{R}')$ for $c = c^*$, and thus we lose at most a factor $(1 - 6\epsilon)$. From now on, let $\mathcal{R}'' = \mathcal{R}_c''$ for some $c \in [(1/\delta\epsilon)^{1/\epsilon}]$.

Snapping by shrinking. When considering $\mathcal{R}''$, the lines of G_t^{V} provide a fine division of every rectangle from vertical level t or larger, while no rectangle of smaller vertical level is crossed by them; symmetrically for horizontal grid lines. The idea now is to shrink each rectangle $R_i \in \mathcal{R}''$ so that its vertical sides are aligned with some vertical grid lines of the vertical level of R_i, while the horizontal sides are aligned with some horizontal grid lines of the horizontal level of R_i. This is formalized in the next lemma, which is also similar to Adamaszek et al. [1]. However, in this step there is a subtle but crucial difference. Consider a rectangle $R_i \in \mathcal{R}''$, and suppose $R_i \in \mathcal{R}_s^{\mathsf{V}} \cap \mathcal{R}_t^{\mathsf{H}}$. In [1], R_i is shrunk in the vertical dimension only if $s \geq t$ and in the horizontal dimension only if $t \geq s$. Here we always do both, which will be important for our dynamic programming.

▶ **Lemma 6.** *In polynomial time we can compute a well-behaved family of axis-parallel rectangles $\mathcal{Q}$ that contains one rectangle Q_i for each $R_i \in \mathcal{R}''$, of the same weight w_i as R_i and satisfying the following conditions:*

- *$R_i^{-\delta} \subseteq Q_i \subseteq R_i$ for each $R_i \in \mathcal{R}''$; and*
- *if $R_i \in \mathcal{R}_s^{\mathsf{V}} \cap \mathcal{R}_t^{\mathsf{H}}$, then both vertical sides of Q_i are contained in some vertical grid lines of G_s^{V}, and both horizontal sides of Q_i are contained in some horizontal grid lines of G_t^{H}.*

Proof. Take any $R_i \in \mathcal{R}''$, and suppose $R_i \in \mathcal{R}_s^{\mathsf{V}} \cap \mathcal{R}_t^{\mathsf{H}}$. We define Q_i as the rectangle cut from the plane by the following four lines:

- the left-most and the right-most vertical grid lines of G_s^{V} that cross R_i;
- the bottom-most and the top-most horizontal grid lines of G_t^{H} that cross R_i.

Clearly, we have that $Q_i \subseteq R_i$ and the second condition of the statement is satisfied. We are left with proving that $R_i^{-\delta} \subseteq Q_i$.

Consider first the left side of Q_i, which is contained in the left-most vertical grid line of G_s^{V} that crosses R_i. Since $R_i \in \mathcal{R}_s^{\mathsf{V}}$, we have that $g_i \geq \nu_s$, while the grid lines of G_s^{V} are spaced at distance $u_t = \delta\nu_s/2$ apart. This means that the left-most vertical grid line crossing R_i has the x-coordinate not larger than $x_i^{(1)} + \delta\nu_s/2$, which in turn is not larger than $x_i^{(1)} + \delta g_i/2$. This means that the left side of Q_i is either to the left or at the same x-coordinate as the left side of $R_i^{-\delta}$. An analogous reasoning can be applied to the other three sides of Q_i, thereby proving that $R_i^{-\delta} \subseteq Q_i$.

Note that since $\mathcal{Q}$ is obtained only by shrinking rectangles from $\mathcal{R}''$, it is still the case that no rectangle of $\mathcal{Q}$ is abusive. ◀

By Lemma 4 and Lemma 5, $\mathsf{OPT}(\mathcal{Q}) \geq \mathsf{OPT}(\mathcal{R}'') \geq (1-6\epsilon)\mathsf{OPT}(\mathcal{R}') \geq (1-8\epsilon)\mathsf{OPT}$ if $\mathcal{R}' = \mathcal{R}'_{b^*}$ and $\mathcal{R}'' = \mathcal{R}''_{c^*}$. Hence, the optimum solution for $\mathcal{Q}$ indeed has large enough weight. Moreover, by the first condition of Lemma 6, for any independent set of rectangles in $\mathcal{Q}$, the corresponding rectangles in $\mathcal{R}^{-\delta}$ are also independent. Hence, any solution for MWISR on $\mathcal{Q}$ projects to a solution of the same weight for MWISR with δ-shrinking on $\mathcal{R}$.

From now on we focus on the family $\mathcal{Q}$. For each $t \in \{1, 2, \ldots, p\}$, let $\mathcal{Q}_t^{\mathsf{V}} = \{Q_i \colon R_i \in \mathcal{R}_t^{\mathsf{V}}\}$ and $\mathcal{Q}_t^{\mathsf{H}} = \{Q_i \colon R_i \in \mathcal{R}_t^{\mathsf{H}}\}$.

3.2 Dynamic programming

We now present a dynamic programming algorithm that, given the family $\mathcal{Q}$ constructed in the previous section, computes the value $\mathsf{OPT}(\mathcal{Q})$. An optimum solution can be recovered from the run of the dynamic program using standard methods, and hence for simplicity we omit this aspect in the description.

We describe the algorithm as *backtracking with memoization*. That is, subproblems are solved by recursion, but once a subproblem has been solved once, the optimum value for it is stored in a map (is *memoized*), and further calls to solving this subproblem will only retrieve the memoized optimum value, rather than solve the subproblem again. Solving each subproblem (excluding recursive subcalls) takes time $f(\delta, \epsilon) \cdot n^{\mathcal{O}(1)}$ for some computable function f, and we argue that at most $g(\delta, \epsilon) \cdot (nN)^{\mathcal{O}(1)}$ subproblems are solved in total, for some other computable function g. This ensures the promised running time of the algorithm.

We first define subproblems. A *subproblem* is a tuple $I = (s, t, x_1, x_2, y_1, y_2)$, where the meaning of the entries is as follows. The pair $(s, t) \in \{1, \ldots, p\} \times \{1, \ldots, p\}$ is the *level* of the subproblem, which consists of the vertical level s and the horizontal level t. The numbers x_1, x_2, y_1, y_2 are integers satisfying $x_1 < x_2 \leq x_1 + (1/\delta\epsilon)^{1/\epsilon}$ and $y_1 < y_2 \leq y_1 + (1/\delta\epsilon)^{1/\epsilon}$. Integers x_1, x_2 are the lower and upper *vertical offsets*, respectively, while y_1, y_2 are the

lower and upper *horizontal offsets*, respectively. The *area covered* by subproblem $I = (s, t, x_1, x_2, y_1, y_2)$ is the rectangle

$$A_I = (a + x_1 \cdot u_s, a + x_2 \cdot u_s) \times (a + y_1 \cdot u_t, a + y_2 \cdot u_t).$$

In other words, (x_1, x_2, y_1, y_2) define the offsets of the four grid lines – two from G_s^{V} and two from G_t^{H} – that cut out A_I from the plane.

For a subproblem I, let $\mathcal{Q}_I$ be the family of all rectangles from $\mathcal{Q}$ that are contained in A_I. The next check follows from a simple calculation of parameters.

▶ **Lemma 7 (♠).** *If subproblem I has level (s, t), then $\mathcal{Q}_I \subseteq \bigcup_{s' \leq s,\, t' \leq t} \mathcal{Q}_{s'}^{\mathsf{V}} \cap \mathcal{Q}_{t'}^{\mathsf{H}}$.*

For a subproblem I, we define the *value* of I, denoted $\mathsf{Value}(I)$, as the maximum weight of a subfamily of $\mathcal{Q}_I$ that is independent. We show that there is a subproblem that encompasses the whole instance. Then, we show how to *solve* each subproblem I, that is, to compute $\mathsf{Value}(I)$, using recursion.

▶ **Lemma 8 (♠).** *There is a subproblem I_{all} of level (p, p), computable in constant time, such that $A_{I_{\mathsf{all}}} \supseteq (1, L) \times (1, L)$. Consequently, $\mathsf{OPT}(\mathcal{Q}) = \mathsf{Value}(I_{\mathsf{all}})$.*

Next comes the crucial point: we show how to *solve* each subproblem I, that is, to compute $\mathsf{Value}(I)$, using recursion.

▶ **Lemma 9 (♠).** *A subproblem I of level (s, t) can be solved using $f(\delta, \epsilon)$ calls to solving subproblems of levels $(s-1, t)$, $(s, t-1)$, and $(s-1, t-1)$, for some computable function f. Moreover, the time needed for this computation, excluding the time spent in the recursive calls, is at most $f(\delta, \epsilon) \cdot n$.*

Proof sketch. Consider all vertical lines of level s and all horizontal lines of level t that cross the rectangle A_I; their number is bounded by $(1/\delta\epsilon)^{1/\epsilon}$. These lines partition A_I into at most $(1/\delta\epsilon)^{2/\epsilon}$ smaller *cells* in a natural manner. Consider an independent set $\mathcal{S} \subseteq \mathcal{Q}_I$ such that $w(\mathcal{S}) = \mathsf{Value}(I)$. By obtaining a well-behaved family and applying the snapping procedure, we have the following structure; see Figure 2. Each rectangle from $\mathcal{S}$ of level (s, t) just occupies a rectangle of cells, each of them entirely. Whenever a rectangle from $\mathcal{S}$ is of level (s', t) for some $s' < s$, it is contained in a single column of cells, and its horizontal sides are aligned with some horizontal lines of the grid of cells. A symmetrical claim holds for rectangles from $\mathcal{S}$ of level (s, t') for any $t' < t$. Finally, any rectangle of $\mathcal{S}$ of level (s', t') for $s' < s$ and $t' < t$ is contained in a single cell.

It can be then easily seen that the whole grid of cells admits a partition into "boxes" such that each rectangle of $\mathcal{S}$ fits into a single box. Each box that is not contained in one column or in one row must be filled with a single rectangle of level (s, t), and we can greedily take the heaviest such rectangle. Each other box defines a subproblem of level (s', t') where $s' \leq s$, $t' \leq t$, and one of these inequalities is strict. Hence, an optimum solution for such a box can be computed using a recursive call. Therefore, the algorithm enumerates all partitions of the cells into boxes, and for each of them computes a candidate value using recursive subcalls; the value of I is the largest among the candidates. ◀

Finally, to bound the running time of the algorithm, we prove that there is only a small number of subproblems I for which $\mathcal{Q}_I$ is nonempty. Obviously, only such subproblems are necessary to solve, as the others have value 0.

▶ **Lemma 10 (♠).** *The number of subproblems I for which $\mathcal{Q}_I$ is nonempty is at most $81 \cdot (1/\delta\epsilon)^{4/\epsilon} \cdot |\mathcal{Q}| p^2$.*

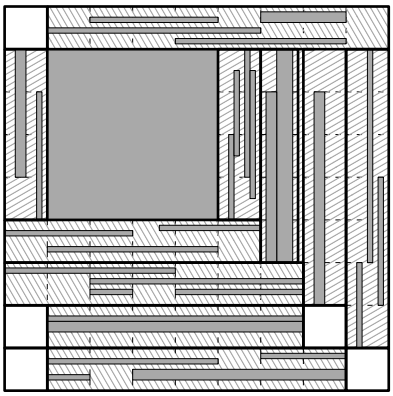


Figure 2 The partition of A_I into large, horizontal, vertical, and small cells. The figure is a slightly modified figure from [1].

Having gathered all the tools, we can describe the algorithm. To compute $\mathsf{OPT}(\mathcal{Q})$, it is sufficient to compute $\mathsf{Value}(I_{\mathsf{all}})$ for the subproblem I_{all} given by Lemma 8. For this we use backtracking with memoization. Starting from I_{all}, we recursively solve subproblems as explained in Lemma 9. Whenever $\mathsf{Value}(I)$ has been computed for some subproblem I, then this value is memoized in a map, and further calls to solving I will only return the value retrieved from the map, instead of recomputing the value again. Furthermore, whenever we attempt to compute $\mathsf{Value}(I)$ for a subproblem I for which $\mathcal{Q}_I$ is empty, we immediately return 0 instead of applying the procedure of Lemma 9. Therefore, the total running time of the algorithm is upper bounded by the number of subproblems I for which $\mathcal{Q}_I$ is nonempty, times the time spent on internal computations for each of them, including checking whether the respective family $\mathcal{Q}_I$ is empty and whether $\mathsf{Value}(I)$ has already been memoized. The first factor is bounded by $81 \cdot (1/\delta\epsilon)^{4/\epsilon} \cdot |\mathcal{Q}|p^2 \leq 81 \cdot (1/\delta\epsilon)^{4/\epsilon} \cdot nN^2$ due to Lemma 10. The second factor is bounded by $f(\delta,\epsilon) \cdot n^d$ for some constant d, due to Lemma 9. Hence, the running time of the whole algorithm is $f(\delta,\epsilon) \cdot (nN)^c$ for some computable function f and constant c. As mentioned before, the algorithm can be trivially adjusted to also compute an independent set of weight $\mathsf{Value}(I_{\mathsf{all}})$ by storing the value of each subproblem together with some independent set that certifies this value.

Summarizing, the dynamic programming described above computes an independent set in $\mathcal{Q}$ of weight $\mathsf{OPT}(\mathcal{Q})$ in time $f(\delta,\epsilon) \cdot (nN)^c$. As argued in the previous section, such an independent set projects to an independent set of the same weight in $\mathcal{R}^{-\delta}$, and $\mathsf{OPT}(\mathcal{Q}) \geq (1-8\epsilon)\mathsf{OPT}$ holds. This concludes the proof of Theorem 2.

4 Kernelization results

In this section we discuss kernelization for the case when the input family consists of squares. We first clarify the definition of a kernel, and then present the results.

Definition of kernel. The classic definition of kernelization is tailored to decision problems. Extending it to optimization problems in a weighted setting is often problematic, and making it compatible with the δ-shrinking model complicate it even more. Hence, we explicitly define kernelization for MWISR in the shrinking model, bearing in mind the main principle of kernelization: solving the kernel should project to a solution for the original instance.

▶ **Definition 11.** Let k be a non-negative integer, let $\delta \in (0,1)$, and let $\mathcal{R}$ be a family of axis-parallel rectangles. Then, a *kernel for* $(\mathcal{R}, k, \delta)$ is a polynomial-time computable subfamily $\mathcal{Q} \subseteq \mathcal{R}$ such that $|\mathcal{Q}| \leq f(k,\delta)$ for a computable function f, called the *size* of the kernel, and:

$$\mathsf{OPT}_k(\mathcal{Q}^{-\delta}) \geq \mathsf{OPT}_k(\mathcal{R}).$$

Thus, MWISR on $\mathcal{R}$ with the δ-shrinking relaxation may be solved by solving MWISR on $\mathcal{Q}^{-\delta}$ (without the δ-shrinking relaxation). If one wishes to use an algorithm for the shrinking relaxation on the kernel, then applying it to $\mathcal{Q}^{-\delta}$ with parameter δ will yield a subfamily $\mathcal{S}$ of size k such that $\mathcal{S}^{-2\delta}$ is independent and $w(\mathcal{S}) \geq \mathsf{OPT}_k(\mathcal{Q}^{-\delta}) \geq \mathsf{OPT}_k(\mathcal{R})$. Hence, this solves the original problem for 2δ-shrinking and we can rescale δ accordingly.

Definition 11 lacks one aspect of the classic notion of kernelization. Namely, the weights and the coordinates of the rectangles in the kernel are inherited from the original instance, so their bit encoding may not be bounded in terms of k and δ. We prefer to work with Definition 11, because it focuses our efforts on the core combinatorial aspects of our kernelization procedures. However, in the full version we argue that the sizes of the bit encodings of both the weights and the coordinates essentially can be reduced to polynomials in k and $1/\delta$ [14].

Results. The following theorem summarizes our kernelization results.

▶ **Theorem 12 (♠).** *Given a non-negative integer k, $\delta \in (0,1)$, and a family of axis-parallel squares $\mathcal{R}$, the following kernels for $(\mathcal{R}, k, \delta)$ can be computed in polynomial time:*

1. *If $\mathcal{R}$ consists of unit squares of uniform weight, then there is a kernel of size $\leq 16k/\delta^2$.*
2. *If $\mathcal{R}$ consists of unit squares of non-uniform weight, then there is a kernel of size $\leq 64k/\delta^2$.*
3. *If $\mathcal{R}$ consists of squares of non-uniform size, but of uniform weight, then there is a kernel of size $\mathcal{O}(k^2 \cdot \frac{\log(1/\delta)}{\delta^3})$.*

We now briefly sketch the main ideas. Consider first the simplest case of unit squares of uniform weight. Suppose two squares R_i and R_j are very close to each other: their centers are at distance less than $\delta/2$ in the ℓ_∞-metric (we will say that R_i and R_j are $(\delta/2)$-*close*). Then it is not hard to prove that $R_j^{-\delta} \subseteq R_i$, so intuitively, in the δ-shrinking model R_j is always a valid substitute for R_i. This allows for the following greedy strategy. Compute an inclusion-wise maximal subfamily $\mathcal{Q} \subseteq \mathcal{R}$ such that the centers of squares in $\mathcal{Q}$ are pairwise at distance at least $\delta/2$. By maximality, for every square $R_i \in \mathcal{R}$, there is a square $\phi(R_i) \in \mathcal{Q}$ that is $(\delta/2)$-close to R_i. By the observation above, the mapping ϕ maps every independent set in $\mathcal{R}$ to a subset of $\mathcal{Q}$ of the same size that is independent after δ-shrinking. Consequently, $\mathsf{OPT}_k(\mathcal{Q}^{-\delta}) \geq \mathsf{OPT}_k(\mathcal{R})$ and we may work with $\mathcal{Q}$ instead. However, the fact that squares of $\mathcal{Q}$ have centers pairwise far from each other immediately shows that the intersection graph of $\mathcal{Q}^{-\delta}$ has maximum degree bounded by a function of $1/\delta$ (similar to [4]). Then it is a standard exercise to give a linear kernel for Independent Set.

For unit squares of non-uniform weight, we follow the same strategy, but we construct $\mathcal{Q}$ greedily by iteratively taking the heaviest square and removing all squares that are $(\delta/2)$-close to it. This ensures that the substitute $\phi(R_i)$ is always at least as heavy as R_i. The maximum degree of the intersection graph of $\mathcal{Q}^{-\delta}$ can be bounded in the same manner. There is also a linear kernel for Maximum Weight Independent Set on bounded degree graphs, following the observation: Iterate k times the procedure of picking the heaviest vertex and removing all vertices at distance at most 2 from it. Then there is some maximum-weight independent set of size at most k that is contained in the removed vertices.

Finally, in the case of squares of non-uniform size and uniform weight, the following observation is crucial: if there are two squares R_i and R_j whose sizes differ by a multiplicative factor at least $2/\delta$, then either one is contained in the other, or they become disjoint after δ-shrinking. Observe that we may assume that the input does not contain any pair of squares where one is contained in the other, since the smaller one can be always selected instead of the larger. Hence, squares of very different sizes become disjoint after δ-shrinking. We partition the squares into levels according to the magnitude of their side lengths, and apply the following win-win approach. If many levels are non-empty, then we can find k of them so that picking one square from each yields an independent set of size k after δ-shrinking. Otherwise, only few levels are non-empty, and we can treat each level separately using essentially the same methods as for unit squares.

We conclude by discussing some applications. By composing our kernelization algorithms with the parameterized algorithm of Marx and Pilipczuk [13] for finding the heaviest k-independent set of polygons in the plane, we obtain algorithms with running time $(k/\delta)^{\mathcal{O}(\sqrt{k})} \cdot (nN)^{\mathcal{O}(1)}$ for all the problems encompassed by Theorem 12; this is much faster than the algorithm of Theorem 1. Also, some of our intermediate tools can be combined with the results of Alber and Fiala [4], yielding algorithms with running time $2^{\mathcal{O}_\delta(\sqrt{k})} \cdot (nN)^{\mathcal{O}(1)}$ for unit squares of non-uniform weight. We also obtain a faster EPTAS than Theorem 2 for squares of uniform size or uniform weight, for which the exponent depends only linearly on $1/\epsilon$. A precise description of these results can be found in the full version [14].

5 Conclusions

In this paper we have initiated the study of the shrinking model for parameterized geometric Independent Set, by giving FPT algorithms and polynomial kernels for the most basic variants. Most importantly, we have showcased that the shrinking model leads to robust tractability of problems that without this relaxation are hard from the parameterized perspective. We hope that this is the start of an interesting research direction, as our work raises several concrete open problems. Can our FPT algorithm and EPTAS for axis-parallel rectangles (Theorems 2 and 1) be generalized to arbitrary convex polygons, as is the case for the PTAS [15]? Recall that it was important for our algorithm that via shrinking we can align all edges of each rectangle with grid lines of suitable granularity. Then we could partition the plane recursively along these grid lines. Such an alignment is no longer possible for polygons, not even if each polygon is essentially a diagonal straight line segment. Instead, the PTAS in [15] crucially relies on guessing separators described via $\mathcal{O}_\epsilon(1)$ input polygons (and additionally $\mathcal{O}_\epsilon(1)$ grid lines). The total number of candidates for such separators is $n^{\mathcal{O}_\epsilon(1)}$, which leads to the running time of $n^{\mathcal{O}_\epsilon(1)}$ of the algorithm. Further, is it possible to improve the running time of our FPT algorithm to subexponential, that is, $2^{o(k)} \cdot (nN)^{\mathcal{O}(1)}$ for every fixed δ? What about polynomial kernels, i.e., kernelization procedures with polynomial output guarantees, for more complex objects than squares? Here, it seems that our arguments apply mutatis mutandis to e.g. (unit) disks instead of (unit) squares, but it is conceivable that even the general setting of convex polygons can be treated, for an appropriate definition of shrinking. Also, is there a polynomial kernel for squares of non-uniform size and non-uniform weight? This is not addressed in its full generality by our kernelization algorithms. Finally, can one give limits to tractability in the shrinking model, by showing W[1]-hardness or the nonexistence of polynomial kernels?

Acknowledgements. The first author thanks Dániel Marx for discussions about the setting.

References

1. Anna Adamaszek, Parinya Chalermsook, and Andreas Wiese. How to Tame Rectangles: Solving Independent Set and Coloring of Rectangles via Shrinking. In *Proc. APPROX/RANDOM 2015*, volume 40 of *LIPIcs*, pages 43–60. Schloss Dagstuhl, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.43.
2. Anna Adamaszek and Andreas Wiese. Approximation schemes for maximum weight independent set of rectangles. In *Proc. FOCS 2013*, pages 400–409. IEEE, 2013. doi:10.1109/FOCS.2013.50.
3. Anna Adamaszek and Andreas Wiese. A QPTAS for maximum weight independent set of polygons with polylogarithmically many vertices. In *Proc. SODA 2014*, pages 645–656. SIAM, 2014. doi:10.1137/1.9781611973402.49.
4. Jochen Alber and Jiří Fiala. Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *J. Algorithms*, 52(2):134–151, 2004. doi:10.1016/j.jalgor.2003.10.001.
5. Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In *Proc. SODA 2009*, pages 892–901. SIAM, 2009.
6. Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *Journal of Algorithms*, 46(2):178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.
7. Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Comp. Geometry*, 48(2):373–392, 2012. doi:10.1007/s00454-012-9417-5.
8. Julia Chuzhoy and Alina Ene. On approximating maximum independent set of rectangles. In *Proc. FOCS 2016*, pages 820–829. IEEE, 2016. doi:10.1109/FOCS.2016.92.
9. Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1):109–131, 1995. doi:10.1016/0304-3975(94)00097-3.
10. Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. on Computing*, 34(6):1302–1323, 2005. doi:10.1137/S0097539702402676.
11. Sariel Har-Peled. Quasi-polynomial time approximation scheme for sparse subsets of polygons. In *Proc. SOCG 2014*, pages 120–129. ACM, 2014. doi:10.1145/2582112.2582157.
12. Dániel Marx. Efficient approximation schemes for geometric problems? In *Proc. ESA 2005*, volume 3669 of *LNCS*, pages 448–459. Springer, 2005. doi:10.1007/11561071_41.
13. Dániel Marx and Michał Pilipczuk. Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams. In *Proc. ESA 2015*, volume 9294 of *LNCS*, pages 865–877. Springer, 2015. doi:10.1007/978-3-662-48350-3_72.
14. Michał Pilipczuk, Erik Jan van Leeuwen, and Andreas Wiese. Approximation and parameterized algorithms for geometric independent set with shrinking. *CoRR*, abs/1611.06501, 2016. arXiv:1611.06501.
15. Andreas Wiese. Independent set of convex polygons: From n^ϵ to $1+\epsilon$ via shrinking. In *Proc. LATIN 2016*, volume 9644 of *LNCS*, pages 700–711. Springer, 2016. doi:10.1007/978-3-662-49529-2_52.
16. David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3:103–128, 2007. doi:10.4086/toc.2007.v003a006.

Eilenberg Theorems for Free

Henning Urbat[*,1], **Jiří Adámek**[*,2], **Liang-Ting Chen**[3], and **Stefan Milius**[†,4]

1 **Institut für Theoretische Informatik, Technische Universität Braunschweig, Germany**
2 **Department of Mathematics, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic**
3 **Department of Computer Science, Swansea University, United Kingdom**
4 **Lehrstuhl für Theoretische Informatik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany**

Abstract

Eilenberg-type correspondences, relating varieties of languages (e.g., of finite words, infinite words, or trees) to pseudovarieties of finite algebras, form the backbone of algebraic language theory. We show that they all arise from the same recipe: one models languages and the algebras recognizing them by monads on an algebraic category, and applies a Stone-type duality. Our main contribution is a variety theorem that covers e.g. Wilke's and Pin's work on ∞-languages, the variety theorem for cost functions of Daviaud, Kuperberg, and Pin, and unifies the two categorical approaches of Bojańczyk and of Adámek et al. In addition we derive new results, such as an extension of the local variety theorem of Gehrke, Grigorieff, and Pin from finite to infinite words.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Eilenberg's theorem, variety of languages, pseudovariety, monad, duality

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.43

1 Introduction

Algebraic language theory studies the behaviors of machines by relating them to algebraic structures. This has proved extremely fruitful. For example, regular languages can be described as the languages recognized by finite monoids, and the decidability of star-freeness rests on Schützenberger's theorem [30]: a regular language is star-free iff it is recognized by a finite aperiodic monoid. At the heart of algebraic language theory are results establishing generic correspondences of this kind. The prototype is Eilenberg's variety theorem [12], which states that *varieties of languages* (classes of regular languages closed under boolean operations, derivatives, and homomorphic preimages) correspond to *pseudovarieties of monoids* (classes of finite monoids closed under quotients, submonoids, and finite products). This together with Reiterman's theorem [25], stating that pseudovarieties of monoids can be specified by profinite equations, establishes a firm connection between automata, languages, and algebras.

Inspired by Eilenberg's work, over the past four decades numerous further variety theorems were discovered for regular languages [14,20,24,31], treating notions of varieties with modified closure properties, but also for machine behaviors beyond finite words, including weighted languages over a field [26], infinite words [21,35], words on linear orderings [5,6], ranked

* Supported by Deutsche Forschungsgemeinschaft (DFG) under project AD 187/2-1
† Supported by Deutsche Forschungsgemeinschaft (DFG) under project MI 717/5-1

trees [4], binary trees [29], and cost functions [11]. This plethora of structurally similar results has raised interest in category-theoretic approaches which allow to derive all the above results as special instances of only *one* general variety theorem. The first step in this direction was achieved in our previous work [1–3, 10]: there we replaced monoids by monoid objects in a category $\mathcal{D}$ of (ordered) algebras such as sets, posets, semilattices or vector spaces, and proved a variety theorem for $\mathcal{D}$-monoids that subsumes five different Eilenberg theorems for regular languages [12, 20, 24, 26, 31]. Subsequently, Bojańczyk [8] took an orthogonal approach: he keeps the category of sets but considers, in lieu of monoids, algebras for a *monad* on sorted sets as recognizing structures, improving earlier generic results from [4, 28].

In order to obtain the desired general variety theorem, a unification of the two approaches is required. On the one hand, one needs to take the step from sets to more general categories $\mathcal{D}$ to capture the proper notion of language recognition; for example, for the treatment of weighted languages [26] one needs to work over the category of vector spaces. On the other hand, to deal with machine behaviors beyond finite words, one has to replace monoids by other algebraic structures. The main contribution of this paper is a variety theorem that achieves the desired unification, and in addition directly encompasses many Eilenberg-type results captured by neither of the previous generic results, including the work [5, 6, 11, 21, 29, 35].

Traditionally, Eilenberg-like correspondences are proved in essentially the same four steps:

1. Identify an algebraic theory T such that the languages in mind are the ones recognized by finite T-algebras. For example, for regular languages one takes the theory of monoids.
2. Describe the syntactic T-algebras, i.e. the minimal algebraic recognizers of languages.
3. Infer the form of the derivatives under which varieties of languages are closed.
4. Establish the Eilenberg correspondence between varieties of languages and pseudovarieties of algebras by relating the languages of a variety to their corresponding syntactic algebras.

The key insight provided by our paper is that all steps can be simplified or even automated.

For Step 1, putting a common roof over Bojańczyk's and our own previous work, we consider an algebraic category $\mathcal{D}$ and algebras for a *monad* $\mathbf{T}$ on $\mathcal{D}^S$, the category of S-sorted $\mathcal{D}$-algebras for some finite set S of sorts. For example, to capture regular languages one takes the free-monoid monad $\mathbf{T}\Sigma = \Sigma^*$ on $\mathbf{Set}$. For regular ∞-languages one takes the monad $\mathbf{T}(\Sigma, \Gamma) = (\Sigma^+, \Sigma^\omega + \Sigma^* \times \Gamma)$ on $\mathbf{Set}^2$ representing ω-semigroups. For weighted languages over a finite field K, ones takes the free K-algebra monad $\mathbf{T}$ on the category of vector spaces.

For Steps 2 and 3, we develop our main technical tool, the concept of a *unary presentation* of a monad. A unary presentation expresses in categorical terms how to present $\mathbf{T}$-algebras by unary operations; for example, a monoid M is presented by the translations $x \mapsto yx$ and $x \mapsto xy$ for $y \in M$. It turns out that unary presentations are, in a precise sense, necessary and sufficient for constructing syntactic algebras (Theorem 4.7). This clarifies the role of syntactic algebras in earlier work on Eilenberg-type theorems, and the nature of derivatives appearing in varieties of languages. In our paper, syntactic algebras are *not* used for proving the variety theorem: we rely solely on the more elementary concept of a unary presentation.

We emphasize that, in general, nontrivial work lies in finding a good unary presentation for a monad $\mathbf{T}$. However, our work here shows that then Steps 3 and 4 are entirely generic: after choosing a unary presentation, we obtain a notion of *variety of* $\mathbf{T}$*-recognizable languages* (involving a notion of derivatives directly inferred from on the presentation) and the following

Variety Theorem. *Varieties of* $\mathbf{T}$*-recognizable languages are in bijective correspondence with pseudovarieties of* $\mathbf{T}$*-algebras.*

The proof has two main ingredients. The first one is duality: besides $\mathcal{D}$ we also consider a variety $\mathcal{C}$ that is dual to $\mathcal{D}$ on the level of finite algebras. Varieties of languages live in $\mathcal{C}$, while over $\mathcal{D}^S$ we form pseudovarieties of $\mathbf{T}$-algebras. This is much inspired by the work of Gehrke, Grigorieff, and Pin [14] who interpreted the original Eilenberg theorem [12] in terms of Stone duality ($\mathcal{C}$ = boolean algebras, $\mathcal{D}$ = sets). The second ingredient is the *profinite monad* of $\mathbf{T}$, introduced in [9]. It extends the classical construction of the free profinite monoid, and allows for the introduction of topological methods to our setting. The key to our approach is a categorical Reiterman theorem (Theorem 5.10) asserting that pseudovarieties of $\mathbf{T}$-algebras correspond to profinite equational theories. The Variety Theorem then boils down to the fact that (i) varieties of $\mathbf{T}$-recognizable languages dualize to profinite theories, and (ii) by the Reiterman theorem, profinite theories correspond to pseudovarieties of $\mathbf{T}$-algebras.

Our results establish a conceptual and highly parametric, yet easily applicable framework for algebraic language theory. In fact, to derive an Eilenberg correspondence in our framework, the traditional four steps indicated above are replaced by a simple three-step procedure:

1. Find a monad $\mathbf{T}$ whose finite algebras recognize the desired languages.
2. Choose a unary presentation for $\mathbf{T}$.
3. Spell out what a variety of $\mathbf{T}$-recognizable languages and a pseudovariety of $\mathbf{T}$-algebras is (by instantiating our general definitions), and invoke the Variety Theorem.

To illustrate the strength of this approach, we will show that roughly a dozen Eilenberg theorems in the literature emerge as special instances. In addition, we get several new results for free, e.g. an extension of the local variety theorem of [14] from finite to infinite words.

2 The Profinite Monad

We start by setting up our categorical framework for algebraic language theory. Recall that for a finitary one-sorted signature Γ, a *variety of algebras* is a class of Γ-algebras presented by equations between Γ-terms. A *variety of ordered algebras* is a class of ordered Γ-algebras (i.e. Γ-algebras on a poset with monotone Γ-operations) presented by inequations between Γ-terms. Morphisms of (ordered) Γ-algebras are (monotone) Γ-homomorphisms.

▶ **Assumptions 2.1.** Fix a variety $\mathcal{C}$ of algebras and a variety $\mathcal{D}$ of algebras/ordered algebras such that (i) $\mathcal{C}$ and $\mathcal{D}$ are *locally finite*, i.e. all finitely generated algebras are finite; (ii) the full subcategories $\mathcal{C}_f$ and $\mathcal{D}_f$ on finite algebras are dually equivalent; (iii) the signature of $\mathcal{C}$ contains a constant; (iv) epimorphisms in $\mathcal{D}$ are surjective. Further, fix a finite set S of sorts and a monad $\mathbf{T} = (T, \eta, \mu)$ on the product category $\mathcal{D}^S$ with T preserving epimorphisms.

▶ **Example 2.2.** The following categories $\mathcal{C}$ and $\mathcal{D}$ satisfy our assumptions:

1. $\mathcal{C} = \mathbf{BA}$ (boolean algebras) and $\mathcal{D} = \mathbf{Set}$: Stone duality [15] yields a dual equivalence $\mathbf{BA}_f^{op} \simeq \mathbf{Set}_f$, mapping a finite boolean algebra to the set of its atoms.
2. $\mathcal{C} = \mathbf{DL}_{01}$ (distributive lattices with $0, 1$) and $\mathcal{D} = \mathbf{Pos}$ (posets): Birkhoff duality [7] gives a dual equivalence $(\mathbf{DL}_{01})_f^{op} \simeq \mathbf{Pos}_f$, mapping a finite distributive lattice to the poset of its join-irreducible elements.
3. $\mathcal{C} = \mathcal{D} = \mathbf{JSL}_0$ (join-semilattices with 0): the self-duality $(\mathbf{JSL}_0)_f^{op} \simeq (\mathbf{JSL}_0)_f$ maps a finite semilattice $(X, \vee, 0)$ to its opposite semilattice $(X, \wedge, 1)$.
4. $\mathcal{C} = \mathcal{D} = \mathbf{Vec}_K$ (vector spaces over a finite field K): the familiar self-duality of $(\mathbf{Vec}_K)_f$ maps a finite (= finite-dimensional) vector space X to its dual space $X^* = \mathbf{Vec}_K(X, K)$.

▶ **Example 2.3.** Our focus is on monads $\mathbf{T}$ whose algebras represent formal languages.

1. Let $\mathbf{T}_*$ be the free-monoid monad on $\mathbf{Set}$. Languages of finite words correspond to subsets of $T_*\Sigma = \Sigma^*$. The category of $\mathbf{T}_*$-algebras is isomorphic to the category of monoids.
2. Languages of finite and infinite words (∞-*languages* for short) are represented by the monad $\mathbf{T}_\infty$ on $\mathbf{Set}^2$ associated to the algebraic theory of ω-semigroups. Recall that an ω-*semigroup* is a two-sorted set $A = (A_+, A_\omega)$ with a binary product $A_+ \times A_+ \xrightarrow{\cdot} A_+$, a mixed binary product $A_+ \times A_\omega \xrightarrow{\cdot} A_\omega$ and an ω-ary product $A_+^\omega \xrightarrow{\pi} A_\omega$ satisfying (mixed) associative laws [19]. The free ω-semigroup on a two-sorted set (Σ, Γ) is $(\Sigma^+, \Sigma^\omega + \Sigma^* \times \Gamma)$ with products given by concatenation of words. Thus $T_\infty(\Sigma, \Gamma) = (\Sigma^+, \Sigma^\omega + \Sigma^* \times \Gamma)$, and ∞-languages over the alphabet Σ are two-sorted subsets of $T_\infty(\Sigma, \emptyset) = (\Sigma^+, \Sigma^\omega)$.
3. Weighted languages $L\colon \Sigma^* \to K$ over a finite field K are represented by the free-K-algebra monad $\mathbf{T}_K$ on $\mathbf{Vec}_K$. Thus for the space K^Σ with finite basis Σ we have $T_K(K^\Sigma) = K\langle\Sigma\rangle$, the space of all polynomials $\sum_{i<n} k_i w_i$ ($k_i \in K$, $w_i \in \Sigma^*$) in non-commuting variables. Since $K\langle\Sigma\rangle$ has the basis Σ^*, weighted languages correspond to linear maps $T_K(K^\Sigma) \to K$.

▶ **Remark 2.4.** We denote by $\mathbf{Alg}_f\,\mathbf{T}$ and $\mathbf{Alg}\,\mathbf{T}$ the categories of (finite) $\mathbf{T}$-algebras. The category $\mathcal{D}^S$ has the factorization system of sortwise surjective morphisms and sortwise injective/order-reflecting morphisms. This lifts to $\mathbf{Alg}\,\mathbf{T}$ since T preserves surjections: every $\mathbf{T}$-homomorphism factors into a surjective $\mathbf{T}$-homomorphism followed by an injective/order-reflecting one. *Quotients* and *subalgebras* in $\mathbf{Alg}\,\mathbf{T}$ are taken in this factorization system.

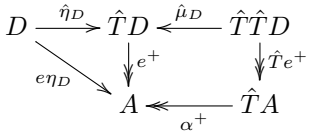
In the theory of regular languages, topology enters the stage via the Stone space $\widehat{\Sigma^*}$ of *profinite words*, formed as the inverse (a.k.a. cofiltered) limit of all finite quotient monoids of Σ^*. In [9] we generalized this construction from monoids to algebras for a monad as follows:

▶ **Notation 2.5.** For a variety $\mathcal{D}$ of algebras, let $\mathbf{Stone}(\mathcal{D})$ denote the category of *Stone-topological $\mathcal{D}$-algebras*; its objects are $\mathcal{D}$-algebras endowed with a Stone topology and continuous $\mathcal{D}$-operations, and its morphisms are continuous $\mathcal{D}$-morphisms. For a variety $\mathcal{D}$ of ordered algebras, let $\mathbf{Priest}(\mathcal{D})$ be the category of ordered topological $\mathcal{D}$-algebras with a Priestley topology, and monotone continuous $\mathcal{D}$-morphisms. Denote by $\widehat{\mathcal{D}}$ the full subcategory of $\mathbf{Stone}(\mathcal{D})/\mathbf{Priest}(\mathcal{D})$ on *profinite $\mathcal{D}$-algebras*, i.e. inverse limits of algebras in $\mathcal{D}_f$. Here we view $\mathcal{D}_f$ as a full subcategory of $\widehat{\mathcal{D}}$ by equipping objects of $\mathcal{D}_f$ with the discrete topology.

▶ **Example 2.6.** For the varieties $\mathcal{D}$ of Example 2.2, every algebra in $\mathbf{Stone}(\mathcal{D})/\mathbf{Priest}(\mathcal{D})$ is profinite: we have $\widehat{\mathbf{Set}} = \mathbf{Stone}$ (Stone spaces), $\widehat{\mathbf{Pos}} = \mathbf{Priest}$ (Priestley spaces), $\widehat{\mathbf{JSL}_0} = \mathbf{Stone}(\mathbf{JSL}_0)$ (Stone semilattices) and $\widehat{\mathbf{Vec}_K} = \mathbf{Stone}(\mathbf{Vec}_K)$ (Stone vector spaces); see [15].

▶ **Construction 2.7.** For any object $D \in \mathcal{D}_f^S$ form the poset $\mathsf{Quo}_f(\mathbf{T}D)$ of all finite quotient algebras $e\colon \mathbf{T}D \twoheadrightarrow (A, \alpha)$ of the free $\mathbf{T}$-algebra $\mathbf{T}D = (TD, \mu_D)$, ordered by $e \leq e'$ iff e factorizes through e'. Define the object $\hat{T}D$ in $\widehat{\mathcal{D}}^S$ to be the inverse limit of the diagram $\mathsf{Quo}_f(\mathbf{T}D) \to \widehat{\mathcal{D}}^S$ mapping $(e\colon \mathbf{T}D \twoheadrightarrow (A, \alpha))$ to A. We denote the limit projection associated to e by $e^+\colon \hat{T}D \twoheadrightarrow A$. In particular, for any finite $\mathbf{T}$-algebra (A, α) we have the projection $\alpha^+\colon \hat{T}A \to A$, since $\alpha\colon \mathbf{T}A \twoheadrightarrow (A, \alpha)$ is a surjective $\mathbf{T}$-homomorphism by the $\mathbf{T}$-algebra laws.

▶ **Theorem 2.8** (see [9])**.** *The object map $D \mapsto \hat{T}D$ from $\mathcal{D}_f^S$ to $\widehat{\mathcal{D}}^S$ extends (by taking inverse limits) to a functor $\hat{T}\colon \widehat{\mathcal{D}}^S \to \widehat{\mathcal{D}}^S$. Further, $\hat{T}$ can be equipped with the structure of a monad $\widehat{\mathbf{T}} = (\hat{T}, \hat{\eta}, \hat{\mu})$ called the* profinite monad *of $\mathbf{T}$. Its unit $\hat{\eta}_D$ and multiplication $\hat{\mu}_D$ for $D \in \mathcal{D}_f^S$ are uniquely determined by the commutativity of the diagrams*



for all $e\colon \mathbf{T}D \twoheadrightarrow (A, \alpha)$ in $\mathsf{Quo}_f(\mathbf{T}D)$. (1)

In categorical terms, $\widehat{\mathbf{T}}$ is the codensity monad of the functor $\mathbf{Alg}_f\,\mathbf{T} \to \widehat{\mathcal{D}}$, $(A,\alpha) \mapsto A$.

▶ **Example 2.9.** The monad $\widehat{\mathbf{T}_*}$ on **Stone** assigns to each finite set Σ the Stone space $\widehat{\Sigma^*}$ of profinite words. The monad $\widehat{\mathbf{T}_K}$ on $\mathbf{Stone}(\mathbf{Vec}_K)$ assigns to each finite vector space K^Σ the Stone vector space arising as the limit of all finite quotient spaces of $K[\Sigma]$.

▶ **Remark 2.10.**

1. If (A,α) is a finite $\mathbf{T}$-algebra, then (A,α^+) is a $\widehat{\mathbf{T}}$-algebra: putting $e=\alpha$ in (1) gives the unit and associative law. By [9, Prop. 3.10] this yields an isomorphism $\mathbf{Alg}_f\,\mathbf{T} \cong \mathbf{Alg}_f\,\widehat{\mathbf{T}}$ given by $(A,\alpha)\mapsto(A,\alpha^+)$ and $h\mapsto h$.
2. Let $V\colon \widehat{\mathcal{D}}^S \to \mathcal{D}^S$ denote the forgetful functor. If $D \in \widehat{\mathcal{D}}^S_f\ (=\mathcal{D}^S_f)$ we usually write D for VD. By [9, Rem. B.6] there is a natural transformation $\iota\colon TV \to V\hat{T}$ whose component $\iota_D\colon TVD \to V\hat{T}D$ for $D\in\mathcal{D}^S_f$ is determined by $Ve^+\cdot\iota_D = e$ for all $e\in \mathsf{Quo}_f(\mathbf{T}D)$.

▶ **Remark 2.11.**

1. $\widehat{\mathcal{D}}$ is the *pro-completion* (the free completion under inverse limits) of $\mathcal{D}_f$, see [15, Rem. VI.2.4]. Further, since the variety $\mathcal{C}$ is locally finite, $\mathcal{C}$ is the *ind-completion* (the free completion under filtered colimits) of $\mathcal{C}_f$. Therefore the dual equivalence between $\mathcal{C}_f$ and $\mathcal{D}_f$ extends to a dual equivalence between $\mathcal{C}$ and $\widehat{\mathcal{D}}$. We denote the equivalence functors by $P\colon \widehat{\mathcal{D}} \xrightarrow{\simeq} \mathcal{C}^{op}$ and $P^{-1}\colon \mathcal{C}^{op} \xrightarrow{\simeq} \widehat{\mathcal{D}}$. For example, for $\mathcal{C}/\mathcal{D} = \mathbf{BA}/\mathbf{Set}$ with $\widehat{\mathcal{D}} = \mathbf{Stone}$, this is the classical Stone duality [15]: P maps a Stone space to the boolean algebra of clopens, and P^{-1} maps a boolean algebra to the Stone space of all ultrafilters.
2. Denote by $|-|$ the forgetful functors of $\mathcal{C}$ and $\widehat{\mathcal{D}}$ into **Set** and by $\mathbb{1}_\mathcal{C}/\mathbb{1}_\mathcal{D}$ the free one-generated objects in $\mathcal{C}/\widehat{\mathcal{D}}$. The two finite objects $O_\mathcal{C} := P\mathbb{1}_\mathcal{D}$ and $O_\mathcal{D} := P^{-1}\mathbb{1}_\mathcal{C}$ play the role of a *dualizing object* [15] of $\mathcal{C}$ and $\widehat{\mathcal{D}}$. This means that there is a natural isomorphism between the functors $|-|\cdot P^{op}$ and $\widehat{\mathcal{D}}(-,O_\mathcal{D})\colon \widehat{\mathcal{D}}^{op}\to\mathbf{Set}$, given for any $D\in\widehat{\mathcal{D}}$ by
$$|PD| \cong \mathcal{C}(\mathbb{1}_\mathcal{C}, PD) \cong \widehat{\mathcal{D}}(P^{-1}PD, P^{-1}\mathbb{1}_\mathcal{C}) = \widehat{\mathcal{D}}(P^{-1}PD, O_\mathcal{D}) \cong \widehat{\mathcal{D}}(D, O_\mathcal{D}).$$
Similarly, $|-|\cdot P^{-1} \cong \mathcal{C}(-,O_\mathcal{C})$. Thus $O_\mathcal{C}$ and $O_\mathcal{D}$ have essentially the same underlying set, as $|O_\mathcal{D}| \cong \widehat{\mathcal{D}}(\mathbb{1}_\mathcal{D}, O_\mathcal{D}) \cong |P\mathbb{1}_\mathcal{D}| = |O_\mathcal{C}|$. For our pairs $\mathcal{C}/\mathcal{D}$ of Ex. 2.2, we get $O_{\mathbf{BA}} = \{0<1\}/O_{\mathbf{Set}} = \{0,1\}$, $O_{\mathbf{DL}_{01}} = \{0<1\} = O_{\mathbf{Pos}}$, $O_{\mathbf{JSL}_0} = \{0<1\}$, $O_{\mathbf{Vec}_K} = K$.
3. *Subobjects* in $\mathcal{C}$ are represented by monomorphisms, i.e. injective morphisms. Dually, *quotients* in $\widehat{\mathcal{D}}$ are represented by epimorphisms, which can be shown to be the surjective morphisms. Thus *quotients* of $\widehat{\mathbf{T}}$-algebras are represented by surjective $\widehat{\mathbf{T}}$-homomorphisms.

3 Recognizable Languages

A language $L\subseteq\Sigma^*$ of finite words may be identified with its characteristic function $L\colon\Sigma^*\to\{0,1\}$. To model languages in our categorical setting, we replace the one-sorted alphabet Σ by an S-sorted alphabet Σ in $\mathbf{Set}^S_f$, and represent it in $\mathcal{D}^S$ via the free object $\not\asymp\in\mathcal{D}^S_f$ generated by Σ (w.r.t. the forgetful functor $|-|\colon\mathcal{D}^S\to\mathbf{Set}^S$). Note that $\not\asymp$ is finite because $\mathcal{D}$ is locally finite. The output set $\{0,1\}$ is replaced by a finite "object of outputs" in $\mathcal{D}^S_f$, viz. the object with $O_\mathcal{D}\in\mathcal{D}_f$ in each sort. By abuse of notation, we denote this object of $\mathcal{D}^S_f$ also by $O_\mathcal{D}$. This leads to the following definition, unifying concepts from [2] and [8].

▶ **Definition 3.1.** A *language* over the alphabet $\Sigma\in\mathbf{Set}^S_f$ is a morphism $L\colon T\not\asymp\to O_\mathcal{D}$ in $\mathcal{D}^S$. It is called $\mathbf{T}$-*recognizable* if there exists a $\mathbf{T}$-homomorphism $h\colon\mathbf{T}\not\asymp\to(A,\alpha)$ with finite codomain and a morphism $p\colon A\to O_\mathcal{D}$ in $\mathcal{D}^S$ with $L=p\cdot h$. In this case, we say that L *is recognized by* h *(via* p*)*. We denote by $\mathsf{Rec}(\Sigma)$ the set of all $\mathbf{T}$-recognizable languages over Σ.

▶ **Example 3.2.**

1. $\mathbf{T} = \mathbf{T}_*$ on $\mathbf{Set}$ with $O_{\mathbf{Set}} = \{0,1\}$: a language $L\colon T_*\Sigma \to O_{\mathbf{Set}}$ corresponds to a language $L \subseteq \Sigma^*$ of finite words. It is recognized by a monoid morphism $h\colon \Sigma^* \to A$ iff $L = h^{-1}[Y]$ for some subset $Y \subseteq A$. Recognizable languages coincide with regular languages, i.e. languages accepted by finite automata; see e.g. [22].
2. $\mathbf{T} = \mathbf{T}_\infty$ on $\mathbf{Set}^2$ with $O_{\mathbf{Set}} = \{0,1\}$: since $T_\infty(\Sigma,\emptyset) = (\Sigma^+, \Sigma^\omega)$, a language $L\colon T_\infty(\Sigma,\emptyset) \to (\{0,1\},\{0,1\})$ corresponds to an ∞-language $L \subseteq (\Sigma^+,\Sigma^\omega)$. It is recognized by an ω-semigroup morphism $h\colon (\Sigma^+,\Sigma^\omega) \to A$ iff $L = h^{-1}[Y]$ for some two-sorted subset $Y \subseteq A$. Recognizable ∞-languages are the ones accepted by finite Büchi automata [19].

The topological perspective on regular languages rests on the important observation that regular languages over Σ correspond to the clopen subsets of the Stone space $\widehat{\Sigma^*}$ of profinite words, or equivalently to continuous maps from $\widehat{\Sigma^*}$ into the discrete space $\{0,1\}$; see e.g. [22, Prop. VI.3.12]. This generalizes from the monad $\mathbf{T}_*$ on $\mathbf{Set}$ to arbitrary monads $\mathbf{T}$:

▶ **Theorem 3.3.** *$\mathbf{T}$-recognizable languages over $\Sigma \in \mathbf{Set}_f^S$ correspond bijectively to morphisms from $\hat{T}\not\prec$ to $O_\mathcal{D}$ in $\widehat{\mathcal{D}}^S$. The bijection is given by $(\hat{T}\not\prec \xrightarrow{\hat{L}} O_\mathcal{D}) \mapsto (T\not\prec \xrightarrow{\iota\not\prec} V\hat{T}\not\prec \xrightarrow{V\hat{L}} O_\mathcal{D})$.*

▶ **Remark 3.4** (𝒞-algebraic structure on $\mathsf{Rec}(\Sigma)$)**.** By the above and Remark 2.11.2, we deduce

$$\mathsf{Rec}(\Sigma) \cong \widehat{\mathcal{D}}^S(\hat{T}\not\prec, O_\mathcal{D}) = \prod_s \widehat{\mathcal{D}}((\hat{T}\not\prec)_s, O_\mathcal{D}) \cong \prod_s |P(\hat{T}\not\prec)_s|. \tag{2}$$

Thus we can consider $\mathsf{Rec}(\Sigma)$ as an object of $\mathcal{C}$ isomorphic to $\prod_s P(\hat{T}\not\prec)_s$. One can show that $\mathsf{Rec}(\Sigma)$ is a subobject of the product $\prod_s O_\mathcal{C}^{|T\not\prec|_s}$ in $\mathcal{C}$: the embedding $\mathsf{Rec}(\Sigma) \rightarrowtail \prod_s O_\mathcal{C}^{|T\not\prec|_s}$ maps a language $L\colon T\not\prec \to O_\mathcal{D}$ to the S-tuple $(|T\not\prec|_s \xrightarrow{|L|_s} |O_\mathcal{D}| \xrightarrow{\cong} |O_\mathcal{C}|)_{s\in S}$, using the bijection $|O_\mathcal{D}| \cong |O_\mathcal{C}|$ of Remark 2.11.2. Consequently the $\mathcal{C}$-algebraic structure of $\mathsf{Rec}(\Sigma)$ is determined by $O_\mathcal{C}$. For example, for $\mathcal{C} = \mathbf{BA}$ with $O_{\mathbf{BA}} = \{0,1\}$, the boolean structure on $\mathsf{Rec}(\Sigma)$ is given by union, intersection and complement. Taking $\mathbf{T} = \mathbf{T}_*$ on $\mathbf{Set}$, we recover an important result of Pippenger [23]: the boolean algebra of regular languages over Σ is dual to the Stone space $\widehat{\Sigma^*}$ of profinite words. In fact, in this case (2) yields $\mathsf{Rec}(\Sigma) \cong P(\widehat{\Sigma^*})$.

4 Unary Presentations

In this section we introduce *unary presentations* of $\mathbf{T}$-algebras that later, in Section 6, will serve as our key tool for defining the derivatives of a language. For motivation, let A be an algebra over a finitary single-sorted signature Γ. By a standard result in universal algebra, an equivalence relation $\equiv$ on A is a Γ-congruence (i.e. stable under Γ-operations) iff it is stable under *elementary translations*, i.e. $a \equiv a'$ implies $u(a) \equiv u(a')$ for all maps $u\colon A \to A$ of the form $a \mapsto \gamma^A(a_0,\dots,a_i,a,a_{i+1},\dots a_n)$, where $\gamma \in \Gamma$ and $a_0,\dots a_n \in A$. (For the sorted case, see [17,18].) If Γ contains infinitary operations, this statement generally fails, but remains valid if $\equiv$ is refinable to a Γ-congruence of finite index; see Examples 4.3.3/5. Identifying equivalence relations with quotients, we first state the concept of refinement categorically:

▶ **Definition 4.1.** Let $(A,\alpha) \in \mathbf{Alg}\,\mathbf{T}$. A quotient $e\colon A \twoheadrightarrow B$ in $\mathcal{D}^S$ is $\mathbf{T}$-*refinable* if there is a finite quotient $\bar{e}\colon (A,\alpha) \twoheadrightarrow (C,\gamma)$ in $\mathbf{Alg}\,\mathbf{T}$ and a morphism $p\colon C \twoheadrightarrow B$ in $\mathcal{D}^S$ with $e = p\cdot\bar{e}$.

Then the description of congruences via translations has the following categorical formulation:

▶ **Definition 4.2.** A *unary operation* on $A \in \mathcal{D}^S$ is a morphism $u\colon A_s \to A_t$ in $\mathcal{D}$, where $s,t \in S$. A *unary presentation* of a $\mathbf{T}$-algebra (A,α) is a set $\mathbb{U}$ of unary operations on A such that, for any $\mathbf{T}$-refinable quotient $e\colon A \twoheadrightarrow B$ in $\mathcal{D}^S$, the following are equivalent:

(U1) e carries a quotient of (A,α) in $\mathbf{Alg}\,\mathbf{T}$, i.e. there exists a $\mathbf{T}$-algebra structure (B,β) on B for which $e\colon (A,\alpha) \twoheadrightarrow (B,\beta)$ is a $\mathbf{T}$-homomorphism.

(U2) Each unary operation $u\colon A_s \to A_t$ in $\mathbb{U}$ admits a lifting along e, i.e. a morphism $u_B\colon B_s \to B_t$ in $\mathcal{D}$ with $e_t \cdot u = u_B \cdot e_s$.

▶ **Example 4.3.**

1. $\mathbf{T} = \mathbf{T}_*$ on $\mathbf{Set}$: every monoid M has a unary presentation given by the unary operations $x \mapsto yx$ and $x \mapsto xy$ on M, where y ranges over all elements of M.
2. $\mathbf{T} = \mathbf{T}_\infty$ on $\mathbf{Set}^2$: every ω-semigroup $A = (A_+, A_\omega)$ has a unary presentation given by the operations (i) $x \mapsto yx$ and $x \mapsto xy$ on A_+, (ii) $x \mapsto xz$ and $x \mapsto x^\omega = \pi(x,x,x,\ldots)$ from A_+ to A_ω, and (iii) $z \mapsto yz$ on A_ω, where $y \in A_+ \cup \{1\}$ and $z \in A_\omega$. The proof uses Ramsey's theorem and appears implicitly in the work of Wilke [35]; see also [19].
3. Let $\mathbf{T}$ be a monad on $\mathbf{Set}^S$. Every $\mathbf{T}$-algebra (A,α) has a generic unary presentation given as follows. Let $1_s \in \mathbf{Set}^S$ be the S-sorted set with one element in sort s and otherwise empty; thus a morphism $1_s \to A$ in $\mathbf{Set}^S$ chooses an element of A_s. A *polynomial* over A is a morphism $p\colon 1_t \to T(A + 1_s)$ with $s,t \in S$, i.e. a "term" of output sort t in a variable of sort s. Denote by $A_s \xrightarrow{[p]} A_t$ the evaluation map that substitutes elements of A_s for the variable. The maps $[p]$ (where p ranges over polynomials over A) form a unary presentation of (A,α). The proof rests on the fact that $\mathbf{T}$-algebras can be viewed as algebras over a (possibly large and infinitary) signature [16]. Note that for monoids and ω-semigroups, the polynomial presentation is much larger than the one in Example 1/2; e.g., for a monoid M it contains all operations $x \mapsto y_0 x y_1 x \ldots x y_n$ with $y_0,\ldots,y_n \in M$. Polynomials appeared in [8] in the context of syntactic congruences; see Example 4.8.
4. In contrast to Example 4.3.3, in general not every $\mathbf{T}$-algebra admits a unary presentation if $\mathcal{D} \neq \mathbf{Set}$. Indeed, let $\mathcal{D} = \mathbf{Set}_{c,d}$ be the variety of sets with two constants c,d, and $\mathbf{Set}_{c\neq d}$ its full reflective subcategory on the terminal object 1 and all $X \in \mathcal{D}$ with $c^X \neq d^X$. The right adjoint $\mathbf{Set}_{c\neq d} \rightarrowtail \mathcal{D}$ induces a monad $\mathbf{T}$ on $\mathcal{D}$ with $\mathbf{Alg}\,\mathbf{T} \cong \mathbf{Set}_{c\neq d}$. One can show that the $\mathbf{T}$-algebra corresponding to $\{x,c,d\} \in \mathbf{Set}_{c\neq d}$ has no unary presentation.
5. If $\mathbf{T}$ represents algebras with finitary operations, the equivalence (U1)⇔(U2) often holds for arbitrary quotients e. However, the restriction to $\mathbf{T}$-*refinable* quotients is crucial in the presence of infinitary operations. For example, let $\mathbf{T}$ be the free Γ-algebra monad on $\mathbf{Set}$ for the signature Γ with one ω-ary operation. Thus TX is the set of well-founded Γ-trees (= ω-branching trees without infinite paths) with X-labeled leaves. Let $X \neq \emptyset$ and $e : TX \twoheadrightarrow \{0,1\}$ be the map sending a tree t to 0 iff t has finite height. Then for the polynomial presentation of $\mathbf{T}$, see Ex. 4.3.3, the direction (U2)⇒(U1) does not hold for e.

Digression: Syntactic T-algebras

Languages are often analyzed by means of *syntactic algebras*, i.e. their minimal recognizing algebras. This language-theoretic concept is closely related to our algebraic notion of a unary presentation, as we now explain. The results of this subsection serve to put our concepts into the context of classical algebraic language theory; they are, however, not used in the sequel and may be skipped by readers interested only in the variety theorem and its applications.

▶ **Definition 4.4.** Let $L\colon T\not\to \to O_{\mathcal{D}}$ be recognizable. A *syntactic* $\mathbf{T}$-*algebra for* L is a finite $\mathbf{T}$-algebra A_L together with a surjective $\mathbf{T}$-homomorphism $e_L\colon \mathbf{T}\not\to \twoheadrightarrow A_L$ (called a *syntactic morphism* for L) such that (i) e_L recognizes L, and (ii) e_L factors through any surjective $\mathbf{T}$-homomorphism $e\colon \mathbf{T}\not\to \twoheadrightarrow A$ recognizing L, i.e. $e_L = h \cdot e$ for some $h\colon A \twoheadrightarrow A_L$ in $\mathbf{Alg}\,\mathbf{T}$.

▶ **Example 4.5.** Let $\mathbf{T} = \mathbf{T}_*$ on $\mathbf{Set}$. The *syntactic monoid* [22] of a recognizable language $L\colon \Sigma^* \to \{0,1\}$ is the quotient monoid $e_L\colon \Sigma^* \twoheadrightarrow \Sigma^*/\equiv_L$, where $\equiv_L$ is the monoid congruence on Σ^* defined by $x \equiv_L x'$ iff $L(yxz) = L(yx'z)$ for all $y, z \in \Sigma^*$.

The above definition of $\equiv_L$ involves for each $y, z \in \Sigma^*$ the unary operation $x \mapsto yxz$ on Σ^*, which can be expressed as the composite of the operations $x \mapsto yx$ and $x \mapsto xz$ appearing in the unary presentation of Σ^* in Example 4.3.1. This is no coincidence: one can always derive a syntactic congruence from a unary presentation, and vice versa. For brevity, we discuss only the case where $\mathcal{D}$ is a variety of algebras; see the full paper [34] for the ordered case.

▶ **Notation 4.6.** Let $\mathbb{U}$ be a set of unary operations on $T\not\leftarrow$, and denote by $\overline{\mathbb{U}}$ its closure under composition and identity morphisms $id\colon (T\not\leftarrow)_s \to (T\not\leftarrow)_s$. Given a language $L\colon T\not\leftarrow \to O_{\mathcal{D}}$, the S-sorted equivalence relation $\equiv_{\mathbb{U},L}$ on $|T\not\leftarrow|$ is defined as follows: for $x, x' \in |T\not\leftarrow|_s$, put

$$x \equiv_{\mathbb{U},L} x' \quad \Leftrightarrow \quad L_t \cdot u(x) = L_t \cdot u(x') \text{ for all sorts } t \text{ and all } u\colon (T\not\leftarrow)_s \to (T\not\leftarrow)_t \text{ in } \overline{\mathbb{U}}.$$

One readily verifies that $\equiv_{\mathbb{U},L}$ is a congruence on $T\not\leftarrow$ in $\mathcal{D}^S$, i.e. sortwise stable under all $\mathcal{D}$-operations. We denote the induced quotient in $\mathcal{D}^S$ by $e_L\colon T\not\leftarrow \twoheadrightarrow T\not\leftarrow/\equiv_{\mathbb{U},L}$.

▶ **Theorem 4.7.** *For any set $\mathbb{U}$ of unary operations on $T\not\leftarrow$, the following are equivalent:*

(i) *$\mathbb{U}$ is a unary presentation of $\mathbf{T}\not\leftarrow$.*

(ii) *Every recognizable language $L\colon T\not\leftarrow \to O_{\mathcal{D}}$ has a syntactic $\mathbf{T}$-algebra, and $e_L\colon T\not\leftarrow \twoheadrightarrow T\not\leftarrow/\equiv_{\mathbb{U},L}$ carries a quotient of $\mathbf{T}\not\leftarrow$ in $\mathbf{Alg}\,\mathbf{T}$ that forms a syntactic morphism for L.*

▶ **Example 4.8.** Let $\mathbb{U}$ be the unary presentation of Σ^* in Example 4.3.1. Then $\equiv_{\mathbb{U},L}$ is precisely the congruence $\equiv_L$ of Example 4.5, and Theorem 4.7 shows that $\equiv_L$ is a syntactic congruence for L. Similarly, Example 4.3.2/3 and Theorem 4.7 give a description of the syntactic ω-semigroup for any recognizable ∞-language [19], and of the syntactic $\mathbf{T}$-algebra for any monad $\mathbf{T}$ on $\mathbf{Set}^S$ and any $\mathbf{T}$-recognizable language [8]. We omit the details.

Theorem 4.7 explains why syntactic algebras are presented as a key technique in earlier work on Eilenberg theorems: they implicitly contain unary presentations. However, the latter are the "heart of the matter", and it is easier to directly work with presentations in lieu of syntactic algebras to derive Eilenberg-type theorems. We will demonstrate this in Section 7.

5 Pseudovarieties of T-algebras and Profinite Theories

In this section we investigate pseudovarieties of $\mathbf{T}$-algebras and establish a categorical Reiterman theorem: pseudovarieties correspond to profinite equational theories. The results of the present section are largely independent of our Assumptions 2.1: they hold for any locally finite variety $\mathcal{D}$ of (ordered) algebras and any monad $\mathbf{T}$ on $\mathcal{D}^S$ that preserves surjections.

▶ **Definition 5.1.** A *Σ-generated $\mathbf{T}$-algebra* is a quotient $e\colon \mathbf{T}\not\leftarrow \twoheadrightarrow A$ of $\mathbf{T}\not\leftarrow$ in $\mathbf{Alg}\,\mathbf{T}$. The *subdirect product* of two quotients $e_i\colon \mathbf{T}\not\leftarrow \twoheadrightarrow A_i$ $(i = 0, 1)$ is the image $e\colon \mathbf{T}\not\leftarrow \twoheadrightarrow A$ of the $\mathbf{T}$-homomorphism $\langle e_0, e_1\rangle\colon \mathbf{T}\not\leftarrow \to A_0 \times A_1$. We say that e_1 is a *quotient* of e_0 if e_1 factors through e_0, i.e. $e_1 = q \cdot e_0$ for some q. A *local pseudovariety of Σ-generated $\mathbf{T}$-algebras* is a class of Σ-generated finite $\mathbf{T}$-algebras closed under subdirect products and quotients.

In order-theoretic terms, local pseudovarieties are precisely the ideals of the poset $\mathsf{Quo}_f(\mathbf{T}\not\leftarrow)$.

▶ **Definition 5.2.** A $\widehat{\mathbf{T}}$-algebra is *profinite* if it is an inverse limit of finite $\widehat{\mathbf{T}}$-algebras. By a *Σ-generated profinite $\widehat{\mathbf{T}}$-algebra* is meant a quotient $\varphi\colon \widehat{\mathbf{T}}\not\leftarrow \twoheadrightarrow P$ of $\widehat{\mathbf{T}}\not\leftarrow$ in $\mathbf{Alg}\,\widehat{\mathbf{T}}$ with P profinite. Σ-generated profinite $\widehat{\mathbf{T}}$-algebras are ordered by $\varphi \leq \varphi'$ iff φ factors through φ'.

▶ **Theorem 5.3** (Local Reiterman Theorem). *For each $\Sigma \in \mathbf{Set}_f^S$, the lattice of local pseudovarieties of Σ-generated $\mathbf{T}$-algebras (ordered by inclusion) is isomorphic to the lattice of Σ-generated profinite $\widehat{\mathbf{T}}$-algebras. The isomorphism maps a Σ-generated profinite $\widehat{\mathbf{T}}$-algebra $\varphi\colon \widehat{\mathbf{T}}\not\leftarrow \twoheadrightarrow P$ to the local pseudovariety of all finite quotients $e\colon \mathbf{T}\not\leftarrow \twoheadrightarrow A$ with $e^+ \leq \varphi$.*

▶ **Remark 5.4.** Theorem 5.3 can be interpreted in terms of profinite (in-)equations. If $\mathcal{D}$ is a variety of ordered algebras, a *profinite inequation* $u \leq v$ *over* Σ is a pair of elements $u, v \in |\hat{T}\not\leftarrow|_s$ in some sort s. We say that a Σ-generated finite $\mathbf{T}$-algebra $e\colon \mathbf{T}\not\leftarrow \twoheadrightarrow A$ *satisfies* $u \leq v$ if $e^+(u) \leq e^+(v)$. Using 5.3 one can show that local pseudovarieties are precisely the classes of Σ-generated finite $\mathbf{T}$-algebras presentable by profinite inequations over Σ. Similarly, if $\mathcal{D}$ is a variety of algebras, local pseudovarieties are presentable by *profinite equations*.

Eilenberg's theorem considers regular languages over arbitrary alphabets. In contrast, in a sorted setting one may need to make a *choice* of alphabets to capture the proper languages (e.g. alphabets of the form $(\Sigma, \emptyset)$ in 2.3.2). On the algebraic side, this requires us to restrict to $\mathbf{T}$-algebras with certain generators. From now on, let $\mathbb{A} \subseteq \mathbf{Set}_f^S$ be a fixed class of alphabets.

▶ **Definition 5.5.** A $\mathbf{T}$-algebra (A, α) is $\mathbb{A}$-*generated* if there is a surjective $\mathbf{T}$-homomorphism $e\colon \mathbf{T}\not\leftarrow \twoheadrightarrow (A, \alpha)$ with $\Sigma \in \mathbb{A}$. A *pseudovariety of* $\mathbf{T}$-*algebras* is a class $\mathcal{V}$ of $\mathbb{A}$-generated finite $\mathbf{T}$-algebras closed under $\mathbb{A}$-generated subalgebras of finite products (i.e. for $A_1, \ldots, A_n \in \mathcal{V}$ and any $\mathbb{A}$-generated subalgebra $A \rightarrowtail \prod_{i=1}^n A_i$, one has $A \in \mathcal{V}$) and quotients.

N.B. We emphasize that, in contrast to Definition 5.1, an $\mathbb{A}$-generated $\mathbf{T}$-algebra (A, α) is not equipped with a *fixed* quotient $e\colon \mathbf{T}\not\leftarrow \twoheadrightarrow (A, \alpha)$. Only the existence of e is required.

▶ **Example 5.6.**

1. Every finite $\mathbf{T}$-algebra (A, α) is $\mathbf{Set}_f^S$-generated: since $\mathcal{D}$ is locally finite, there is a surjective morphism $e\colon \not\leftarrow \twoheadrightarrow A$ with $\Sigma \in \mathbf{Set}_f^S$, so (A, α) is a quotient of $\mathbf{T}\not\leftarrow$ via $(\mathbf{T}\not\leftarrow \xrightarrow{Te} \mathbf{T}A \xrightarrow{\alpha} (A, \alpha))$. Consequently, for $\mathbb{A} = \mathbf{Set}_f^S$, a pseudovariety is a class of finite $\mathbf{T}$-algebras closed under quotients, subalgebras, and finite products. This concept was studied in [9]. For the monad $\mathbf{T}_*$ on $\mathbf{Set}$ we recover the original concept of Eilenberg [12]: a class of finite monoids closed under quotients, submonoids, and finite products.
2. Let $\mathbf{T} = \mathbf{T}_\infty$ on $\mathbf{Set}^2$. As suggested by Example 2.3.2, we choose $\mathbb{A} = \{\, (\Sigma, \emptyset) : \Sigma \in \mathbf{Set}_f \,\}$. A finite $\mathbf{T}_\infty$-algebra (= finite ω-semigroup) A is $\mathbb{A}$-generated iff it is *complete*, i.e. every element $a \in A_\omega$ can be expressed as an infinite product $a = \pi(a_0, a_1, \ldots)$ for some $a_i \in A_+$. Clearly complete ω-semigroups are closed under finite products. Thus a pseudovariety of $\mathbf{T}_\infty$-algebras is a class of finite complete ω-semigroups closed under quotients, complete ω-subsemigroups, and finite products. This concept is due to Wilke [35]; see also [19].

▶ **Remark 5.7.** Every $\mathbf{T}$-homomorphism $g\colon \mathbf{T}D' \to \mathbf{T}D$ with $D, D' \in \mathcal{D}_f^S$ extends uniquely to a $\widehat{\mathbf{T}}$-homomorphism $\hat{g}\colon \widehat{\mathbf{T}}D' \to \widehat{\mathbf{T}}D$ with $\iota_D \cdot g = V\hat{g} \cdot \iota_{D'}$ (for ι_D recall Remark 2.10.2).

▶ **Definition 5.8.** A *profinite theory* is a family $\varrho = (\, \varrho_\Sigma\colon \widehat{\mathbf{T}}\not\leftarrow \twoheadrightarrow P_\Sigma \,)_{\Sigma\in\mathbb{A}}$ such that (i) ϱ_Σ is a Σ-generated profinite $\widehat{\mathbf{T}}$-algebra for each $\Sigma \in \mathbb{A}$, and (ii) for every $\mathbf{T}$-homomorphism $g\colon \mathbf{T}\not\leftarrow \to \mathbf{T}\not\leftarrow$ with $\Sigma, \Delta \in \mathbb{A}$, there exists a $\widehat{\mathbf{T}}$-homomorphism $g_P\colon P_\Delta \to P_\Sigma$ with $\varrho_\Sigma \cdot \hat{g} = g_P \cdot \varrho_\Delta$. Profinite theories are ordered by $\varrho \leq \varrho'$ iff ϱ_Σ factors through ϱ'_Σ for each $\Sigma \in \mathbb{A}$.

▶ **Remark 5.9.** Profinite theories generalize the *varieties of filters of congruences* introduced by Almeida [4] for algebras over a finitary signature, and earlier by Thérien [32] for monoids.

▶ **Theorem 5.10** (Reiterman Theorem). *The lattice of pseudovarieties of* $\mathbf{T}$*-algebras (ordered by inclusion) is isomorphic to the lattice of profinite theories. The isomorphism maps a theory* $(\varrho_\Sigma\colon \widehat{\mathbf{T}}\not\prec \twoheadrightarrow P_\Sigma)_{\Sigma\in\mathbb{A}}$ *to the class of all finite* $\mathbf{T}$*-algebras arising as a quotient of some* P_Σ*.*

▶ **Remark 5.11.** Theorem 5.10 has again an interpretation in terms of profinite (in-)equations. If $\mathcal{D}$ is a variety of ordered algebras, we say that a finite $\mathbf{T}$-algebra (A,α) *satisfies* a profinite inequation $u \le v$ over $\Sigma \in \mathbb{A}$ if $h(u) \le h(v)$ for all $\widehat{\mathbf{T}}$-homomorphisms $h\colon \widehat{\mathbf{T}}\not\prec \to (A,\alpha^+)$. Using 5.10 one can show that pseudovarieties are precisely the classes of $\mathbb{A}$-generated finite $\mathbf{T}$-algebras presentable by profinite inequations over $\mathbb{A}$. In the unordered case, one takes profinite equations $u = v$. For $\mathbb{A} = \mathbf{Set}_f^S$, this was proved in [9, Thm. 4.12, Rem. 5.7].

6 The Variety Theorem

To investigate varieties of languages in our categorical setting, we need a notion of *language derivatives* extending the classical concept. To this end, we make use of our Assumption 2.1(iii) that the variety $\mathcal{C}$ has a constant in the signature. Choosing a constant gives a natural transformation from the constant functor $C_{\mathbb{1}_\mathcal{C}}$ on $\mathbb{1}_\mathcal{C}$ to the identity functor $\mathsf{Id}_\mathcal{C}$. It dualizes to a natural transformation $\bot\colon \mathsf{Id}_{\widehat{\mathcal{D}}} \to C_{O_\mathcal{D}}$. The purpose of $\bot$ is to model the empty set:

▶ **Example 6.1.** For our categories $\mathcal{D}$ of Example 2.2 and the corresponding objects $O_\mathcal{D}$ (see Remark 2.11.2) we choose $\bot\colon D \to O_\mathcal{D}$ for $D \in \widehat{\mathcal{D}}$ to be the constant morphism with value 0.

▶ **Definition 6.2.** Let $L\colon T\not\prec \to O_\mathcal{D}$ be a language. Then we define the following languages:

1. the *preimage* $g^{-1}L$ of L under a $\mathbf{T}$-homomorphism $g\colon \mathbf{T}\gtrdot \to \mathbf{T}\not\prec$ by $T\gtrdot \xrightarrow{g} T\not\prec \xrightarrow{L} O_\mathcal{D}$;
2. the *derivative* $u^{-1}L\colon T\not\prec \to O_\mathcal{D}$ of L w.r.t. a unary operation $u\colon (T\not\prec)_s \to (T\not\prec)_t$ by

$$(u^{-1}L)_s = (T\not\prec)_s \xrightarrow{u} (T\not\prec)_t \xrightarrow{L_t} O_\mathcal{D} \quad ; \quad (u^{-1}L)_r = (T\not\prec)_r \xrightarrow{\iota_\not\prec} (V\hat{T}\not\prec)_r \xrightarrow{V\bot} O_\mathcal{D} \ (r \neq s)$$

N.B. In the single-sorted case $S = 1$ the derivative $u^{-1}L$ is equal to $L \cdot u$ and the natural transformation $\bot$ is not used. Therefore Assumption 2.1(iii) can be dropped.

▶ **Example 6.3.**

1. $\mathbf{T} = \mathbf{T}_*$ on $\mathbf{Set}$: consider the unary operations of Example 4.3.1 presenting Σ^*. The induced derivatives of a language $L \subseteq \Sigma^*$ are exactly the classical ones, i.e. the languages $y^{-1}L = \{\, x \in \Sigma^* : yx \in L \,\}$ and $Ly^{-1} = \{\, x \in \Sigma^* : xy \in L \,\}$ for $y \in \Sigma^*$.
2. $\mathbf{T} = \mathbf{T}_\infty$ on $\mathbf{Set}^2$: consider the unary operations of Example 4.3.2 presenting $\mathbf{T}_\infty(\Sigma,\emptyset) = (\Sigma^+,\Sigma^\omega)$. The induced derivatives of $L \subseteq \Sigma^+ \cup \Sigma^\omega$ are $\{\, x \in \Sigma^+ : yx \in L \,\}$, $\{\, x \in \Sigma^+ : xy \in L \,\}$, $\{\, x \in \Sigma^+ : xz \in L \,\}$, $\{\, x \in \Sigma^+ : x^\omega \in L \,\}$, and $\{\, z \in \Sigma^\omega : yz \in L \,\}$, where $y \in \Sigma^*$ and $z \in \Sigma^\omega$. These are the derivatives for ∞-languages studied by Wilke [35].
3. Let $\mathbf{T}$ be any monad on $\mathbf{Set}^S$, and consider the unary operations $[p]$ of Example 4.3.3 presenting $\mathbf{T}\Sigma$. The induced derivatives of $L \subseteq T\Sigma$ are the languages $p^{-1}L \subseteq T\Sigma$ with $(p^{-1}L)_s = \{\, x \in (T\Sigma)_s : [p](x) \in L_t \,\}$ and $(p^{-1}L)_r = \emptyset$ for $r \neq s$, where $p\colon 1_t \to T(T\Sigma + 1_s)$ is a polynomial. These *polynomial derivatives* were studied by Bojańczyk [8].

▶ **Definition 6.4.** Given an S-indexed family $\mathcal{L} = (\, L^s\colon T\not\prec \to O_\mathcal{D} \,)_{s\in S}$ of languages over Σ, the *diagonal* of $\mathcal{L}$ is the language $\Delta\mathcal{L}$ over Σ with $(\Delta\mathcal{L})_s = L^s_s\colon (T\not\prec)_s \to O_\mathcal{D}$ for all $s \in S$.

▶ **Notation 6.5.** Recall that we work with a fixed class $\mathbb{A} \subseteq \mathbf{Set}_f^S$ of alphabets. Fix for each $\Sigma \in \mathbb{A}$ a unary presentation $\mathbb{U}_\Sigma$ of the free $\mathbf{T}$-algebra $\mathbf{T}\not\prec$.

▶ **Definition 6.6.**

1. A *local variety of languages* over $\Sigma \in \mathbb{A}$ is a subobject $W_\Sigma \subseteq \mathsf{Rec}(\Sigma)$ in $\mathcal{C}$ closed under $\mathbb{U}_\Sigma$-derivatives ($L \in W_\Sigma$ implies $u^{-1}L \in W_\Sigma$ for $u \in \mathbb{U}_\Sigma$) and diagonals.
2. A *variety of languages* is a family of local varieties $(\, W_\Sigma \subseteq \mathsf{Rec}(\Sigma)\,)_{\Sigma\in\mathbb{A}}$ closed under preimages, i.e. $L \in W_\Sigma$ implies $g^{-1}L \in W_\Delta$ for all $\Sigma, \Delta \in \mathbb{A}$ and $g\colon \mathbf{T}^{\geqslant}_{+} \to \mathbf{T}\not\leftarrow$ in $\mathbf{Alg}\,\mathbf{T}$.

▶ **Remark 6.7.**

1. Using the isomorphism $\mathsf{Rec}(\Sigma) \cong \prod_s P(\hat{T}\not\leftarrow)_s$ of Remark 3.4, one can show that a subobject $W_\Sigma \subseteq \mathsf{Rec}(\Sigma)$ is closed under diagonals iff it has the form $\prod_s m_s\colon \prod_s (W'_\Sigma)_s \rightarrowtail \prod_s P(\hat{T}\not\leftarrow)_s$ where $m_s\colon (W'_\Sigma)_s \rightarrowtail P(\hat{T}\not\leftarrow)_s$ is a monomorphism in $\mathcal{C}$.
2. There are two important cases where the closure under diagonals in Definition 6.6.1 is trivially satisfied and thus can be dropped. First, if $S = 1$, clearly every subobject of $\mathsf{Rec}(\Sigma)$ is closed under diagonals. Secondly, if $\mathcal{C}$ is one of the categories of Example 2.2 and $\mathbb{U}_\Sigma$ contains all identity morphisms, one can show that every subobject of $\mathsf{Rec}(\Sigma)$ closed under $\mathbb{U}_\Sigma$-derivatives is closed under diagonals. This will hold in all our applications.

We are ready to state the main result of our paper, which holds under the Assumptions 2.1.

▶ **Theorem 6.8** (Variety Theorem)**.**

1. *For each $\Sigma \in \mathbb{A}$, local varieties of languages over Σ and local pseudovarieties of Σ-generated $\mathbf{T}$-algebras form isomorphic lattices.*
2. *Varieties of languages and pseudovarieties of $\mathbf{T}$-algebras form isomorphic lattices. The isomorphism maps a pseudovariety $\mathcal{V}$ to the variety of all languages recognized by some algebra in $\mathcal{V}$.*

Proof sketch. Duality + Reiterman! For 1. one shows that a diagonal-closed subobject $W_\Sigma \subseteq \mathsf{Rec}(\Sigma)$, given by a monomorphism $(\, m_s\colon (W'_\Sigma)_s \rightarrowtail P(\hat{T}\not\leftarrow)_s \,)_{s\in S}$ in $\mathcal{C}^S$ by Rem. 6.7.1, is closed under derivatives iff the dual epimorphism $(\, P^{-1}m_s\colon (\hat{T}\not\leftarrow)_s \twoheadrightarrow P^{-1}(W'_\Sigma)_s \,)_{s\in S}$ in $\widehat{\mathcal{D}}^S$ carries a Σ-generated profinite $\widehat{\mathbf{T}}$-algebra. Then the Local Reiterman Theorem 5.3 gives the isomorphism. For 2. one shows that a family $(\, W_\Sigma \,)_{\Sigma\in\mathbb{A}}$ of local varieties is closed under preimages iff its dual family in $\widehat{\mathcal{D}}^S$ is a profinite theory, and uses the Reiterman Theorem. ◀

7 Applications

In this section, we derive some concrete variety theorems, including *new* results, as special instances of Theorem 6.8. In each case, we follow the three-step plan from the introduction.

Languages of finite words. Eilenberg's theorem [12] relates varieties of regular languages to pseudovarieties of monoids. It was later extended to ordered monoids [20], idempotent semirings [24] and K-algebras [26]. In [1, 3, 10] we unified all these results to an Eilenberg theorem for $\mathcal{D}$-monoids in a commutative variety $\mathcal{D}$, based on the dual view of automata as algebras and coalgebras. Recall that a variety $\mathcal{D}$ is *commutative* if for any $A, B \in \mathcal{D}$ the hom-set $\mathcal{D}(A, B)$ carries a subalgebra of $B^{|A|}$ in $\mathcal{D}$. A $\mathcal{D}$-*monoid* is an object $D \in \mathcal{D}$ with a monoid structure $(|D|, \bullet, 1)$ whose multiplication is a $\mathcal{D}$-*bimorphism*, i.e. for every $y \in |D|$ the maps $y \bullet -\colon |D| \to |D|$ and $- \bullet y\colon |D| \to |D|$ carry endomorphisms on D. Monoids in $\mathcal{D} = \mathbf{Set}$, $\mathbf{Pos}$, $\mathbf{JSL}_0$, $\mathbf{Vec}_K$ are classical monoids, ordered monoids, idempotent semirings and K-algebras, respectively. For any set Σ, the free $\mathcal{D}$-monoid $(\not\leftarrow^*, \bullet, \varepsilon)$ on $\not\leftarrow$ consists of the free $\mathcal{D}$-object $\not\leftarrow^*$ on Σ^*, the multiplication $\bullet$ extending the concatenation of words, and the empty word ε. The variety theorem for $\mathcal{D}$-monoids emerges from our three steps as follows:

1. Let $\mathbf{T}_M$ be the free-monoid monad on $\mathcal{D}$. Thus $T_M \not< = \not<^*$ and $\mathbf{Alg}\,\mathbf{T}_M$ is isomorphic to the category of $\mathcal{D}$-monoids. A language $L\colon \not<^* \to O_{\mathcal{D}}$ is $\mathbf{T}_M$-recognizable iff its restriction $L^{@}\colon \Sigma^* \to |O_{\mathcal{D}}|$ is a regular behavior, i.e. a function computed by some finite automaton with output set $|O_{\mathcal{D}}|$. If $|O_{\mathcal{D}}| \cong \{0,1\}$ (e.g., for $\mathcal{D} = \mathbf{Set}$, $\mathbf{Pos}$, $\mathbf{JSL}_0$), regular behaviors are exactly the characteristic functions of regular languages over Σ.
2. Generalizing Example 4.3.1, the free $\mathcal{D}$-monoid $\mathbf{T}_M \not< = \not<^*$ has the unary presentation $\mathbb{U}_\Sigma = \{\, \not<^* \xrightarrow{y\bullet -} \not<^*,\ \not<^* \xrightarrow{-\bullet y} \not<^* : y \in \Sigma^* \,\}$. Thus the $\mathbb{U}_\Sigma$-derivatives of a language $L\colon \not<^* \to O_{\mathcal{D}}$ are (after identifying L with $L^{@}$) the classical derivatives of Example 6.3.1.
3. Let $\mathbb{A} = \mathbf{Set}_f$. Instantiating Definition 6.6 gives the notion of a *variety of regular behaviors in* $\mathcal{C}$: a family $(W_\Sigma \subseteq \mathsf{Rec}(\Sigma))_{\Sigma\in\mathbb{A}}$ of regular behaviors closed under $\mathcal{C}$-algebraic operations (see Rem. 3.4), derivatives and preimages of $\mathcal{D}$-monoid morphisms, i.e. $g^{-1}L \in W_\Delta$ for any $L\colon \not<^* \to O_{\mathcal{D}}$ in W_Σ and any $\mathcal{D}$-monoid morphism $g\colon \gtrdot^* \to \not<^*$. Theorem 6.8 gives

▶ **Theorem 7.1** ([1,3,10]). *Let $\mathcal{C}$ and $\mathcal{D}$ be varieties satisfying the Assumptions 2.1(i),(ii),(iv), and suppose that the variety $\mathcal{D}$ is commutative. Then the lattice of (local) varieties of regular behaviors in $\mathcal{C}$ is isomorphic to the lattice of (local) pseudovarieties of $\mathcal{D}$-monoids.*

Four special instances are listed below. The third column describes the $\mathcal{C}$-algebraic operations under which (local) varieties are closed, and the fourth one states what $\mathcal{D}$-monoids are. All correspondences are known in the literature, and are uniformly covered by Theorem 7.1.

$\mathcal{C}$	$\mathcal{D}$	(local) var. of behav. closed under	$\cong$	(local) pseudovarieties of	proved in
$\mathbf{BA}$	$\mathbf{Set}$	boolean operations		monoids	[12, 14]
$\mathbf{DL}_{01}$	$\mathbf{Pos}$	finite union and finite intersection		ordered monoids	[14, 20]
$\mathbf{JSL}_0$	$\mathbf{JSL}_0$	finite union		idempotent semirings	[24]
$\mathbf{Vec}_K$	$\mathbf{Vec}_K$	addition of weighted languages		K-algebras	[26]

Polynomial varieties. Next, we derive Bojańczyk's polynomial variety theorem [8]. **1.** Let $\mathbf{T}$ be a monad on $\mathbf{Set}^S$. **2.** Choose the polynomial presentation of $\mathbf{T}\Sigma$. **3.** Let $\mathbb{A} = \mathbf{Set}_f^S$. Applying Def. 6.6, a *polynomial variety of languages* is a family of $\mathbf{T}$-recognizable languages closed under boolean operations, polynomial derivatives (see Ex. 6.2.3), and preimages of $\mathbf{T}$-homomorphisms. Thm. 6.8 gives Bojańczyk's variety theorem [8] and a new local version:

▶ **Theorem 7.2.** *The lattice of (local) polynomial varieties of $\mathbf{T}$-recognizable languages is isomorphic to the lattice of (local) pseudovarieties of $\mathbf{T}$-algebras.*

∞-languages. Finally, we derive two variety theorems for ∞-languages. For the first one, **1.** let $\mathbf{T} = \mathbf{T}_\infty$ on $\mathbf{Set}^2$. **2.** Choose $\mathbb{A} = \{\, (\Sigma, \emptyset) : \Sigma \in \mathbf{Set}_f \,\}$, and for each $\Sigma \in \mathbf{Set}_f$ the unary presentation of $\mathbf{T}_\infty(\Sigma,\emptyset) = (\Sigma^+, \Sigma^\omega)$ as in Example 4.3.2. **3.** Def. 6.6 yields the notion of a *variety of ∞-languages*: a family of ∞-regular languages closed under boolean operations, derivatives (see Example 6.3.2) and preimages of ω-semigroup morphisms. Theorem 6.8 gives

▶ **Theorem 7.3.** *The lattice of (local) varieties of ∞-languages is isomorphic to the lattice of (local) pseudovarieties of ω-semigroups.*

The non-local part is due to Wilke [19, 35] and the local part is a new result, extending the local variety theorem of [14] to infinite words. Similarly, we can obtain an ordered version of Theorem 7.3: **1.** take the monad $\mathbf{T}_{\infty,\leq}$ on $\mathbf{Pos}^2$ representing *ordered ω-semigroups* (i.e. ω-semigroups on a poset with monotone products). **2.** Choose $\mathbb{A}$ and the unary presentation

of $\mathbf{T}_{\infty,\leq}(\not<,\emptyset)$ as above. **3.** Since $\mathcal{C} = \mathbf{DL}_{01}$, Def. 6.6 gives *positive varieties of ∞-languages*, emerging from varieties by dropping closure under complement. Then Theorem 6.8 yields the theorem below. Its non-local part is due to Pin [21], and the local part is again a new result.

▶ **Theorem 7.4.** *The lattice of (local) positive varieties of ∞-languages is isomorphic to the lattice of (local) pseudovarieties of ordered ω-semigroups.*

The two above theorems do not follow from Theorem 7.1/7.2, which shows that our framework has a wider scope than the earlier work in [1, 3, 8, 10]. For many more applications, including variety theorems for tree languages [29] and cost functions [11], see the full paper [34].

8 Conclusions and Future Work

We presented a categorical framework for algebraic language theory that captures, as special instances, the bulk of the Eilenberg theorems in the literature for pseudovarieties of finite algebras and varieties of recognizable languages. Let us mention directions for future work.

First, we aim to investigate if it is possible to obtain a variety theorem for data languages based on nominal Stone duality [13]. On a similar note, it would also be interesting to see whether dualities modeling probabilistic phenomena (e.g. Gelfand or Kadison duality) lead to a meaningful algebraic language theory for probabilistic automata and languages.

Secondly, although *finite* structures are of most relevance from the automata-theoretic perspective, there has been some work on variety theorems with relaxed finiteness restrictions. One example is Reutenauer's theorem [26] for weighted languages over arbitrary fields K. To cover this in our setting the results of Section 5 should be presented for $(\mathcal{E},\mathcal{M})$-structured categories $\mathcal{D}$ in lieu of varieties. This has been worked out in [33] and, independently, in the recent preprint [27]. In the latter, also a formal "Eilenberg correspondence" is stated for dual $(\mathcal{E},\mathcal{M})$-categories. An important conceptual difference to our present work is that in loc. cit. one uses discrete dualities (e.g. complete atomic boolean algebras/sets instead of boolean algebras/Stone spaces) and that unary presentations do not appear. This makes the concept of a variety of languages (called a *coequational theory*) and the Eilenberg correspondence in [27] easy to state, but much harder to apply in practice. The results of [27] and of our paper do not entail each other, and we leave it for future work to find a common roof.

References

1 J. Adámek, S. Milius, R. Myers, and H. Urbat. Generalized Eilenberg Theorem I: Local Varieties of Languages. In A. Muscholl, editor, *Proc. FoSSaCS'14*, volume 8412 of *LNCS*, pages 366–380. Springer, 2014. Full version: `http://arxiv.org/pdf/1501.02834v1.pdf`.

2 J. Adámek, S. Milius, and H. Urbat. Syntactic monoids in a category. In *Proc. CALCO'15*, LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015. Full version: `http://arxiv.org/abs/1504.02694`.

3 J. Adámek, R. Myers, S. Milius, and H. Urbat. Varieties of languages in a category. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE, 2015.

4 J. Almeida. On pseudovarieties, varieties of languages, filters of congruences, pseudoidentities and related topics. *Algebra Universalis*, 27(3):333–350, 1990.

5 N. Bedon and O. Carton. An Eilenberg theorem for words on countable ordinals. In *Proc. LATIN'98*, volume 1380 of *LNCS*, pages 53–64. Springer, 1998.

6 N. Bedon and C. Rispal. Schützenberger and Eilenberg theorems for words on linear orderings. In *Proc. DLT'05*, volume 3572 of *LNCS*, pages 134–145. Springer, 2005.

7 G. Birkhoff. Rings of sets. *Duke Mathematical Journal*, 3(3):443–454, 1937.

8 M. Bojańczyk. Recognisable languages over monads. In I. Potapov, editor, *Proc. DLT'15*, volume 9168 of *LNCS*, pages 1–13. Springer, 2015. http://arxiv.org/abs/1502.04898.

9 L.-T. Chen, J. Adámek, S. Milius, and H. Urbat. Profinite monads, profinite equations and Reiterman's theorem. In B. Jacobs and C. Löding, editors, *Proc. FoSSaCS'16*, volume 9634 of *LNCS*. Springer, 2016. http://arxiv.org/abs/1511.02147.

10 L.-T. Chen and H. Urbat. A fibrational approach to automata theory. In L. S. Moss and P. Sobocinski, editors, *Proc. CALCO'15*, LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015.

11 L. Daviaud, D. Kuperberg, and J.-É. Pin. Varieties of cost functions. In N. Ollinger and H. Vollmer, editors, *Proc. STACS 2016*, volume 47 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

12 S. Eilenberg. *Automata, Languages, and Machines Vol. B.* Academic Press, 1976.

13 M. J. Gabbay, T. Litak, and D. Petrişan. Stone duality for nominal boolean algebras with new. In A. Corradini, B. Klin, and C. Cîrstea, editors, *Proc. CALCO'11*, volume 6859 of *LNCS*, pages 192–207. Springer, 2011.

14 M. Gehrke, S. Grigorieff, and J.-É. Pin. Duality and equational theory of regular languages. In L. Aceto and al., editors, *Proc. ICALP'08, Part II*, volume 5126 of *LNCS*, pages 246–257. Springer, 2008.

15 P. T. Johnstone. *Stone spaces.* Cambridge University Press, 1982.

16 E. G. Manes. *Algebraic Theories*, volume 26 of *Graduate Texts in Mathematics.* Springer, 1976.

17 G. Matthiessen. Theorie der heterogenen Algebren. Technical report, Universität Bremen, 1976.

18 G. Matthiessen. A heterogeneous algebraic approach to some problems in automata theory, many-valued logic and other topics. In *Proc. Klagenfurt Conf.*, 1979.

19 D. Perrin and J.-É. Pin. *Infinite Words.* Elsevier, 2004.

20 J.-É. Pin. A variety theorem without complementation. *Russ. Math.*, 39:80–90, 1995.

21 J.-É. Pin. Positive varieties and infinite words. In *LATIN 98*, volume 1380 of *LNCS*, pages 76–87. Springer, 1998.

22 J.-É. Pin. Mathematical foundations of automata theory. Available at http://www.liafa.jussieu.fr/~jep/PDF/MPRI/MPRI.pdf, November 2016.

23 N. Pippenger. Regular languages and Stone duality. *Th. Comp. Sys.*, 30(2):121–134, 1997. URL: http://link.springer.com/10.1007/BF02679444, doi:10.1007/BF02679444.

24 L. Polák. Syntactic semiring of a language. In J. Sgall, A. Pultr, and P. Kolman, editors, *Proc. MFCS'01*, volume 2136 of *LNCS*, pages 611–620. Springer, 2001.

25 J. Reiterman. The Birkhoff theorem for finite algebras. *Algebra Universalis*, 14(1):1–10, 1982.

26 C. Reutenauer. Séries formelles et algèbres syntactiques. *J. Algebra*, 66:448–483, 1980.

27 J. Salamánca. Unveiling Eilenberg-type Correspondences: Birkhoff's Theorem for (finite) Algebras + Duality. https://arxiv.org/abs/1702.02822, February 2017.

28 S. Salehi and M. Steinby. Varieties of many-sorted recognizable sets. TUCS Technical Report 626, Turku Center for Computer Science, 2004.

29 S. Salehi and M. Steinby. Tree algebras and varieties of tree languages. *Theor. Comput. Sci.*, 377(1-3):1–24, 2007.

30 M. P. Schützenberger. On finite monoids having only trivial subgroups. *Inform. and Control*, 8:190–194, 1965.

31 H. Straubing. On logical descriptions of regular languages. In S. Rajsbaum, editor, *LATIN 2002 Theor. Informatics*, volume 2286 of *LNCS*, pages 528–538. Springer, 2002.

32 D. Thérien. *Classification of Regular Languages by Congruences.* PhD thesis, University of Waterloo, 1980.

33 H. Urbat. A note on HSP theorems. `https://www.tu-braunschweig.de/Medien-DB/iti/hspnote.pdf`.

34 H. Urbat, J. Adámek, L.-T. Chen, and S. Milius. Eilenberg Theorems for Free. `https://arxiv.org/abs/1602.05831`. February 2017.

35 T. Wilke. An Eilenberg theorem for ∞-languages. In *Proc. ICALP'91*, volume 510 of *LNCS*, pages 588–599. Springer, 1991.

Membership Problem in $\mathrm{GL}(2,\mathbb{Z})$ Extended by Singular Matrices*

Igor Potapov[1] and Pavel Semukhin[2]

1 **Department of Computer Science, University of Liverpool, United Kingdom**
potapov@liverpool.ac.uk
2 **Department of Computer Science, University of Liverpool, United Kingdom**
semukhin@liverpool.ac.uk

Abstract

We consider the membership problem for matrix semigroups, which is the problem to decide whether a matrix belongs to a given finitely generated matrix semigroup.

In general, the decidability and complexity of this problem for two-dimensional matrix semigroups remains open. Recently there was a significant progress with this open problem by showing that the membership is decidable for 2×2 nonsingular integer matrices. In this paper we focus on the membership for singular integer matrices and prove that this problem is decidable for 2×2 integer matrices whose determinants are equal to 0, 1, -1 (i.e. for matrices from $\mathrm{GL}(2,\mathbb{Z})$ and any singular matrices). Our algorithm relies on a translation of numerical problems on matrices into combinatorial problems on words and conversion of the membership problem into decision problem on regular languages.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems, F.1.1 Models of Computation

Keywords and phrases Matrix Semigroups, Membership Problem, General Linear Group, Singular Matrices, Automata and Formal Languages

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.44

1 Introduction

Matrices and matrix products play a crucial role in the representation and analysis of various computational processes such as linear recurrent sequences [13, 21, 22], arithmetic circuits [11], hybrid and dynamical systems [20, 2], probabilistic and quantum automata [7], stochastic games, broadcast protocols [10]. Many problems for matrices in dimension three and four are undecidable, but the decidability and complexity of problems for two-dimensional matrix semigroups remains open. One of such hard questions is the Membership problem.

Membership problem: Given a finite set of $m \times m$ matrices $\mathcal{F} = \{M_1, M_2, \ldots, M_n\}$ and a matrix M. Determine whether there exist an integer $k \geq 1$ and $i_1, i_2, \ldots, i_k \in \{1, \ldots, n\}$ such that

$$M_{i_1} \cdot M_{i_2} \cdots M_{i_k} = M.$$

In other words, determine whether a matrix M belongs to the semigroup generated by $\mathcal{F}$.

* This work was supported by EPSRC grant "Reachability problems for words, matrices and maps" (EP/M00077X/1).

There are only few known decidability results for the membership problem when the dimension is not bounded. In 1986 Kannan and Lipton [14] proved that the membership is decidable in polynomial time for a semigroup generated by a single $m \times m$ matrix (known as the *Orbit problem*). Later, in 1996 this decidability result was extended to a more general case of commutative matrices [1]. A generalization of this result to a special class of non-commutative matrices (a class of row-monomial matrices over a commutative semigroup satisfying some natural effectiveness conditions) was shown in 2004 in [16]. On the other hand, it is known that the membership is already undecidable for 3×3 integer matrices with determinant 0 (i.e. singular matrices), see [23]. As for the nonsingular case, it is known that the membership is undecidable for 4×4 integer matrices with determinant one, see [5]. A more recent survey of undecidable problems can be found in [8].

Due to a severe lack of methods and techniques the status of decision problems for 2×2 matrices (like membership, vector reachability, freeness) remains a long standing open problem. Recently, a new approach of translating numerical problems on 2×2 integer matrices into a variety of combinatorial and computational problems on words over group alphabet and studying their transformations as specific rewriting systems have led to new results on decidability and complexity. The main ingredient of the translation into combinatorial problems on words is the well-known result that the groups $\mathrm{SL}(2,\mathbb{Z})$ and $\mathrm{GL}(2,\mathbb{Z})$ are finitely generated. For example, $\mathrm{SL}(2,\mathbb{Z})$ can be generated by a pair of matrices

$$S = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix}$$

with the following relations: $S^4 = I$, $R^6 = I$ and $S^2 = R^3$. So, we can represent a matrix $M \in \mathrm{SL}(2,\mathbb{Z})$ as a word in the alphabet $\{S, R\}$.

In particular, this symbolic representation was successfully used to show the decidability of the membership problem for the semigroups of $\mathrm{GL}(2,\mathbb{Z})$ [9] in 2005 and of the mortality problem for 2×2 integer matrices with determinants $0, \pm1$ [19] in 2008. It also found applications in the design of the polynomial time algorithm for the membership problem for the modular group [12] in 2007. Furthermore, it was used to show NP-hardness for most of the reachability problems in dimension two [6, 3] in 2012 and to prove decidability of the vector/scalar reachability problems in $\mathrm{SL}(2,\mathbb{Z})$ [24] in 2016.

In 2017 a significant progress was made towards decidability of the membership problem for 2×2 integer matrices extending previously known result on $\mathrm{GL}(2,\mathbb{Z})$ [9]. In [25] the first algorithm was discovered that can check the membership problem for a matrix semigroup generated by nonsingular 2×2 integer matrices. In this paper we show another extension of [9] and prove that the membership problem is decidable for 2×2 integer matrices whose determinants are equal to 0, 1, -1 (i.e. for matrices from $\mathrm{GL}(2,\mathbb{Z})$ and any singular matrices). As a first step we give an alternative proof of the decidability of the mortality problem (i.e. membership for the zero matrix) from [19], in which we will use the Smith normal forms of matrices. In contrast to [19], our new approach allows us to generalize this proof to show decidability of the membership problem for singular matrices. The algorithm is based on a nontrivial combination of algebraic properties of $\mathrm{GL}(2,\mathbb{Z})$, automata theory and properties of matrix products with singular matrices.

2 Preliminaries

The semigroup of 2×2 integer matrices is denoted by $\mathbb{Z}^{2\times2}$. Let $\mathrm{GL}(2,\mathbb{Z})$ be the general linear group of dimension 2 over $\mathbb{Z}$, that is, the group of 2×2 integer matrices whose determinant is equal to ±1. We will use $\mathbf{O}$ to denote the zero 2×2 matrix. A matrix is called *singular* if its determinant is equal to zero and *nonsingular* otherwise.

If $\mathcal{F}$ is a finite collection of matrices from $\mathbb{Z}^{2\times 2}$, then $\langle\mathcal{F}\rangle$ denotes the semigroup generated by $\mathcal{F}$ (including the identity matrix), that is, $M \in \langle\mathcal{F}\rangle$ if and only if $M = I$ or there are matrices $M_1, \ldots, M_n \in \mathcal{F}$ such that $M = M_1 \cdots M_n$.

Consider the following matrices from $\mathrm{GL}(2,\mathbb{Z})$:

$$N = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},\ S = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix},\ R = \begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix},\ X = -I = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Every word in the alphabet $\Sigma = \{N, S, R, X\}$ corresponds to a matrix from $\mathrm{GL}(2,\mathbb{Z})$ in a natural way. Namely, the letters N, S, R, X correspond to the matrices defined by the above formulas, and a word $w \in \Sigma^*$ corresponds to the product of its letters. For example, the word SR corresponds to the matrix $\begin{bmatrix} -1 & -1 \\ 0 & -1 \end{bmatrix}$.

It is well-known that $\mathrm{GL}(2,\mathbb{Z})$ is generated by the above matrices. So any $M \in \mathrm{GL}(2,\mathbb{Z})$ can be presented by a word in the alphabet $\Sigma = \{N, S, R, X\}$. Such presentation is not unique because of the identities like $S^2 = R^3 = X$. However for every matrix $M \in \mathrm{GL}(2,\mathbb{Z})$, there is a unique canonical word that represents it, as described below.

▶ **Definition 1.** A word $w \in \Sigma^*$ is called a *canonical word* if it has the form

$$w = N^\delta X^\gamma S^\beta R^{\alpha_0} S R^{\alpha_1} S R^{\alpha_2} \ldots S R^{\alpha_{n-1}} S R^{\alpha_n},$$

where $\beta, \delta, \gamma \in \{0, 1\}$, $\alpha_0, \ldots, \alpha_{n-1} \in \{1, 2\}$, and $\alpha_n \in \{0, 1, 2\}$. In other words, w is *canonical* if it does not contain subwords SS or RRR. Moreover, letter N may appear only once in the first position, and letter X may appear only once either in the first position or after N.

▶ **Proposition 2** ([17, 18, 26, 25])**.** *For every $M \in \mathrm{GL}(2,\mathbb{Z})$, there is a unique canonical word w which represents M.*

We will also use two additional matrices $T = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, $U = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ and the following identities: $R = ST$, $T = XSR$, $T^{-1} = XR^2S$, $U = XSR^2$ and $U^{-1} = XRS$.

▶ **Definition 3.** A subset $\mathcal{S} \subseteq \mathrm{GL}(2,\mathbb{Z})$ is called *regular* or *automatic* if there is a regular language L in alphabet Σ that describes $\mathcal{S}$. That is, every word $w \in L$ corresponds to a matrix M from $\mathcal{S}$, and for every matrix $M \in \mathcal{S}$, there is a word $w \in L$ that represents M.

▶ **Definition 4.** We call two words w_1 and w_2 from Σ^* *equivalent*, denoted $w_1 \sim w_2$, if they represent the same matrix. Two languages L_1 and L_2 in the alphabet Σ are *equivalent*, denoted $L_1 \sim L_2$, if
(i) for each $w_1 \in L_1$, there exists $w_2 \in L_2$ such that $w_1 \sim w_2$, and
(ii) for each $w_2 \in L_2$, there exists $w_1 \in L_1$ such that $w_2 \sim w_1$.
In other words, L_1 and L_2 are equivalent if and only if they describe the same language. Two finite automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over alphabet Σ are *equivalent*, denoted $\mathcal{A}_1 \sim \mathcal{A}_2$, if $L(\mathcal{A}_1) \sim L(\mathcal{A}_2)$.

The next theorem will be a crucial ingredient of our decidability result.

▶ **Theorem 5.** *Given two regular subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ of $\mathrm{GL}(2,\mathbb{Z})$, it is decidable whether the intersection $\mathcal{S}_1 \cap \mathcal{S}_2$ is empty or not.*

Proof. The proof is based on the following result: for every finite automaton $\mathcal{A}$ over alphabet Σ, there is an automaton $\mathrm{Can}(\mathcal{A})$ such that $\mathrm{Can}(\mathcal{A})$ accepts only canonical words and $\mathrm{Can}(\mathcal{A}) \sim \mathcal{A}$, that is, $\mathrm{Can}(\mathcal{A})$ and $\mathcal{A}$ describe the same subset of $\mathrm{GL}(2,\mathbb{Z})$. The construction of $\mathrm{Can}(\mathcal{A})$ can be found in [25], see also [9] for an alternative construction.

Now let L_1 and L_2 be regular languages that describe $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively, and let $\mathcal{A}_1$ and $\mathcal{A}_2$ be finite automata such that $L(\mathcal{A}_1) = L_1$ and $L(\mathcal{A}_2) = L_2$. By Proposition 2, every matrix from $\mathrm{GL}(2,\mathbb{Z})$ corresponds to a unique canonical word. Therefore, we obtain the following equivalence: $\mathcal{S}_1 \cap \mathcal{S}_2 \neq \emptyset$ if and only if the languages of the automata $\mathrm{Can}(\mathcal{A}_1)$ and $\mathrm{Can}(\mathcal{A}_2)$ have nonempty intersection. Since the emptiness problem for regular languages is decidable, it is decidable whether $\mathcal{S}_1 \cap \mathcal{S}_2$ is empty or not. ◀

Another important ingredient of our proof is the existence and uniqueness of the Smith normal form of a matrix.

▶ **Theorem 6** (Smith normal form [15]). *For any matrix $A \in \mathbb{Z}^{2\times 2}$, there are matrices E, F from $\mathrm{GL}(2,\mathbb{Z})$ such that $A = E \begin{bmatrix} t_1 & 0 \\ 0 & t_2 \end{bmatrix} F$ for some nonnegative integers t_1 and t_2 such that $t_1 \mid t_2$. The diagonal matrix $\begin{bmatrix} t_1 & 0 \\ 0 & t_2 \end{bmatrix}$, which is unique, is called the* Smith normal form *of A. In fact, t_1 is equal to the gcd of the coefficients of A. Moreover, E, F, t_1, and t_2 can be computed in polynomial time.*

Remark. If $A \in \mathbb{Z}^{2\times 2}$ is nonzero matrix with $\det(A) = 0$, then the Smith normal form of A is equal to $\begin{bmatrix} t & 0 \\ 0 & 0 \end{bmatrix}$, where t is the gcd of the coefficients of A.

Using uniqueness of the Smith normal form we obtain the following corollary.

▶ **Corollary 7.** *If $E, F \in \mathrm{GL}(2,\mathbb{Z})$, then $E \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} F$ is not a zero matrix.*

3 Main result

The main result of our paper is the following theorem.

▶ **Theorem 8.** *Let $M \in \mathbb{Z}^{2\times 2}$ and let $\mathcal{F} = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$ be a collection of matrices from $\mathbb{Z}^{2\times 2}$ such that $A_i \in \mathrm{GL}(2,\mathbb{Z})$ for $i = 1, \ldots, n$, and B_j is a singular matrix for $j = 1, \ldots, m$. Then it is decidable whether $M \in \langle \mathcal{F} \rangle$.*

Proof. First, note that if $M \in \langle \mathcal{F} \rangle$, then M is either singular or $M \in \mathrm{GL}(2,\mathbb{Z})$. Therefore, if $|\det(M)| > 1$, then we know that $M \notin \langle \mathcal{F} \rangle$. On the other hand, if $\det(M) = \pm 1$, i.e. if $M \in \mathrm{GL}(2,\mathbb{Z})$, then $M \in \langle \mathcal{F} \rangle$ if and only if $M \in \langle A_1, \ldots, A_n \rangle$. In other words, our problem reduces to the membership problem in $\mathrm{GL}(2,\mathbb{Z})$, and the decidability of the membership in $\mathrm{GL}(2,\mathbb{Z})$ was proven in [9].

Hence from now on we will assume that M is a singular matrix. First, we consider the case when M is the zero matrix and after that we consider the case when M is a nonzero singular matrix. The case when $M = \mathbf{O}$ is also called the *Mortality problem*. The decidability of the mortality problem for matrices with determinants 0 and ± 1 was shown in [19]. In Theorem 11 below we provide an alternative proof of this fact, which is based on the use of the Smith normal form of a matrix. Another reason why we include the proof of Theorem 11 is that it presents a simplified version of a more complicated construction for a nonzero M. Finally, in Theorem 13 we prove decidability of the membership problem for a nonzero singular matrix M. ◀

First, we prove Proposition 9 which will play an important role in the proofs of Theorems 11 and 13. Moreover, Proposition 9 and Corollary 10 reveal new structural properties of certain subsets of $\mathrm{GL}(2,\mathbb{Z})$ in symbolic presentation.

▶ **Proposition 9.** *For any fixed $a \in \mathbb{Z}$, let $M(a) = \left\{ \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathrm{GL}(2,\mathbb{Z}) \ : \ b,c,d \in \mathbb{Z} \right\}$. Then $M(a)$ is a regular subset of $\mathrm{GL}(2,\mathbb{Z})$.*

Proof. First, suppose that $a = 0$. Then

$$M(0) = \left\{ \begin{bmatrix} 0 & -1 \\ 1 & d \end{bmatrix} : d \in \mathbb{Z} \right\} \bigcup \left\{ \begin{bmatrix} 0 & 1 \\ -1 & d \end{bmatrix} : d \in \mathbb{Z} \right\} \bigcup \left\{ \begin{bmatrix} 0 & 1 \\ 1 & d \end{bmatrix} : d \in \mathbb{Z} \right\} \bigcup \left\{ \begin{bmatrix} 0 & -1 \\ -1 & d \end{bmatrix} : d \in \mathbb{Z} \right\}.$$

Note that $\begin{bmatrix} 0 & -1 \\ 1 & d \end{bmatrix} = ST^d$, $\begin{bmatrix} 0 & 1 \\ -1 & d \end{bmatrix} = -ST^{-d}$, $\begin{bmatrix} 0 & 1 \\ 1 & d \end{bmatrix} = SNT^d$, $\begin{bmatrix} 0 & -1 \\ -1 & d \end{bmatrix} = -SNT^{-d}$. Hence we can express $M(0)$ as

$$\begin{aligned} M(0) =& \{ST^d : d \in \mathbb{Z}\} \cup \{-ST^{-d} : d \in \mathbb{Z}\} \cup \{SNT^d : d \in \mathbb{Z}\} \cup \{-SNT^{-d} : d \in \mathbb{Z}\} = \\ & \{ST^d : d \geq 0\} \cup \{S(T^{-1})^d : d \geq 0\} \cup \{-S(T^{-1})^d : d \geq 0\} \cup \\ & \{-ST^d : d \geq 0\} \cup \{SNT^d : d \geq 0\} \cup \{SN(T^{-1})^d : d \geq 0\} \cup \\ & \{-SN(T^{-1})^d : d \geq 0\} \cup \{-SNT^d : d \geq 0\}. \end{aligned}$$

Therefore, $M(0)$ can be described by the following regular expression

$$\begin{aligned} & S(XSR)^* + S(XR^2S)^* + XS(XR^2S)^* + XS(XSR)^* + \\ & SN(XSR)^* + SN(XR^2S)^* + XSN(XR^2S)^* + XSN(XSR)^*. \end{aligned}$$

Now suppose that $a \neq 0$. Consider a matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathrm{GL}(2,\mathbb{Z})$ and let $b = b_0 + ma$ and $c = c_0 + na$, where $b_0, c_0 \in \{0, \ldots, |a| - 1\}$ and $m, n \in \mathbb{Z}$. Since $A \in \mathrm{GL}(2,\mathbb{Z})$, we have $ad - bc = \pm 1$ or

$$d = \frac{bc \pm 1}{a} = \frac{(b_0 + ma)(c_0 + na) \pm 1}{a} = \frac{b_0c_0 \pm 1}{a} + mc_0 + nb_0 + mna.$$

Since d is an integer, $\frac{b_0c_0 \pm 1}{a}$ must also be an integer. Note that

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & b_0 + ma \\ c_0 + na & \frac{b_0c_0 \pm 1}{a} + mc_0 + nb_0 + mna \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ n & 1 \end{bmatrix} \begin{bmatrix} a & b_0 \\ c_0 & \frac{b_0c_0 \pm 1}{a} \end{bmatrix} \begin{bmatrix} 1 & m \\ 0 & 1 \end{bmatrix}.$$

So, $A = U^n \begin{bmatrix} a & b_0 \\ c_0 & \frac{b_0c_0 \pm 1}{a} \end{bmatrix} T^m$. Let $N^+(a)$ and $N^-(a)$ be the following finite sets

$$\begin{aligned} N^+(a) &= \{(b_0, c_0) \ : \ b_0, c_0 \in \{0, \ldots, |a| - 1\} \text{ and } \tfrac{b_0c_0 + 1}{a} \text{ is an integer}\}, \\ N^-(a) &= \{(b_0, c_0) \ : \ b_0, c_0 \in \{0, \ldots, |a| - 1\} \text{ and } \tfrac{b_0c_0 - 1}{a} \text{ is an integer}\}. \end{aligned}$$

Then

$$M(a) = \bigcup_{(b_0,c_0) \in N^+(a)} \left\{ U^n \begin{bmatrix} a & b_0 \\ c_0 & \frac{b_0c_0+1}{a} \end{bmatrix} T^m \ : \ n,m \in \mathbb{Z} \right\} \bigcup \bigcup_{(b_0,c_0) \in N^-(a)} \left\{ U^n \begin{bmatrix} a & b_0 \\ c_0 & \frac{b_0c_0-1}{a} \end{bmatrix} T^m \ : \ n,m \in \mathbb{Z} \right\}.$$

For each $(b_0, c_0) \in N^+(a)$, let $w^+(b_0, c_0)$ be a word that represents the matrix $\begin{bmatrix} a & b_0 \\ c_0 & \frac{b_0c_0+1}{a} \end{bmatrix}$. Note that for every $(b_0, c_0) \in N^+(a)$, we can present $\left\{ U^n \begin{bmatrix} a & b_0 \\ c_0 & \frac{b_0c_0+1}{a} \end{bmatrix} T^m : n, m \in \mathbb{Z} \right\}$ as a union of four sets

$$\left\{ U^n \begin{bmatrix} a & b_0 \\ c_0 & \frac{b_0c_0+1}{a} \end{bmatrix} T^m : \ n, m \geq 0 \right\} \bigcup \left\{ (U^{-1})^n \begin{bmatrix} a & b_0 \\ c_0 & \frac{b_0c_0+1}{a} \end{bmatrix} T^m : \ n, m \geq 0 \right\} \bigcup$$
$$\left\{ U^n \begin{bmatrix} a & b_0 \\ c_0 & \frac{b_0c_0+1}{a} \end{bmatrix} (T^{-1})^m : \ n, m \geq 0 \right\} \bigcup \left\{ (U^{-1})^n \begin{bmatrix} a & b_0 \\ c_0 & \frac{b_0c_0+1}{a} \end{bmatrix} (T^{-1})^m : \ n, m \geq 0 \right\}.$$

Hence it can be described by the following regular expression

$$(XSR^2)^* w^+(b_0, c_0)(XSR)^* + (XRS)^* w^+(b_0, c_0)(XSR)^* +$$
$$(XSR^2)^* w^+(b_0, c_0)(XR^2S)^* + (XRS)^* w^+(b_0, c_0)(XR^2S)^*.$$

Similarly, we have that for every $(b_0, c_0) \in N^-(a)$, the set $\left\{ U^n \begin{bmatrix} a & b_0 \\ c_0 & \frac{b_0c_0-1}{a} \end{bmatrix} T^m : n, m \in \mathbb{Z} \right\}$ can be described by a regular expression. Since $M(a)$ is equal to a finite union of such sets, we conclude that it is also regular. ◀

▶ **Corollary 10.** *For every $i = 1, 2, 3, 4$ and any fixed $a \in \mathbb{Z}$, the following subset of $\mathrm{GL}(2,\mathbb{Z})$ is a regular set:* $M_i(a) = \left\{ \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \in \mathrm{GL}(2,\mathbb{Z}) \ : \ a_i = a \text{ and } a_j \in \mathbb{Z} \text{ for } j \neq i \right\}$.

Proof. By definition, $M_1(a) = M(a)$, hence it is regular by Proposition 9. Now let $L(a)$ be a regular language that describes $M(a)$ and let $K = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. It is not hard to see that $M_2(a) = M(a) \cdot K$, $M_3(a) = K \cdot M(a)$ and $M_4(a) = K \cdot M(a) \cdot K$. Note that matrix K corresponds to the word NXS. Therefore, $M_2(a)$, $M_3(a)$ and $M_4(a)$ can be described by the regular languages $L(a) \cdot \{NXS\}$, $\{NXS\} \cdot L(a)$ and $\{NXS\} \cdot L(a) \cdot \{NXS\}$, respectively. ◀

3.1 Mortality problem

In this section we will give an alternative proof of the decidability of the mortality problem from [19] which will be based on the use of the Smith normal form of a matrix.

▶ **Theorem 11.** *The mortality problem for 2×2 integer matrices with determinants $0, \pm 1$ is decidable.*

This theorem will be a consequence of Theorem 5 and Propositions 9 and 12.

▶ **Proposition 12.** *Let $\mathcal{F} = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$ be a collection of matrices from $\mathbb{Z}^{2\times 2}$ such that $A_i \in \mathrm{GL}(2,\mathbb{Z})$ for $i = 1, \ldots, n$, and $\det(B_j) = 0$ for $j = 1, \ldots, m$. Then $\mathbf{O} \in \langle \mathcal{F} \rangle$ if and only if $B_j = \mathbf{O}$ for some j or there are indices $i, j \in \{1, \ldots, m\}$ and a matrix $C \in \langle A_1, \ldots, A_n \rangle$ such that $B_i C B_j = \mathbf{O}$.*

Proof. Suppose that $B_j \neq \mathbf{O}$ for every j. Under this assumption, if $\mathbf{O} \in \langle \mathcal{F} \rangle$ then there are indices $i_1, \ldots, i_s \in \{1, \ldots, m\}$ and matrices $C_1, \ldots, C_{s+1} \in \langle A_1, \ldots, A_n \rangle$ such that

$$C_1 B_{i_1} C_2 B_{i_2} \cdots C_s B_{i_s} C_{s+1} = \mathbf{O}. \tag{1}$$

By Theorem 6, we can write each matrix B_{i_r}, for $i = 1, \ldots, s$, as $B_{i_r} = E_r \begin{bmatrix} t_r & 0 \\ 0 & 0 \end{bmatrix} F_r$, where $E_r, F_r \in \mathrm{GL}(2, \mathbb{Z})$ and $t_r > 0$. Then (1) is equivalent to

$$C_1 E_1 \begin{bmatrix} t_1 & 0 \\ 0 & 0 \end{bmatrix} F_1 C_2 E_2 \begin{bmatrix} t_2 & 0 \\ 0 & 0 \end{bmatrix} F_2 \cdots \\ \cdots C_{s-1} E_{s-1} \begin{bmatrix} t_{s-1} & 0 \\ 0 & 0 \end{bmatrix} F_{s-1} C_s E_s \begin{bmatrix} t_s & 0 \\ 0 & 0 \end{bmatrix} F_s C_{s+1} = \mathbf{O}. \quad (2)$$

Dividing (2) by the product $t_1 t_2 \cdots t_{s-1} t_s$, which is nonzero by our assumption, we obtain

$$C_1 E_1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} F_1 C_2 E_2 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} F_2 \cdots C_{s-1} E_{s-1} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} F_{s-1} C_s E_s \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} F_s C_{s+1} = \mathbf{O}. \quad (3)$$

Suppose for each $r = 2, \ldots, s$ the matrix $F_{r-1} C_r E_r$ have the form $F_{r-1} C_r E_r = \begin{bmatrix} a_r & b_r \\ c_r & d_r \end{bmatrix}$. Note that $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & 0 \end{bmatrix}$. Therefore, (3) is equivalent to

$$C_1 E_1 \begin{bmatrix} a_2 a_3 \cdots a_s & 0 \\ 0 & 0 \end{bmatrix} F_s C_{s+1} = \mathbf{O}. \quad (4)$$

Suppose that $a_2 a_3 \cdots a_s \neq 0$. In this case (4) is equivalent to

$$C_1 E_1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} F_s C_{s+1} = \mathbf{O},$$

where $C_1 E_1$ and $F_s C_{s+1}$ are matrices from $\mathrm{GL}(2, \mathbb{Z})$. This contradicts Corollary 7. Hence $a_2 a_3 \cdots a_s = 0$, and therefore there is $r \in \{2, \ldots, s\}$ such that $a_r = 0$. This implies that $B_{i_{r-1}} C_r B_{i_r} = \mathbf{O}$. Indeed,

$$B_{i_{r-1}} C_r B_{i_r} = E_{r-1} \begin{bmatrix} t_{r-1} & 0 \\ 0 & 0 \end{bmatrix} F_{r-1} C_r E_r \begin{bmatrix} t_r & 0 \\ 0 & 0 \end{bmatrix} F_r.$$

By assumption, $F_{r-1} C_r E_r = \begin{bmatrix} a_r & b_r \\ c_r & d_r \end{bmatrix}$. Thus

$$B_{i_{r-1}} C_r B_{i_r} = t_{r-1} t_r E_{r-1} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a_r & b_r \\ c_r & d_r \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} F_r = t_{r-1} t_r E_{r-1} \begin{bmatrix} a_r & 0 \\ 0 & 0 \end{bmatrix} F_r.$$

Since $a_r = 0$, we have $B_{i_{r-1}} C_r B_{i_r} = \mathbf{O}$.

The implication on the other direction is trivial. ◀

Proof of Theorem 11. Obviously, if $B_j = \mathbf{O}$ for some $j = 1, \ldots, m$, then $\mathbf{O} \in \langle \mathcal{F} \rangle$. Therefore, from now on we assume that all B_j's are nonzero singular matrices. In this case Proposition 12 implies that $\mathbf{O} \in \langle \mathcal{F} \rangle$ if and only if there are indices $i, j \in \{1, \ldots, m\}$ and a matrix $C \in \langle A_1, \ldots, A_n \rangle$ such that $B_i C B_j = \mathbf{O}$. We now show that the latter property is algorithmically decidable.

Let $B_i = E_i \begin{bmatrix} t_i & 0 \\ 0 & 0 \end{bmatrix} F_i$ and $B_j = E_j \begin{bmatrix} t_j & 0 \\ 0 & 0 \end{bmatrix} F_j$ be the Smith normal forms of B_i and B_j, respectively. Note that by our assumption $t_i, t_j > 0$. Let C be a matrix from $\langle A_1, \ldots, A_n \rangle$. We have $B_i C B_j = \mathbf{O}$ if and only if

$$E_i \begin{bmatrix} t_i & 0 \\ 0 & 0 \end{bmatrix} F_i C E_j \begin{bmatrix} t_j & 0 \\ 0 & 0 \end{bmatrix} F_j = \mathbf{O} \quad \text{or, equivalently,} \quad E_i \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} F_i C E_j \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} F_j = \mathbf{O}.$$

Let $F_iCE_j = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ for some $a, b, c, d \in \mathbb{Z}$. Then the above equation is equivalent to $E_i \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} F_j = \mathbf{O}$ or $E_i \begin{bmatrix} a & 0 \\ 0 & 0 \end{bmatrix} F_j = \mathbf{O}$. By Corollary 7, $E_i \begin{bmatrix} a & 0 \\ 0 & 0 \end{bmatrix} F_j = \mathbf{O}$ if and only if $a = 0$. So, we showed the following equivalence: $B_iCB_j = \mathbf{O}$ if and only if $F_iCE_j = \begin{bmatrix} 0 & b \\ c & d \end{bmatrix}$ for some $b, c, d \in \mathbb{Z}$.

Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be the following subsets of $\mathrm{GL}(2,\mathbb{Z})$: $\mathcal{S}_1 = \{F_iCE_j : C \in \langle A_1, \ldots, A_n \rangle\}$ and $\mathcal{S}_2 = \left\{ \begin{bmatrix} 0 & b \\ c & d \end{bmatrix} \in \mathrm{GL}(2,\mathbb{Z}) \ : \ b, c, d \in \mathbb{Z} \right\}$. In this notations the above equivalence can be written as follows: there is a matrix $C \in \langle A_1, \ldots, A_n \rangle$ such that $B_iCB_j = \mathbf{O}$ if and only if $\mathcal{S}_1 \cap \mathcal{S}_2 \neq \emptyset$.

It is easy to see that $\mathcal{S}_1$ is a regular subset of $\mathrm{GL}(2,\mathbb{Z})$ as it can be described by the regular expression $u_i(w_1 + \cdots + w_n)^* v_j$, where $w_1, \ldots, w_n$ are words representing the matrices $A_1, \ldots, A_n$ and u_i, v_j represent the matrices F_i, E_j, respectively. By Proposition 9, $\mathcal{S}_2$ is also a regular subset of $\mathrm{GL}(2,\mathbb{Z})$. Using Theorem 5, we can decide whether $\mathcal{S}_1 \cap \mathcal{S}_2 \neq \emptyset$ and hence decide whether there is a matrix $C \in \langle A_1, \ldots, A_n \rangle$ such that $B_iCB_j = \mathbf{O}$. This finishes the proof of Theorem 11. ◀

3.2 Membership problem

We are now ready to consider the case when M is a nonzero singular matrix.

▶ **Theorem 13.** *Let $\mathcal{F} = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$ be a finite collection of matrices such that $A_1, \ldots, A_n \in \mathrm{GL}(2,\mathbb{Z})$ and $B_1, \ldots, B_m$ are singular matrices from $\mathbb{Z}^{2\times 2}$. Also let $M \in \mathbb{Z}^{2\times 2}$ be a nonzero singular matrix. Then it is decidable whether $M \in \langle \mathcal{F} \rangle$.*

Proof. Let $M = E \begin{bmatrix} t & 0 \\ 0 & 0 \end{bmatrix} F$ be the Smith normal form of M, and for each $j = 1, \ldots, m$, let $B_j = E_j \begin{bmatrix} t_j & 0 \\ 0 & 0 \end{bmatrix} F_j$ be the Smith normal form of B_j. Since M is a nonzero matrix, we may assume that for each $j = 1, \ldots, m$, B_j is also a nonzero matrix. Hence without loss of generality we assume that $t, t_1, \ldots, t_m > 0$.

We will construct a graph $\mathcal{G}(M, \mathcal{F})$, depending on M and $\mathcal{F}$, which will have the following property: $M \in \langle \mathcal{F} \rangle$ if and only if there is a path in $\mathcal{G}(M, \mathcal{F})$ from an initial to a final node of weight t.

Description of $\mathcal{G}(M, \mathcal{F})$. Graph $\mathcal{G}(M, \mathcal{F})$ has m nodes labelled by singular matrices $B_1, \ldots, B_m$ and two special nodes **In** and **Fin**, where **In** is the only initial node and **Fin** is the only final node. The weights of the nodes are defined as follows.

▶ **Definition 14.** Recall that $E_j \begin{bmatrix} t_j & 0 \\ 0 & 0 \end{bmatrix} F_j$ is the Smith normal form of B_j. Then the weight of the node with label B_j is equal to t_j.

Furthermore, we add edges to this graph according to the following rules.

▶ **Definition 15.**

1. For every integer $u \neq 0$ such that $-t \leq u \leq t$ we add an edge from node B_i to node B_j of weight u if and only if there is a matrix $C \in \langle A_1, \ldots, A_n \rangle$ such that $F_iCE_j \in \left\{ \begin{bmatrix} u & b \\ c & d \end{bmatrix} \in \mathrm{GL}(2,\mathbb{Z}) \ : \ b, c, d \in \mathbb{Z} \right\}$.

2. We also add an edge of weight u from the initial node **In** to a node with label B_j if there is a matrix $C \in \langle A_1, \dots, A_n \rangle$ such that $E^{-1}CE_j \in \left\{ \begin{bmatrix} u & b \\ 0 & d \end{bmatrix} \in \mathrm{GL}(2,\mathbb{Z}) \ : \ b, d \in \mathbb{Z} \right\}$.
3. Finally, we add an edge of weight u from a node with label B_j to the final node **Fin** if there is a matrix $C \in \langle A_1, \dots, A_n \rangle$ such that $F_jCF^{-1} \in \left\{ \begin{bmatrix} u & 0 \\ c & d \end{bmatrix} \in \mathrm{GL}(2,\mathbb{Z}) \ : \ c, d \in \mathbb{Z} \right\}$.

Note that the set $\{F_iCE_j : C \in \langle A_1, \dots, A_n \rangle\}$ is a regular subset of $\mathrm{GL}(2,\mathbb{Z})$ because it can be described by the regular expression $u_i(w_1 + \cdots + w_n)^* v_j$, where $w_1, \dots, w_n$ are words representing the matrices $A_1, \dots, A_n$ and u_i, v_j represent the matrices F_i, E_j, respectively. By Proposition 9, $\left\{ \begin{bmatrix} u & b \\ c & d \end{bmatrix} \in \mathrm{GL}(2,\mathbb{Z}) \ : \ b, c, d \in \mathbb{Z} \right\}$ is also a regular subset of $\mathrm{GL}(2,\mathbb{Z})$. Therefore, by Theorem 5, we can algorithmically decide if there is an edge from node B_i to node B_j of weight u.

Moreover, the edges going out of **In** or ending in **Fin** can only have weights 1 or -1. Again, using Proposition 9, Corollary 10 and Theorem 5, we can algorithmically decide if there is an edge from **In** to B_j or from B_j to **Fin** of weight 1 or -1.

▶ **Definition 16.** The weight of a path in $\mathcal{G}(M, \mathcal{F})$ from **In** to **Fin** is equal to the product of the weights of nodes and edges that occur in it. That is, the weight of a path

$$\mathbf{In} \xrightarrow{u_0} B_{i_0} \xrightarrow{u_1} B_{i_1} \xrightarrow{u_2} B_{i_2} \quad \cdots \quad B_{i_{s-1}} \xrightarrow{u_s} B_{i_s} \xrightarrow{u_{s+1}} \mathbf{Fin}$$

is equal to $u_0 t_{i_0} u_1 t_{i_1} u_2 t_{i_2} \cdots t_{i_{s-1}} u_s t_{i_s} u_{s+1}$.

In the following proposition we will show that the membership problem is equivalent to the existence of a path in $\mathcal{G}(M, F)$ with a given weight.

▶ **Proposition 17.** *Let $M = E \begin{bmatrix} t & 0 \\ 0 & 0 \end{bmatrix} F$ be the Smith normal form of matrix M. Then $M \in \langle \mathcal{F} \rangle$ if and only if there is a path in $\mathcal{G}(M, \mathcal{F})$ from **In** to **Fin** of weight t.*

Proof. Suppose

$$\mathbf{In} \xrightarrow{u_0} B_{i_0} \xrightarrow{u_1} B_{i_1} \xrightarrow{u_2} B_{i_2} \quad \cdots \quad B_{i_{s-1}} \xrightarrow{u_s} B_{i_s} \xrightarrow{u_{s+1}} \mathbf{Fin}$$

is a path in $\mathcal{G}(M, \mathcal{F})$ from **In** to **Fin** of weight t. Recall that for every $r = 0, \dots, s$, we have $B_{i_r} = E_{i_r} \begin{bmatrix} t_{i_r} & 0 \\ 0 & 0 \end{bmatrix} F_{i_r}$. Hence $t = u_0 t_{i_0} u_1 t_{i_1} u_2 t_{i_2} \cdots t_{i_{s-1}} u_s t_{i_s} u_{s+1}$.

Since for every $r = 1, \dots, s$ we have an edge $B_{i_{r-1}} \xrightarrow{u_r} B_{i_r}$ of weight u_r, there is a matrix $C_r \in \langle A_1, \dots, A_n \rangle$ such that $F_{i_{r-1}} C_r E_{i_r} = \begin{bmatrix} u_r & b_r \\ c_r & d_r \end{bmatrix}$ for some $b_r, c_r, d_r \in \mathbb{Z}$. Since we have an edge $\mathbf{In} \xrightarrow{u_0} B_{i_0}$ of weight u_0, there is a matrix $C_0 \in \langle A_1, \dots, A_n \rangle$ such that $E^{-1} C_0 E_{i_0} = \begin{bmatrix} u_0 & b_0 \\ 0 & d_0 \end{bmatrix}$ for some $b_0, d_0 \in \mathbb{Z}$. And since we have an edge $B_{i_s} \xrightarrow{u_{s+1}} \mathbf{Fin}$ of weight u_{s+1}, there is a matrix $C_{s+1} \in \langle A_1, \dots, A_n \rangle$ such that $F_{i_s} C_{s+1} F^{-1} = \begin{bmatrix} u_{s+1} & 0 \\ c_{s+1} & d_{s+1} \end{bmatrix}$ for some $c_{s+1}, d_{s+1} \in \mathbb{Z}$.

Hence we obtain the following equation

$$
\begin{aligned}
&E^{-1}C_0B_{i_0}C_1B_{i_1}C_2B_{i_2}\cdots B_{i_{s-1}}C_sB_{i_s}C_{s+1}F^{-1} = \\
&E^{-1}C_0E_{i_0}\begin{bmatrix}t_{i_0} & 0\\ 0 & 0\end{bmatrix}F_{i_0}C_1E_{i_1}\begin{bmatrix}t_{i_1} & 0\\ 0 & 0\end{bmatrix}F_{i_1}C_2E_{i_2}\begin{bmatrix}t_{i_2} & 0\\ 0 & 0\end{bmatrix}F_{i_2}\cdots \\
&\qquad\cdots E_{i_{s-1}}\begin{bmatrix}t_{i_{s-1}} & 0\\ 0 & 0\end{bmatrix}F_{i_{s-1}}C_sE_{i_s}\begin{bmatrix}t_{i_s} & 0\\ 0 & 0\end{bmatrix}F_{i_s}C_{s+1}F^{-1} = \\
&t_{i_0}t_{i_1}t_{i_2}\cdots t_{i_{s-1}}t_{i_s}\begin{bmatrix}u_0 & b_0\\ 0 & d_0\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_1 & b_1\\ c_1 & d_1\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_2 & b_2\\ c_2 & d_2\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\cdots \\
&\qquad\cdots\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_s & b_s\\ c_s & d_s\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_{s+1} & 0\\ c_{s+1} & d_{s+1}\end{bmatrix} = \\
&t_{i_0}u_1t_{i_1}u_2t_{i_2}\cdots t_{i_{s-1}}u_st_{i_s}\begin{bmatrix}u_0 & b_0\\ 0 & d_0\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_{s+1} & 0\\ c_{s+1} & d_{s+1}\end{bmatrix} = \\
&t_{i_0}u_1t_{i_1}u_2t_{i_2}\cdots t_{i_{s-1}}u_st_{i_s}\begin{bmatrix}u_0u_{s+1} & 0\\ 0 & 0\end{bmatrix} = u_0t_{i_0}u_1t_{i_1}\cdots t_{i_{s-1}}u_st_{i_s}u_{s+1}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix} = \begin{bmatrix}t & 0\\ 0 & 0\end{bmatrix}.
\end{aligned}
$$

Therefore, $C_0B_{i_0}C_1B_{i_1}C_2B_{i_2}\cdots B_{i_{s-1}}C_sB_{i_s}C_{s+1} = E\begin{bmatrix}t & 0\\ 0 & 0\end{bmatrix}F = M$, and hence $M \in \langle\mathcal{F}\rangle$.

Now suppose that $M \in \langle\mathcal{F}\rangle$. It is not hard to see that there is a sequence of indices $i_0, i_1, \ldots, i_s \in \{1,\ldots,m\}$, and matrices $C_0, C_1, \ldots, C_{s+1} \in \langle A_1,\ldots,A_n\rangle$ such that

$$C_0B_{i_0}C_1B_{i_1}C_2B_{i_2}\cdots B_{i_{s-1}}C_sB_{i_s}C_{s+1} = M. \tag{5}$$

Recall that $E\begin{bmatrix}t & 0\\ 0 & 0\end{bmatrix}F$ is the Smith normal form of M, and $E_{i_r}\begin{bmatrix}t_{i_r} & 0\\ 0 & 0\end{bmatrix}F_{i_r}$ is the Smith normal form of B_{i_r}, for $r = 0,\ldots,s$. So we can rewrite (5) as follows

$$
\begin{aligned}
&E^{-1}C_0E_{i_0}\begin{bmatrix}t_{i_0} & 0\\ 0 & 0\end{bmatrix}F_{i_0}C_1E_{i_1}\begin{bmatrix}t_{i_1} & 0\\ 0 & 0\end{bmatrix}F_{i_1}C_2E_{i_2}\begin{bmatrix}t_{i_2} & 0\\ 0 & 0\end{bmatrix}F_{i_2}\cdots \\
&\qquad\cdots E_{i_{s-1}}\begin{bmatrix}t_{i_{s-1}} & 0\\ 0 & 0\end{bmatrix}F_{i_{s-1}}C_sE_{i_s}\begin{bmatrix}t_{i_s} & 0\\ 0 & 0\end{bmatrix}F_{i_s}C_{s+1}F^{-1} = \begin{bmatrix}t & 0\\ 0 & 0\end{bmatrix}.
\end{aligned}
\tag{6}
$$

For $r = 1,\ldots,s$, let $F_{i_{r-1}}C_rE_{i_r} = \begin{bmatrix}u_r & b_r\\ c_r & d_r\end{bmatrix}$. Then for every $r = 1,\ldots,s$, there is an edge $B_{i_{r-1}} \xrightarrow{u_r} B_{i_r}$ in $\mathcal{G}(M,\mathcal{F})$ of weight u_r. Furthermore, suppose that $E^{-1}C_0E_{i_0} = \begin{bmatrix}u_0 & b_0\\ c_0 & d_0\end{bmatrix}$ and $F_{i_s}C_{s+1}F^{-1} = \begin{bmatrix}u_{s+1} & b_{s+1}\\ c_{s+1} & d_{s+1}\end{bmatrix}$. Then we can rewrite (6) as

$$
\begin{aligned}
&\begin{bmatrix}u_0 & b_0\\ c_0 & d_0\end{bmatrix}\begin{bmatrix}t_{i_0} & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_1 & b_1\\ c_1 & d_1\end{bmatrix}\begin{bmatrix}t_{i_1} & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_2 & b_2\\ c_2 & d_2\end{bmatrix}\begin{bmatrix}t_{i_2} & 0\\ 0 & 0\end{bmatrix}\cdots \\
&\qquad\cdots\begin{bmatrix}t_{i_{s-1}} & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_s & b_s\\ c_s & d_s\end{bmatrix}\begin{bmatrix}t_{i_s} & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_{s+1} & b_{s+1}\\ c_{s+1} & d_{s+1}\end{bmatrix} = \begin{bmatrix}t & 0\\ 0 & 0\end{bmatrix}
\end{aligned}
$$

or equivalently

$$
\begin{aligned}
&t_{i_0}t_{i_1}t_{i_2}\cdots t_{i_{s-1}}t_{i_s}\begin{bmatrix}u_0 & b_0\\ c_0 & d_0\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_1 & b_1\\ c_1 & d_1\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_2 & b_2\\ c_2 & d_2\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\cdots \\
&\qquad\cdots\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_s & b_s\\ c_s & d_s\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_{s+1} & b_{s+1}\\ c_{s+1} & d_{s+1}\end{bmatrix} = \begin{bmatrix}t & 0\\ 0 & 0\end{bmatrix}.
\end{aligned}
$$

From this equation we obtain

$$t_{i_0}u_1t_{i_1}u_2t_{i_2}\cdots t_{i_{s-1}}u_st_{i_s}\begin{bmatrix}u_0 & b_0\\ c_0 & d_0\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_{s+1} & b_{s+1}\\ c_{s+1} & d_{s+1}\end{bmatrix}=\begin{bmatrix}t & 0\\ 0 & 0\end{bmatrix} \quad \text{or}$$

$$t_{i_0}u_1t_{i_1}u_2t_{i_2}\cdots t_{i_{s-1}}u_st_{i_s}\begin{bmatrix}u_0u_{s+1} & u_0b_{s+1}\\ c_0u_{s+1} & c_0b_{s+1}\end{bmatrix}=\begin{bmatrix}t & 0\\ 0 & 0\end{bmatrix}.$$

Therefore, we have that $t = u_0t_{i_0}u_1t_{i_1}u_2t_{i_2}\cdots t_{i_{s-1}}u_st_{i_s}u_{s+1}$ and $u_0b_{s+1} = c_0u_{s+1} = 0$. By assumption $t \neq 0$, and so $u_0 \neq 0$ and $u_{s+1} \neq 0$. Therefore, $c_0 = 0$ and $b_{s+1} = 0$. Hence we have that $E^{-1}C_0E_{i_0} = \begin{bmatrix}u_0 & b_0\\ 0 & d_0\end{bmatrix}$ and $F_{i_s}C_{s+1}F^{-1} = \begin{bmatrix}u_{s+1} & 0\\ c_{s+1} & d_{s+1}\end{bmatrix}$, which means that there is an edge $\mathbf{In} \xrightarrow{u_0} B_{i_0}$ of weight u_0 and an edge $B_{i_s} \xrightarrow{u_{s+1}} \mathbf{Fin}$ of weight u_{s+1}. Thus we showed that there is path

$$\mathbf{In} \xrightarrow{u_0} B_{i_0} \xrightarrow{u_1} B_{i_1} \xrightarrow{u_2} B_{i_2} \quad \cdots \quad B_{i_{s-1}} \xrightarrow{u_s} B_{i_s} \xrightarrow{u_{s+1}} \mathbf{Fin}$$

in $\mathcal{G}(M, \mathcal{F})$ from **In** to **Fin** of weight $u_0t_{i_0}u_1t_{i_1}u_2t_{i_2}\cdots t_{i_{s-1}}u_st_{i_s}u_{s+1} = t$. ◀

The next proposition provides a bound on the length of a path in $\mathcal{G}(M, \mathcal{F})$ with weight t.

▶ **Proposition 18.** *For any integer $t > 0$, if there is a path in $\mathcal{G}(M, \mathcal{F})$ from **In** to **Fin** of weight t, then there is such path of length at most $2m\log_2 t + 2m + \log_2 t$.*

Proof. Suppose P is a path in $\mathcal{G}(M, \mathcal{F})$ from **In** to **Fin** of weight t. Then the number of nodes and edges in P whose weight is greater than 1 or less than -1 is bounded by $\log_2 t$.

A *simple cycle* at node B_j is a closed path that starts and ends at B_j and in which no vertex appears twice except for B_j itself.

Note that if P contains a simple cycle of weight 1, then it can be removed from P without changing its weight. On the other hand, if P contains a simple cycle of weight -1, then removing such cycle will change the sign of the weight of P.

Let W_1 and W_2 be a successive pair of nodes or edges in P with weight different from ± 1. Then any node and edge that appears in P strictly between W_1 and W_2 has weight equal to ± 1. By the above observation we can remove all cycles of weight 1 that occur between W_1 and W_2 and leave at most one simple cycle of weight -1 between W_1 and W_2 in order to preserve the sign of the weight of P. So we can replace the original path from W_1 to W_2 by a new path with the same weight and length at most $2m$.

Recall there are at most $\log_2 t$ nodes and edges in P whose weight is different from ± 1. We now apply the above procedure to every pair W_1 and W_2 of successive nodes or edges in P whose weight is different from ± 1 including the cases when $W_1 = \mathbf{In}$ or $W_2 = \mathbf{Fin}$. There are at most $\log_2 t + 1$ such successive pairs. Therefore, we replace the whole path P with another path of the same weight and of length at most $2m(\log_2 t + 1) + \log_2 t$. Note that we added $\log_2 t$ in the end because every edge of weight different from ± 1 contributes 1 to the length of the path. ◀

Now we complete the proof of Theorem 13 using Propositions 17 and 18. Indeed, by Propositions 17 to decide whether $M \in \langle \mathcal{F} \rangle$, we need to check if there is a path in $\mathcal{G}(M, \mathcal{F})$ from **In** to **Fin** of weight t. By Propositions 18 the length of such path is bounded by $2m\log_2 t + 2m + \log_2 t$. Hence we can check all paths in $\mathcal{G}(M, \mathcal{F})$ of length up to $2m\log_2 t + 2m + \log_2 t$ to see if there is one with weight t. ◀

Conclusion and future work

The complexity of our algorithm is in EXPTIME. This is because a canonical word that represents a given matrix M has length exponential in the binary presentation of M. Hence the construction of regular languages in our proof takes exponential time. Moreover, the number of paths in $\mathcal{G}(M,\mathcal{F})$ of length up to $2m\log_2 t + 2m + \log_2 t$ is exponential in m.

In [4] it has been shown that the identity problem in $\mathrm{SL}(2,\mathbb{Z})$ is NP-complete. We would like to find out whether this construction can be combined with our result to show that the membership in $\mathrm{GL}(2,\mathbb{Z})$ extended by singular matrices is also NP-complete.

In our previous work [25] we proved that the membership problem is decidable for 2×2 nonsingular integer matrices. In this paper we considered matrices with determinants $0,\pm 1$. So, the next natural step will be to study the decidability of the membership problem for all 2×2 integer matrices, i.e. both singular and nonsingular ones.

References

1 László Babai, Robert Beals, Jin-yi Cai, Gábor Ivanyos, and Eugene M. Luks. Multiplicative equations over commuting matrices. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'96, pages 498–507, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

2 Paul Bell and Igor Potapov. On undecidability bounds for matrix decision problems. *Theoretical Computer Science*, 391(1-2):3–13, 2008.

3 Paul C. Bell, Mika Hirvensalo, and Igor Potapov. Mortality for 2x2 matrices is NP-hard. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 148–159. Springer Berlin Heidelberg, 2012.

4 Paul C. Bell, Mika Hirvensalo, and Igor Potapov. The identity problem for matrix semigroups in $\mathrm{SL}_2(\mathbb{Z})$ is NP-complete. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 187–206, 2017. doi:10.1137/1.9781611974782.13.

5 Paul C. Bell and Igor Potapov. On the undecidability of the identity correspondence problem and its applications for word and matrix semigroups. *Int. J. Found. Comput. Sci.*, 21(6):963–978, 2010.

6 Paul C. Bell and Igor Potapov. On the computational complexity of matrix semigroup problems. *Fundam. Inf.*, 116(1-4):1–13, January 2012.

7 Vincent D. Blondel, Emmanuel Jeandel, Pascal Koiran, and Natacha Portier. Decidable and undecidable problems about quantum automata. *SIAM J. Comput.*, 34(6):1464–1473, June 2005.

8 Julien Cassaigne, Vesa Halava, Tero Harju, and François Nicolas. Tighter undecidability bounds for matrix mortality, zero-in-the-corner problems, and more. *CoRR*, abs/1404.0644, 2014.

9 Christian Choffrut and Juhani Karhumaki. Some decision problems on integer matrices. *RAIRO-Theor. Inf. Appl.*, 39(1):125–131, 2005.

10 J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Logic in Computer Science, 1999. Proceedings. 14th Symposium on*, pages 352–359, 1999.

11 Esther Galby, Joël Ouaknine, and James Worrell. On Matrix Powering in Low Dimensions. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 329–340, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

12 Yuri Gurevich and Paul Schupp. Membership problem for the modular group. *SIAM J. Comput.*, 37(2):425–459, May 2007.

13 Vesa Halava, Tero Harju, Mika Hirvensalo, and Juhani Karhumaki. Skolem's problem - on the border between decidability and undecidability. Technical Report 683, Turku Centre for Computer Science, 2005.

14 R. Kannan and R. J. Lipton. Polynomial-time algorithm for the orbit problem. *J. ACM*, 33(4):808–821, August 1986. doi:10.1145/6490.6496.

15 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.

16 Alexei Lisitsa and Igor Potapov. Membership and reachability problems for row-monomial transformations. In *Mathematical Foundations of Computer Science 2004, 29th International Symposium, MFCS 2004, Prague, Czech Republic, August 22-27, 2004, Proceedings*, pages 623–634, 2004.

17 Roger C. Lyndon and Paul E. Schupp. *Combinatorial group theory*. Springer-Verlag, Berlin-New York, 1977. Ergebnisse der Mathematik und ihrer Grenzgebiete, Band 89.

18 Wilhelm Magnus, Abraham Karrass, and Donald Solitar. *Combinatorial group theory*. Dover Publications, Inc., New York, revised edition, 1976. Presentations of groups in terms of generators and relations.

19 C. Nuccio and Emanuele Rodaro. Mortality problem for 2×2 integer matrices. In *SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 19-25, 2008, Proceedings*, pages 400–405, 2008. doi:10.1007/978-3-540-77566-9_34.

20 Joël Ouaknine, João Sousa Pinto, and James Worrell. On termination of integer linear loops. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'15, pages 957–969. SIAM, 2015.

21 Joël Ouaknine and James Worrell. On the positivity problem for simple linear recurrence sequences,. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 318–329, 2014.

22 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 330–341, 2014.

23 M. S. Paterson. Unsolvability in 3×3 matrices. *Studies in Applied Mathematics*, 49(1):pp.105–107, 1970.

24 Igor Potapov and Pavel Semukhin. Vector reachability problem in SL(2,Z). In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 84:1–84:14, 2016. doi:10.4230/LIPIcs.MFCS.2016.84.

25 Igor Potapov and Pavel Semukhin. Decidability of the membership problem for 2×2 integer matrices. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 170–186, 2017. doi:10.1137/1.9781611974782.12.

26 Robert A. Rankin. *Modular forms and functions*. Cambridge University Press, Cambridge-New York-Melbourne, 1977.

Grammars for Indentation-Sensitive Parsing

Härmel Nestra

Institute of Computer Science, University of Tartu, Tartu, Estonia
harmel.nestra@ut.ee

Abstract

Adams' extension of parsing expression grammars enables specifying indentation sensitivity using two non-standard grammar constructs – indentation by a binary relation and alignment. This paper is a theoretical study of Adams' grammars. It proposes a step-by-step transformation of well-formed Adams' grammars for elimination of the alignment construct from the grammar. The idea that alignment could be avoided was suggested by Adams but no process for achieving this aim has been described before. This paper also establishes general conditions that binary relations used in indentation constructs must satisfy in order to enable efficient parsing.

1998 ACM Subject Classification D.3.1 Formal Definitions and Theory, D.3.4 Processors, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases Parsing expression grammars, indentation, grammar transformation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.45

1 Introduction

Parsing expression grammars (PEG) introduced by Ford [6] serve as a modern framework for specifying the syntax of programming languages and are an alternative to the classic context-free grammars (CFG). The core difference between CFG and PEG is that descriptions in CFG can be ambiguous while PEGs are inherently deterministic. A syntax specification written in PEG can in principle be interpreted as a top-down parser for that syntax; in the case of left recursion, this treatment is not straightforward but doable (see, e.g., [8]).

Formally, a PEG is a quadruple $G = (N, T, \delta, \mathsf{s})$ where:

- N is a finite set of *non-terminals*;
- T is a finite set of *terminals*;
- δ is a function mapping each non-terminal to its replacement (corresponding to the set of productions of CFG);
- s is the *start expression* (corresponding to the start symbol of CFG).

So $\delta : N \to \mathcal{E}_G$ and $\mathsf{s} \in \mathcal{E}_G$, where the set $\mathcal{E}_G$ of all *parsing expressions* writable in G is defined inductively as follows:

1. $\varepsilon \in \mathcal{E}_G$ (the empty string);
2. $a \in \mathcal{E}_G$ for every $a \in T$ (the terminals);
3. $X \in \mathcal{E}_G$ for every $X \in N$ (the non-terminals);
4. $\mathsf{pq} \in \mathcal{E}_G$ whenever $\mathsf{p} \in \mathcal{E}_G$, $\mathsf{q} \in \mathcal{E}_G$ (*concatenation*)
5. $\mathsf{p}/\mathsf{q} \in \mathcal{E}_G$ whenever $\mathsf{p} \in \mathcal{E}_G$, $\mathsf{q} \in \mathcal{E}_G$ (*choice*);
6. $!\mathsf{p} \in \mathcal{E}_G$ whenever $\mathsf{p} \in \mathcal{E}_G$ (*negation*, or *lookahead*);
7. $\mathsf{p}^* \in \mathcal{E}_G$ whenever $\mathsf{p} \in \mathcal{E}_G$ (*repetition*).

All constructs of PEG except for negation are direct analogues of constructs of the EBNF form of CFG, but their semantics is always deterministic. So p^* repeats parsing of p until failure, and p/q always tries to parse p first, q is parsed only if p fails. For example, the expression ab/a consumes the input string `ab` entirely while a/ab only consumes its first

character. The corresponding EBNF expressions $ab \mid a$ and $a \mid ab$ are equivalent, both can match either a or ab from the input string. Negation $!p$ tries to parse p and fails if p succeeds; if p fails then $!p$ succeeds with consuming no input. Other constructs of EBNF like non-null repetition p^+ and optional occurrence $[p]$ can be introduced to PEG as syntactic sugar.

Languages like Python and Haskell allow the syntactic structure of programs to be shown by indentation and alignment, instead of the more conventional braces and semicolons. Handling indentation and alignment in Python has been specified in terms of extra tokens INDENT and DEDENT that mark increasing and decreasing of indentation and must be generated by the lexer. In Haskell, rules for handling indentation and alignment are more sophisticated. Both these languages enable to locally use a different layout mode where indentation does not matter, which additionally complicates the task of formal syntax specification. Adams and Ağacan [3] proposed an extension of PEG notation for specifying indentation sensitivity and argued that it considerably simplifies this task for Python, Haskell and many other indentation-sensitive languages.

In this extension, expression $p^{>}$, for example, denotes parsing of p while assuming a greater indentation than that of the surrounding block. In general, parsing expressions may be equipped with binary relations (as was $>$ in the example) that must hold between the baselines of the local and the current indentation block. In addition, $\lfloor p \rfloor$ denotes parsing of p while assuming the first token of the input being aligned, i.e., positioned on the current indentation baseline. For example, the do expressions in Haskell can be specified by

$$\begin{array}{lcl} \langle doexp \rangle & ::= & \texttt{do}^{>}\ (\langle istmts \rangle / \langle stmts \rangle) \\ \langle istmts \rangle & ::= & (\lfloor \langle stmt \rangle \rfloor^{+})^{>} \\ \langle stmts \rangle & ::= & \texttt{\{}^{>} (\langle stmt \rangle (\texttt{;} \langle stmt \rangle)^{*} [\texttt{;}] \texttt{\}})^{\circledast} \end{array}$$

Here, $\langle istmts \rangle$ and $\langle stmts \rangle$ stand for statement lists in the indentation and relaxed mode, respectively. In the indentation mode, a statement list is indented (marked by $>$ in the second production) and all statements in it are aligned (marked by $\lfloor \cdot \rfloor$). In the relaxed mode, however, relation $\circledast$ is used to indicate that the indentation baseline of the contents can be anything. (Technically, $\circledast$ is the binary relation containing all pairs of natural numbers.) Terminals `do` and `{` are also equipped with $>$ to meet the Haskell requirement that subsequent tokens of aligned blocks must be indented more than the first token.

Alignment construct provides fulcra for disambiguating the often large variety of indentation baseline candidates. Besides simplicity of this grammar extension and its use, a strength of it lies in the fact that grammars can still serve as parsers.

The rest of the paper is organized as follows. Section 2 formally introduces additional constructs of PEG for specifying code layout, defines their semantics and studies their semantic properties. In Sect. 3, a semantics-preserving process of eliminating the alignment construct from grammars is described. General criteria for deciding if parsing can handle a relation efficiently are found in Sect. 4. Section 5 refers to related work and Sect. 6 concludes.

2 Indentation extension of PEG

Adams and Ağacan [3] extend PEGs with the indentation and alignment constructs. We propose a slightly different extension with three rather than two extra constructs. Our approach agrees with that implemented by Adams in his `indentation` package for Haskell [1], whence calling the grammars in our approach *Adams' grammars* is justified. All differences between the definitions in this paper and in [3] are listed and discussed in Subsect. 2.4.

Let $\mathbb{N}$ denote the set of all natural numbers, and let $\mathbb{B} = \{tt, ff\}$ (the Boolean domain). Denote by $\wp(X)$ the set of all subsets of set X, and let $\Re(X)$ denote the set of all binary

relations on set X, i.e., $\Re(X) = \wp(X \times X)$. Standard examples are $>\in \Re(\mathbb{N})$ (consisting of all pairs (n, m) of natural numbers such that $n > m$) and $\triangle \in \Re(\mathbb{N})$ (the identity relation consisting of all pairs of equal natural numbers); the indentation extension also makes use of $\circledast \in \Re(\mathbb{N})$ (the relation containing all pairs of natural numbers). Whenever $\rho \in \Re(X)$ and $Y \subseteq X$, denote $\rho(Y) = \{x \in X : \exists y \in Y.(y, x) \in \rho\}$ (the image of Y under relation ρ). The inverse relation of ρ is defined by $\rho^{-1} = \{(x, y) : (y, x) \in \rho\}$, and the composition of relations σ and ρ by $\sigma \circ \rho = \{(x, z) : \exists y.(x, y) \in \sigma \wedge (y, z) \in \rho\}$. Finally, denote $\Re^+(X) = \left\{\rho \in \Re(X) : \forall x \in X.\rho^{-1}(\{x\}) \neq \varnothing\right\} = \{\rho \in \Re(X) : \rho(X) = X\}$.

2.1 Adams' grammars

Extend the definition of $\mathcal{E}_G$ given in Sect. 1 with the following three additional clauses:

8. $p^\rho \in \mathcal{E}_G$ for every $p \in \mathcal{E}_G$ and $\rho \in \Re(\mathbb{N})$ (*indentation*);
9. $p_\sigma \in \mathcal{E}_G$ for every $p \in \mathcal{E}_G$ and $\sigma \in \Re(\mathbb{N})$ (*token position*);
10. $\lfloor p \rfloor \in \mathcal{E}_G$ for every $p \in \mathcal{E}_G$ (*alignment*).

Parsing of an expression p^ρ means parsing of p while assuming that the part of the input string corresponding to p forms a new indentation block whose baseline is in relation ρ to the baseline of the surrounding block. (Baselines are identified with column numbers.) The position construct p_σ, missing in [3], determines how tokens of the input can be situated w.r.t. the current indentation baseline. Finally, parsing an expression $\lfloor p \rfloor$ means parsing of p while assuming the first token of the input being positioned on the current indentation baseline (unlike the position operator, this construct does not affect processing the subsequent tokens).

Inspired by the `indentation` package [1], we call the relations that determine token positioning w.r.t. the indentation baseline *token modes*. In the token mode $>$ for example, tokens may appear only to the right of the indentation baseline. Applying the position operator with relation $>$ to parts of Haskell grammar to be parsed in the indentation mode avoids indenting every single terminal in the example in Sect. 1. Also, indenting terminals with $>$ is inadequate for do expressions occurring inside a block of relaxed mode but the position construct can be easily used to change the token mode for such blocks (e.g., to $\geq$).

We call a PEG extended with these three constructs a PEG$^>$. Recall from Sect. 1 that N and T denote the set of non-terminal and terminal symbols of the grammar, respectively, and $\delta : N \to \mathcal{E}_G$ is the production function. Concerning the semantics of PEG$^>$, each expression parses an input string of terminals ($w \in T^*$) in the context of a current set of indentation baseline candidates ($I \in \wp(\mathbb{N})$) and a current alignment flag indicating whether the next terminal should be aligned or not ($b \in \mathbb{B}$), assuming a certain token mode ($\tau \in \Re(\mathbb{N})$). Parsing may succeed, fail, or diverge. If parsing succeeds, it returns as a result a new triple containing the rest of the input w', a new set I' of baseline candidates updated according to the information gathered during parsing, and a new alignment flag b'. This result is denoted by $\top(w', I', b')$. If parsing fails, there is no result in a triple form; failure is denoted by $\bot$.

Triples of the form $(w, I, b) \in T^* \times \wp(\mathbb{N}) \times \mathbb{B}$ are behaving as *operation states* of parsing, as each parsing step may use these data and update them. We will write $State = T^* \times \wp(\mathbb{N}) \times \mathbb{B}$ (as we never deal with different terminal sets, dependence on T is not explicitly marked), and denote by $State + 1$ the set of possible results of parsing, i.e., $\{\top(s) : s \in State\} \cup \{\bot\}$.

The assertion that parsing expression e in grammar G with input string w in the context of I and b assuming token mode τ results in $o \in State + 1$ is denoted by $e, \tau \vdash_G (w, I, b) \to o$. The formal definition below must be interpreted inductively, i.e., an assertion of the form $G, \tau \vdash_e s \to o$ is valid iff it has a finite derivation by the following ten rules:

1. $\varepsilon, \tau \vdash_G s \to \top(s)$.
2. For every $a \in T$, $a, \tau \vdash_G (w, I, b) \to o$ holds in two cases:
   - If $o = \top(w', I', \mathit{ff})$ for w', I', i such that $w = a^i w'$ (a^i denotes a occurring at column i) and either $b = \mathit{ff}$ and $i \in \tau^{-1}(I)$, $I' = I \cap \tau(\{i\})$, or $b = \mathit{tt}$ and $i \in I$, $I' = \{i\}$;
   - If $o = \bot$, and there are no w' and i such that $w = a^i w'$ with either $b = \mathit{ff}$ and $i \in \tau^{-1}(I)$ or $b = \mathit{tt}$ and $i \in I$.
3. For every $X \in N$, $X, \tau \vdash_G s \to o$ holds if $\delta(X), \tau \vdash_G s \to o$ holds.
4. For every $\mathsf{p}, \mathsf{q} \in \mathcal{E}_G$, $\mathsf{pq}, \tau \vdash_G s \to o$ holds in two cases:
   - If there exists a triple s' such that $\mathsf{p}, \tau \vdash_G s \to \top(s')$ and $\mathsf{q}, \tau \vdash_G s' \to o$;
   - If $\mathsf{p}, \tau \vdash_G s \to \bot$ and $o = \bot$.
5. For every $\mathsf{p}, \mathsf{q} \in \mathcal{E}_G$, $\mathsf{p}/\mathsf{q}, \tau \vdash_G s \to o$ holds in two cases:
   - If there exists a triple s' such that $\mathsf{p}, \tau \vdash_G s \to \top(s')$ and $o = \top(s')$;
   - If $\mathsf{p}, \tau \vdash_G s \to \bot$ and $\mathsf{q}, \tau \vdash_G s \to o$.
6. For every $\mathsf{p} \in \mathcal{E}_G$, $!\mathsf{p}, \tau \vdash_G s \to o$ holds in two cases:
   - If $\mathsf{p}, \tau \vdash_G s \to \bot$ and $o = \top(s)$;
   - If there exists a triple s' such that $\mathsf{p}, \tau \vdash_G s \to \top(s')$ and $o = \bot$.
7. For every $\mathsf{p} \in \mathcal{E}_G$, $\mathsf{p}^*, \tau \vdash_G s \to o$ holds in two cases:
   - If $\mathsf{p}, \tau \vdash_G s \to \bot$ and $o = \top(s)$;
   - If there exists a triple s' such that $\mathsf{p}, \tau \vdash_G s \to \top(s')$ and $\mathsf{p}^*, \tau \vdash_G s' \to o$.
8. For every $\mathsf{p} \in \mathcal{E}_G$ and $\varrho \in \Re(\mathbb{N})$, $\mathsf{p}^{\varrho}, \tau \vdash_G (w, I, b) \to o$ holds in two cases:
   - If there exists a triple (w', I', b') such that $\mathsf{p}, \tau \vdash_G (w, \varrho^{-1}(I), b) \to \top(w', I', b')$ and $o = \top(w', I \cap \varrho(I'), b')$;
   - If $\mathsf{p}, \tau \vdash_G (w, \varrho^{-1}(I), b) \to \bot$ and $o = \bot$.
9. For every $\mathsf{p} \in \mathcal{E}_G$ and $\sigma \in \Re(\mathbb{N})$, $\mathsf{p}_{\sigma}, \tau \vdash_G s \to o$ holds if $\mathsf{p}, \sigma \vdash_G s \to o$ holds.
10. For every $\mathsf{p} \in \mathcal{E}_G$, $\lfloor \mathsf{p} \rfloor, \tau \vdash_G (w, I, b) \to o$ holds in two cases:
    - If there exists a triple (w', I', b') such that $\mathsf{p}, \tau \vdash_G (w, I, \mathit{tt}) \to \top(w', I', b')$ and $o = \top(w', I', b \wedge b')$;
    - If $\mathsf{p}, \tau \vdash_G (w, I, \mathit{tt}) \to \bot$ and $o = \bot$.

The idea behind the conditions $i \in \tau^{-1}(I)$ and $i \in I$ occurring in clause 2 is that any column i where a token may appear is in relation τ with the current indentation baseline (known to be in I) if the alignment flag is false, and coincides with the indentation baseline otherwise. For the same reason, consuming a token in column i restricts the set of allowed indentations to $\tau(\{i\})$ or $\{i\}$ depending on the alignment flag. In both cases, the alignment flag is set to ff. Similar principles lie behind the changes of the operation states in clauses 8 and 10.

For a toy example, consider parsing of $\lfloor ab \rfloor^{>}$ with the operation state $(a^2b^3, \mathbb{N}, \mathit{ff})$ assuming the token mode $\geq$. For that, we must parse $\lfloor ab \rfloor$ with $(a^2b^3, \mathbb{N} \setminus \{0\}, \mathit{ff})$ by clause 8 since $>^{-1}(\mathbb{N}) = \mathbb{N} \setminus \{0\}$. For that in turn, we must parse ab with $(a^2b^3, \mathbb{N} \setminus \{0\}, \mathit{tt})$ by clause 10. By clause 2, we have $a, \geq \vdash_G (a^2b^3, \mathbb{N} \setminus \{0\}, \mathit{tt}) \to \top(b^3, \{2\}, \mathit{ff})$ (as $2 \in \mathbb{N} \setminus \{0\}$) and $b, \geq \vdash_G (b^3, \{2\}, \mathit{ff}) \to \top(\varepsilon, \{2\}, \mathit{ff})$ (as $(2, 3) \in \geq^{-1}$). Therefore, by clause 4, $ab, \geq \vdash_G (a^2b^3, \mathbb{N} \setminus \{0\}, \mathit{tt}) \to \top(\varepsilon, \{2\}, \mathit{ff})$. Finally, $\lfloor ab \rfloor, \geq \vdash_G (a^2b^3, \mathbb{N} \setminus \{0\}, \mathit{ff}) \to \top(\varepsilon, \{2\}, \mathit{ff})$ and $\lfloor ab \rfloor^{>}, \geq \vdash_G (a^2b^3, \mathbb{N}, \mathit{ff}) \to \top(\varepsilon, \{0, 1\}, \mathit{ff})$ by clauses 10 and 8. The set $\{0, 1\}$ in the final state shows that only 0 and 1 are still candidates for the indentation baseline outside the parsed part of the input (before parsing, the candidate set was the whole $\mathbb{N}$).

Note that this definition involves circular dependencies. For instance, if $\delta(X) = X$ for some $X \in N$ then $X, \tau \vdash_G s \to o$ if $X, \tau \vdash_G s \to o$ by clause 3. There is no result of parsing in such cases (not even $\bot$). We call this behaviour *divergence*.

2.2 Properties of the semantics

Ford [6] proves that parsing in PEG is unambiguous, whereby the consumed part of an input string always is its prefix. Theorem 2.1 below is an analogous result for PEG$^>$. Besides the uniqueness of the result of parsing, it states that if we only consider relations in $\Re^+(\mathbb{N})$ then the whole operation state in our setting is in a certain sense decreasing during parsing.

Denote by $\geq$ the *suffix order* of strings (i.e., $w \geq w'$ iff $w = uw'$ for some $u \in T^*$) and by $\sqsupseteq$ the *implication order* of truth values (i.e., $tt \sqsupset ff$). Denote by $\geqslant$ the pointwise order on operation states, i.e., $(w, I, b) \geqslant (w', I', b')$ iff $w \geq w'$, $I \supseteq I'$ and $b \sqsupseteq b'$.

▶ **Theorem 2.1.** *Let $G = (N, T, \delta, \mathsf{s})$ be a PEG$^>$, $\mathsf{e} \in \mathcal{E}_G$, $\tau \in \Re^+(\mathbb{N})$ and $s \in State$. Then $\mathsf{e}, \tau \vdash_G s \to o$ for at most one o, whereby $o = \top(s')$ implies $s \geqslant s'$. Also if $s = (w, I, b)$ and $s' = (w', I', b')$ then $s \neq s'$ implies both $w > w'$ and $b' = ff$, and $I \neq \varnothing$ implies $I' \neq \varnothing$.*

Proof. By induction on the shape of the derivation tree of the assertion $\mathsf{e}, \tau \vdash_G s \to o$. ◀

Theorem 2.1 enables to observe a common pattern in the semantics of indentation and alignment. Denoting by $\kappa(\mathsf{p})$ either p° or $|\mathsf{p}|$, both clauses 8 and 10 have the following form, parametrized on two mappings $\alpha, \gamma : State \to State$:

For $\mathsf{p} \in \mathcal{E}_G$, $\kappa(\mathsf{p}), \tau \vdash_G s \to o$ holds in two cases:

- If there exists a state s' such that $\mathsf{p}, \tau \vdash_G \alpha(s) \to \top(s')$ and $o = \top(s \wedge \gamma(s'))$;
- If $\mathsf{p}, \tau \vdash_G \alpha(s) \to \bot$ and $o = \bot$.

The meanings of indentation and alignment constructs are distinguished solely by α and γ. For many properties, proofs that rely on this abstract common definition can be carried out, assuming that γ is monotone, preserves the largest element and follows together with α the axiom $x \wedge \gamma(y) \leq \gamma(\alpha(x) \wedge y)$. The class of all meet semilattices L with top element, equipped with mappings α, γ satisfying these three conditions, contains identities (i.e., semilattices L with $\alpha = \gamma = \mathrm{id}_L$) and is closed under compositions (of different α, γ defined on the same semilattice L) and under direct products. If $\rho \in \Re^+(\mathbb{N})$ then the conditions hold for $\alpha_1, \gamma_1 : \wp(\mathbb{N}) \to \wp(\mathbb{N})$ with $\alpha_1(I) = \rho^{-1}(I)$, $\gamma_1(I) = \rho(I)$, similarly in the case if $\alpha_2, \gamma_2 : \mathbb{B} \to \mathbb{B}$ with $\alpha_2(b) = tt$, $\gamma_2(b) = b$. Now the direct product of the identities of T^* and $\mathbb{B}$ with (α_1, γ_1) on $\wp(\mathbb{N})$ gives the indentation case, and the direct product of the identities of T^* and $\wp(\mathbb{N})$ and the Boolean lattice $\mathbb{B}$ with (α_2, γ_2) gives the alignment case.

If α, γ satisfy the conditions then $\gamma(\alpha(x)) \geq x$ since $x = x \wedge \top = x \wedge \gamma(\top) \leq \gamma(\alpha(x) \wedge \top) = \gamma(\alpha(x))$. Adding dual conditions (α monotone, $\alpha(\bot) = \bot$ and $\alpha(x) \vee y \geq \alpha(x \vee \gamma(y))$) would make (α, γ) a Galois' connection. In our cases, the dual axioms do not hold.

2.3 Semantic equivalence

▶ **Definition 2.2.** Let $G = (N, T, \delta, \mathsf{s})$ be a PEG$^>$ and $\mathsf{p}, \mathsf{q} \in \mathcal{E}_G$. We say that p and q are *semantically equivalent* in G and denote $\mathsf{p} \sim_G \mathsf{q}$ iff $\mathsf{p}, \tau \vdash_G s \to o \iff \mathsf{q}, \tau \vdash_G s \to o$ for every $\tau \in \Re^+(\mathbb{N})$, $s \in State$ and $o \in State + 1$.

For example, one can easily prove that $\mathsf{p}\varepsilon \sim_G \mathsf{p} \sim_G \varepsilon\mathsf{p}$, $\mathsf{p}(\mathsf{qr}) \sim_G (\mathsf{pq})\mathsf{r}$, $\mathsf{p}/(\mathsf{q}/\mathsf{r}) \sim_G (\mathsf{p}/\mathsf{q})/\mathsf{r}$, $\mathsf{p}(\mathsf{q}/\mathsf{r}) \sim_G \mathsf{pq}/\mathsf{pr}$, $\mathsf{p}/\mathsf{q} \sim_G \mathsf{p}/!\mathsf{pq}$ for all $\mathsf{p}, \mathsf{q}, \mathsf{r} \in \mathcal{E}_G$ [6]. We are particularly interested in equivalences involving the additional operators of PEG$^>$. In Sect. 3, they will be useful in eliminating alignment and position operators. The following Theorem 2.3 states distributivity laws of the three new operators of PEG$^>$ w.r.t. other constructs:

▶ **Theorem 2.3.** *Let $G = (N, T, \delta, s)$ be a PEG$^{>}$. Then:*

1. $\varepsilon_\sigma \sim_G \varepsilon$, $(pq)_\sigma \sim_G p_\sigma q_\sigma$, $(p/q)_\sigma \sim_G p_\sigma / q_\sigma$, $(!p)_\sigma \sim_G !p_\sigma$, $(p^*)_\sigma \sim_G (p_\sigma)^*$, $(p^\varrho)_\sigma \sim_G (p_\sigma)^\varrho$, $\lfloor p \rfloor_\sigma \sim_G \lfloor p_\sigma \rfloor$ *for all* $\sigma \in \Re^+(\mathbb{N})$;
2. $\varepsilon^\varrho \sim_G \varepsilon$, $(p/q)^\varrho \sim_G p^\varrho / q^\varrho$, $(!p)^\varrho \sim_G !p^\varrho$, $(p_\sigma)^\varrho \sim_G (p^\varrho)_\sigma$ *for all* $\varrho \in \Re^+(\mathbb{N})$;
3. $\lfloor \varepsilon \rfloor \sim_G \varepsilon$, $\lfloor p/q \rfloor \sim_G \lfloor p \rfloor / \lfloor q \rfloor$, $\lfloor !p \rfloor \sim_G ! \lfloor p \rfloor$, $\lfloor p_\sigma \rfloor \sim_G \lfloor p \rfloor_\sigma$.

Proof. The equivalences in claim 1 hold as the token mode steadily distributes to each case of the semantics definition. Those in claims 2 and 3 have straightforward proofs using the joint form of the semantics of indentation and alignment and the axioms of α, γ. ◀

Note that indentation does not distribute with concatenation, i.e., $(pq)^\varrho \not\sim_G p^\varrho q^\varrho$. This is because $(pq)^\varrho$ assumes one indentation block with a baseline common to p and q while $p^\varrho q^\varrho$ tolerates different baselines for p and q. For example, take $p = a \in T$, $q = b \in T$, let the token mode be $\triangle$ and the input state be $(a^1 b^2, \mathbb{N}, ff)$ (recall that a^i means terminal a occurring in column i). We have $a, \triangle \vdash_G (a^1 b^2, \mathbb{N} \setminus \{0\}, ff) \to \top(b^2, \{1\}, ff)$ and $b, \triangle \vdash_G (b^2, \{1\}, ff) \to \bot$ (since $(2,1) \notin \triangle$), therefore $ab, \triangle \vdash_G (a^1 b^2, \mathbb{N} \setminus \{0\}, ff) \to \bot$ and $(ab)^{>}, \triangle \vdash_G (a^1 b^2, \mathbb{N}, ff) \to \bot$. On the other hand, $a, \triangle \vdash_G (a^1 b^2, \mathbb{N} \setminus \{0\}, ff) \to \top(b^2, \{1\}, ff)$ implies $a^{>}, \triangle \vdash_G (a^1 b^2, \mathbb{N}, ff) \to \top(b^2, \{0\}, ff)$ (since $\mathbb{N} \cap (> (\{1\})) = \{0\}$) and, analogously, $b^{>}, \triangle \vdash_G (b^2, \{0\}, ff) \to \top(\varepsilon, \{0\}, ff)$ (since $>^{-1}(\{0\}) = \mathbb{N} \setminus \{0\} \ni 2$ and $\{0\} \cap (> (\{2\})) = \{0\}$). Consequently, $a^{>} b^{>}, \triangle \vdash_G (a^1 b^2, \mathbb{N}, ff) \to \top(\varepsilon, \{0\}, ff)$.

We can however prove the following facts:

▶ **Theorem 2.4.** *Let $G = (N, T, \delta, s)$ be a PEG$^{>}$.*

1. *Identity indentation law: For all $p \in \mathcal{E}_G$, $p^\triangle \sim_G p$.*
2. *Composition law of indentations: For all $p \in \mathcal{E}_G$ and $\varrho, \sigma \in \Re^+(\mathbb{N})$, $(p^\varrho)^\sigma \sim_G p^{\sigma \circ \varrho}$.*
3. *Distributivity of indentation and alignment: For all $p \in \mathcal{E}_G$ and $\varrho \in \Re^+(\mathbb{N})$, $\lfloor p \rfloor^\varrho \sim_G \lfloor p^\varrho \rfloor$.*
4. *Idempotence of alignment: For all $p \in \mathcal{E}_G$, $\lfloor \lfloor p \rfloor \rfloor \sim_G \lfloor p \rfloor$.*
5. *Cancellation of outer token modes: For all $p \in \mathcal{E}_G$ and $\sigma, \tau \in \Re(\mathbb{N})$, $(p_\sigma)_\tau \sim_G p_\sigma$.*
6. *Terminal alignment property: For all $a \in T$, $\lfloor a \rfloor \sim_G a_\triangle$.*

Proof. Claim 1 follows easily from the semantics of indentation. By the conditions imposed on α and γ, it follows that the composition of the effects of indentations or alignments with respective mapping pairs (α_1, γ_1) and (α_2, γ_2) coincides with the effect of a prospective construct of similar kind with mapping pair $(\alpha_2 \circ \alpha_1, \gamma_1 \circ \gamma_2)$. Claims 2–4 follow directly from this observation, as the composition of structures (α, γ) used for indentation and alignment is commutative and the structure (α, γ) used for alignment is idempotent. Claim 5 is trivial. Claim 6 follows from a straightforward case study. ◀

Theorems 2.3 and 2.4 enact bringing alignments through all syntactic constructs except concatenation. Alignment does not distribute with concatenation, because in parsing of an expression of the form $\lfloor pq \rfloor$, the terminal to be aligned can be in the part of the input consumed by p or (if parsing of p succeeds with consuming no input) by q. Alignment can nevertheless be moved through concatenation if any successful parsing of the first expression in the concatenation either never consumes any input or always consumes some input:

▶ **Theorem 2.5.** *Let $G = (N, T, \delta, s)$ be a PEG$^{>}$ and $p, q \in \mathcal{E}_G$.*

1. *If $p, \tau \vdash_G s \to \top(s')$ implies $s' = s$ for all $\tau \in \Re^+(\mathbb{N})$, $s, s' \in State$, then $\lfloor pq \rfloor \sim_G \lfloor p \rfloor \lfloor q \rfloor$.*
2. *If $p, \tau \vdash_G s \to \top(s')$ implies $s' \neq s$ for all $\tau \in \Re^+(\mathbb{N})$, $s, s' \in State$, then $\lfloor pq \rfloor \sim_G \lfloor p \rfloor q$.*

Proof. Straightforward case study. ◀

Theorem 2.5 (1) holds also for indentation (instead of alignment), the same proof in terms of α, γ is valid. Finally, the following theorem states that position and indentation of terminals are equivalent if the alignment flag is false and the token mode is the identity:

▶ **Theorem 2.6.** *Let* $G = (N, T, \delta, s)$ *be a PEG*$^{>}$. *Let* $a \in T$, $\sigma \in \Re^{+}(\mathbb{N})$ *and* $w \in T^{*}$, $I \in \wp(\mathbb{N})$, $o \in State + 1$. *Then* $a_{\sigma}, \triangle \vdash_{G} (w, I, \mathit{ff}) \to o \iff a^{\sigma}, \triangle \vdash_{G} (w, I, \mathit{ff}) \to o$.

Proof. Straightforward case study. ◀

2.4 Differences of our approach from previous work

Our specification of PEG$^{>}$differs from the definition used by Adams and Ağacan [3] by three essential aspects listed below. The last two discrepancies can be understood as bugs in the original description that have been corrected in the Haskell `indentation` package by Adams [1]. This package also provides means for locally changing the token mode. All in all, our modifications fully agree with the `indentation` package.

1. The position operator p_{σ} is missing in [3]. The treatment there assumes just one default token mode applying to the whole grammar, whence token positions deviating from the default must be specified using the indentation operator. The benefits of the position operator were shortly discussed in Subsect. 2.1.
2. According to the grammar semantics provided in [3], the alignment flag is never changed at the end of parsing of an expression of the form $\lfloor p \rfloor$. This is not appropriate if p succeeds without consuming any token, as the alignment flag would unexpectedly remain true during parsing of the next token that is out of scope of the alignment operator. The value the alignment flag had before starting parsing $\lfloor p \rfloor$ should be restored in this case. This is the purpose of conjunction in the alignment semantics described in this paper.
3. In [3], an alignment is interpreted w.r.t. the indentation baseline of the block that corresponds to the parsing expression to which the alignment operator is applied. Indentation operators occurring inside this expression and processed while the alignment flag is true are neglected. In the semantics described in our paper, raising the alignment flag does not suppress new indentations. Alignments are interpreted w.r.t. the indentation baseline in force at the aligned token site. This seems more appropriate than the former approach where the indentations cancelled because of an alignment do not apply even to the subsequent non-aligned tokens. Distributivity of indentation and alignment fails in the semantics of [3]. Note that alignment of a block nevertheless suppresses the influence of position operators whose scope extend over the first token of the block.

Our grammar semantics has also two purely formal deviations from the semantics used by Adams and Ağacan [3] and Ford [6].

1. We keep track of the rest of the input in the operation state while both [3, 6] expose the consumed part of the input instead. This difference was introduced for simplicity and to achieve uniform decreasing of operation states in Theorem 2.1.
2. We do not have explicit step counts. They were used in [6] to compose proofs by induction. We provide analogous proofs by induction on the shape of derivation trees.

3 Elimination of alignment and position operators

Adams [2] describes alignment elimination in the context of CFGs. In [3], Adams and Ağacan claim that alignment elimination process for PEGs is more difficult due to the lookahead construct. To our knowledge, no concrete process of semantics-preserving alignment

elimination is described for PEGs before. We provide one below for well-formed grammars. We rely on the existence of position operators in the grammar; this is not an issue since we also show that position operators can be eliminated from alignment-free grammars.

We describe our process informally on an example; a general description together with correctness theorems can be found in our online paper [9].

As the repetition operator can always be eliminated (by adding a new non-terminal A_{p} with $\delta(A_{\mathsf{p}}) = \mathsf{p}A_{\mathsf{p}}/\varepsilon$ for each subexpression p that occurs under the star operator), we may assume that the input grammar G is repetition-free. The process also assumes that G is well-formed, all negations are applied to atomic expressions, and all choices are disjoint. A choice expression p/q is called *disjoint* if parsing of p and q cannot succeed in the same input state and token mode. *Well-formedness* is a decidable conservative approximation of the predicate that is true iff parsing in G never diverges (it definitely excludes grammars with left recursion but can exclude also some safe grammars). Well-formedness of PEGs was introduced by Ford [6]. Extending the notion to expressions containing the extra operators of PEG$^{>}$ is straightforward, details are provided in [9]. Achieving the other two preconditions can be considered as a preparatory and previously studied (e.g. in [6] as stage 1 of negation elimination) step of the process.

We will work on the example grammar $G = (N, T, \delta, \mathsf{s})$ where $N = \{A, B\}$, $T = \{a, b, c\}$, $\delta = \left\{A \mapsto !!c/a/B, B \mapsto b{\restriction}AA{\restriction}^{>}\right\}$ and $\mathsf{s} = A_{\geq}$. This grammar is well-formed and choices in the rule for A are disjoint ($!!c$, a and B can succeed only if the input string starts with c, a or b, respectively). Not all negations are in front of atoms; this can be fixed by introducing a new non-terminal C with rule $C \mapsto !c$ and replacing the rule for A with $A \mapsto !C/a/B$. Elimination of alignment and position operators from the grammar is done in 3 stages.

1. *Transform G to an equivalent grammar G_1 where for each expression of the form pq occurring in δ or s, parsing of p either never succeeds without consuming some input or can succeed only if consuming no input.*
   This "splitting" step enables to later bring alignments through concatenations (by Theorem 2.5). It only modifies rules and the start expression. The new set of rules and start expression could be

   $$\delta_1 = \left\{A \mapsto a/B \quad B \mapsto b{\restriction}AA/A!!c/!!cA/!!c!!c{\restriction}^{>}\right\}, \qquad \mathsf{s}_1 = (A/!!c)_{\geq}$$

   (in the version of the grammar with non-terminal C, there would be an additional rule for C that never succeeds). The formal process described in [9] would give a somewhat more complicated result but this simplified variant works fine and perfectly explains the ideas. The alternative with negation is removed from the rule of A to allow parsing of A succeed only if consuming some input (the same transformation would be performed on other non-terminals if it was necessary). The removed alternative (which happens to succeed only if consuming no input) is inserted into each concatenation of A, as well as into the start expression. Basically the same transformation was used by Ford [6] on stage 2 of his negation elimination process.
2. *Using the semantic equivalences of Subsect. 2.3, move all alignments down to atoms. Rewrite alignment of terminals in terms of the position operator and the identity relation. For each existing non-terminal X, introduce a new non-terminal with a rule whose right-hand side is obtained from ${\restriction}\delta_1(X){\restriction}$ by moving all alignments down to non-terminals, and replace all aligned non-terminals with the corresponding new non-terminals.*
   In our example, we have to introduce two new non-terminals A' and B' with right-hand sides obtained from ${\restriction}a/B{\restriction}$ and ${\restriction}b{\restriction}AA/A!!c/!!cA/!!c!!c{\restriction}^{>}{\restriction}$, respectively. Using that

$\lfloor AA\rfloor \sim \lfloor A\rfloor A$, $\lfloor A!!c\rfloor \sim \lfloor A\rfloor !!c$, $\lfloor !!cA\rfloor \sim !!c\lfloor\rfloor A\rfloor$ and $\lfloor !!c!!c\rfloor \sim !!c\lfloor\rfloor !!c\rfloor$, we end up with

$$\delta_2 = \left\{ \begin{array}{llll} A & \mapsto a/B & B & \mapsto b(A'A/A'!!c/!!c_\triangle A'/!!c_\triangle !!c_\triangle)^> \\ A' & \mapsto a_\triangle/B' & B' & \mapsto b_\triangle(A'A/A'!!c/!!c_\triangle A'/!!c_\triangle !!c_\triangle)^> \end{array} \right\}, \quad s_2 = (A/!!c)_\geq.$$

3. *Using the semantic equivalences of Subsect. 2.3, move all position operators down to atoms. For each non-terminal X and relation τ used by position operators, introduce a new non-terminal with a rule whose right-hand side is obtained from $(\delta_2(X))_\tau$ by moving position operators down to atoms, and replace all non-terminals under position operators with the corresponding new non-terminals. Replace position operators applied to terminals with indentation, omit identity indentations.*

   In our example, $\geq$ is the only relation used by position operators. Hence we must introduce one new non-terminal for each existing non-terminal. Denote them $\hat{A}$, $\hat{B}$, $\hat{A}'$, $\hat{B}'$. As the old non-terminals will never be used when parsing the new start expression, we can omit their rules. The rules of the new non-terminals and the start expression are

$$\delta_3 = \left\{ \begin{array}{llll} \hat{A} & \mapsto a^\geq/\hat{B} & \hat{B} & \mapsto b^\geq(\hat{A}'\hat{A}/\hat{A}'!!c^\geq/!!c\hat{A}'/!!c!!c)^> \\ \hat{A}' & \mapsto a/\hat{B}' & \hat{B}' & \mapsto b(\hat{A}'\hat{A}/\hat{A}'!!c^\geq/!!c\hat{A}'/!!c!!c)^> \end{array} \right\}, \quad s_3 = \hat{A}/!!c^\geq.$$

Note how terminals that do not have to be aligned have indentation $\geq$ while terminals to be aligned have no indentation. Parsings in the resulting grammar must run with the alignment flag unset and assume the identity token mode.

At step 1, the sizes of the right-hand sides of the rules can grow exponentially though the number of rules stays unchanged. Preprocessing the grammar via introducing new non-terminals in such a way that all concatenations were applied to atoms (similarly to Ford [6]) would hinder the growth, but the size in the worst case remains exponential. Steps 2 and 3 cause at most a linear growth of right-hand sides.

4 Which relations are good?

Speed of grammar-driven parsing of expressions that involve relations depends on the nature of the relations. The representation of the baseline candidate sets in operation states plays a particular role. Adams and Ağacan [3] prove that, during parsing of expressions that involve only relations $\triangle$, $>$, $\geq$ and $\circledast$, all intermediate sets I occurring in operation states have the form of a connected interval of natural numbers (possibly extending to infinity). This enables to represent any set I by its minimum $\min I$ and supremum $\sup I$ (supremum means maximum for finite sets and ∞ for infinite ones).

In practice, languages may require other indentation relations. Adams [2] mentions $\{(i+2, i) : i \in \mathbb{N}\}$ needed for occam, the `indentation` package [1] implements constant relations $\{(c, i) : i \in \mathbb{N}\}$. Here we generalize the result of [3] by finding a criterion for deciding which indentation relations preserve the interval form of the set of baseline candidates. The result also applies to the relations used by position operators since they matter only during parsing of terminals and the way they are used there is the same as in the case of indentation.

In this section, we denote $l_\rho(i) = \min(\rho(\{i\}))$ and $h_\rho(i) = \sup(\rho(\{i\}))$ for any $\rho \in \Re(\mathbb{N})$ and $i \in \mathbb{N}$. Functions l and h are undefined on i if $\rho(\{i\}) = \varnothing$. *Intervals* are sets of the form $\{j \in \mathbb{N} : n \leq j \leq o\}$ for any $n \in \mathbb{N}$, $o \in \mathbb{N} \cup \{\infty\}$. For uniform treatment, $\varnothing$ is also considered an interval (the case $n > o$ in the definition). This has no bad consequences as the set of baseline candidates is guaranteed to stay non-empty by Theorem 2.1.

4.1 Relations that keep indentation sets as intervals

When parsing of an expression of the form e^{ϱ} starts, it must create a new set $\varrho^{-1}(I)$ where I is the current set of indentation baseline candidates. There are two obvious conditions that must hold for $\varrho^{-1}(I)$ being an interval whenever I is:

1. For each $i \in \mathbb{N}$, $\varrho^{-1}(\{i\})$ must be an interval, as I can be a one-element set.
2. For any $i \in \mathbb{N}$, $\varrho^{-1}(\{i\}) \cup \varrho^{-1}(\{i+1\})$ must be an interval, as I can be $\{i, i+1\}$.

One can easily prove by induction on the size of I that if a relation ϱ satisfies conditions 1 and 2 then $\varrho^{-1}(I)$ is an interval for any interval I. Note that condition 2 holds iff for any two consecutive natural numbers i and j in any order, $l_{\varrho^{-1}}(i) \leq h_{\varrho^{-1}}(j) + 1$.

At the end of parsing of an expression of the form e^{ϱ}, a new set $I \cap \varrho(I')$ must be created to combine the information provided by the set I of baseline candidates for the surrounding indentation and the set I' of baseline candidates for the local indentation. Hence $I \cap \varrho(I')$ must be an interval whenever I and I' are. Taking $I = \mathbb{N}$ and $I' = \{i\}$ or $I' = \{i, i+1\}$, we see as before that $\varrho(\{i\})$ and $\varrho(\{i, i+1\})$ must be intervals for every $i \in \mathbb{N}$. Conversely, an easy induction on the number of elements in I' shows that if all sets $\varrho(\{i\})$ and $\varrho(\{i, i+1\})$ are intervals then $\varrho(I')$ is an interval for any interval I'. As the intersection of two intervals is an interval, this condition is also sufficient for $I \cap \varrho(I')$ being an interval.

To conclude, if for each used relation ϱ, all sets of the form $\varrho^{-1}(\{i\})$, $\varrho^{-1}(\{i, i+1\})$, $\varrho(\{i\})$ and $\varrho(\{i, i+1\})$ are intervals whereby $\varrho \in \Re^+(\mathbb{N})$, then all sets of baseline candidates occurring in the operation state are intervals during any parsing that starts with an interval as the baseline set. By Theorem 4.2 below, the set $\varrho(\{i, i+1\})$ can be omitted from this list, so three out of four conditions remain. For every relation $\varrho \in \Re^+(\mathbb{N})$ that fails to meet these three conditions, one can find a parsing expression e and an initial state such that a non-interval set appears during parsing. Indeed, the set $\varrho^{-1}(I)$ for an arbitrarily chosen finite interval $I = \{i, i+1, \ldots, i+k\}$ is evaluated during parsing of $\mathsf{e} = {}_{|}a(bc^{\varrho})^{\geq}{}_{|\geq}$ on an input string of the shape $a^i b^{i+k} w$, and for any $i \in \mathbb{N}$, if $\varrho(\{i\})$ is not an interval and hence contains some $n \in \mathbb{N}$ then $\varrho(\{i\})$ is evaluated during parsing of $\mathsf{e} = (ab^{\varrho})_{\triangle}$ on the input $a^n b^i$.

4.2 Implementation issues

By condition 1 at the beginning of Subsect. 4.1, any feasible relation ϱ is uniquely determined by the pair of functions $(l_{\varrho^{-1}}, h_{\varrho^{-1}})$. Similarly, ϱ is determined by $(l_{\varrho}, h_{\varrho})$ because of the analogous condition for $\varrho(\{i\})$. Functions $l_{\varrho^{-1}}$ and $h_{\varrho^{-1}}$ are total as we assume $\varrho \in \Re^+(\mathbb{N})$, while l_{ϱ} and h_{ϱ} can be partial. For the four relations considered in [3] for instance,

$$
\begin{array}{lcllcl}
(l_{\triangle^{-1}}(i), h_{\triangle^{-1}}(i)) & = & (i, i); & (l_{\triangle}(i), h_{\triangle}(i)) & = & (i, i); \\
(l_{>^{-1}}(i), h_{>^{-1}}(i)) & = & (i+1, \infty); & (l_{>}(i), h_{>}(i)) & = & (0, i-1) \text{ (provided } i > 0); \\
(l_{\geq^{-1}}(i), h_{\geq^{-1}}(i)) & = & (i, \infty); & (l_{\geq}(i), h_{\geq}(i)) & = & (0, i); \\
(l_{\circledast^{-1}}(i), h_{\circledast^{-1}}(i)) & = & (0, \infty); & (l_{\circledast}(i), h_{\circledast}(i)) & = & (0, \infty).
\end{array}
$$

We recall two well-known notions.

▶ **Definition 4.1.** 1. Call a function $f : \mathbb{N} \to \mathbb{Z} \cup \{\infty\}$ *non-decreasing* iff, for every $i, j \in \mathbb{N}$, $i \leq j$ implies $f(i) \leq f(j)$.

2. Call a function $f : \mathbb{N} \to \mathbb{Z} \cup \{\infty\}$ *weakly unimodal* iff there exists some $m \in \mathbb{N}$ such that, for every $i, j \in \mathbb{N}$, $i \leq j \leq m$ implies $f(i) \leq f(j)$ and $m \leq i \leq j$ implies $f(i) \geq f(j)$.

Unimodality of f means that the values of f are increasing until some argument m called *mode* and decreasing after that. Weakness specifies that increasing and decreasing can be non-strict (letting values at consecutive arguments equal). We will use also the corresponding

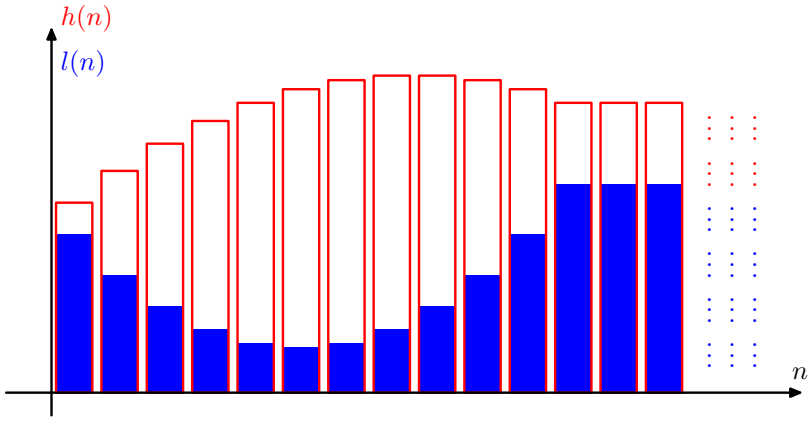


Figure 1 Functions h with the weak unimodality and l with the reverse unimodality property

reverse properties that hold for $f : \mathbb{N} \to \mathbb{Z}$ iff $-f$ has the original property. Thereby f is called *non-increasing* iff $-f$ is non-decreasing. Figure 1 depicts two functions l, h defined on $\mathbb{N}$ such that $l < h$ and both h and $-l$ are weakly unimodal (blue filled and red empty bars depict l and h, respectively).

▶ **Theorem 4.2.** *Let $\rho \in \Re^+(\mathbb{N})$ satisfy conditions 1 and 2 at the beginning of Subsect. 4.1. Then the following conditions are equivalent:*

(*) For every $i \in \mathbb{N}$, $\rho(\{i\})$ is an interval;

(**) Each of $h_{\rho^{-1}}$ and $-l_{\rho^{-1}}$ is either non-decreasing or weakly unimodal;

(***) For every $i \in \mathbb{N}$, both $\rho(\{i\})$ and $\rho(\{i, i+1\})$ are intervals.

Proof. Assume (*) and suppose that (**) does not hold. If $h_{\rho^{-1}}$ is neither non-decreasing nor weakly unimodal then there exist $i, j, j' \in \mathbb{N}$, $j + 1 < j'$ such that $i \le \min(h_{\rho^{-1}}(j), h_{\rho^{-1}}(j'))$ and, for every j'', if $j < j'' < j'$ then $h_{\rho^{-1}}(j'') < i$. By condition 2 assumed by the theorem, $l_{\rho^{-1}}(j) \le h_{\rho^{-1}}(j+1) + 1 \le i \le \min(h_{\rho^{-1}}(j), h_{\rho^{-1}}(j')) \le h_{\rho^{-1}}(j)$ and similarly $l_{\rho^{-1}}(j') \le h_{\rho^{-1}}(j'-1) + 1 \le i \le \min(h_{\rho^{-1}}(j), h_{\rho^{-1}}(j')) \le h_{\rho^{-1}}(j')$. Hence $\rho(\{i\})$ contains both j and j' but none of the numbers between j and j', which contradicts the fact that $\rho(\{i\})$ is an interval. The case with $-l_{\rho^{-1}}$ being neither non-decreasing nor weakly unimodal is handled analogously. Thus (*) implies (**).

Now assume (**). For every natural number i, denote $H_i = \{j \in \mathbb{N} : h_{\rho^{-1}}(j) \ge i\}$ and $L_i = \{j \in \mathbb{N} : l_{\rho^{-1}}(j) \le i\}$; then $\rho(\{i\}) = H_i \cap L_i$. By (**), H_i and L_i are intervals, hence $\rho(\{i\})$ is an interval for each i. Note that $i < i'$ implies $H_i \supseteq H_{i'}$ and $L_i \subseteq L_{i'}$. Moreover, $H_{i+1} \cup L_i = \mathbb{N}$: Indeed, $j \notin L_i$ implies $l_{\rho^{-1}}(j) > i$, meaning that $h_{\rho^{-1}}(j) \ge l_{\rho^{-1}}(j) \ge i + 1$ whence $j \in H_{i+1}$. Clearly $\rho(\{i, i+1\}) = \rho(\{i\}) \cup \rho(\{i+1\}) = (H_i \cap L_i) \cup (H_{i+1} \cap L_{i+1})$. To prove that $\rho(\{i, i+1\})$ is an interval, suppose that $j \in H_i \cap L_i$, $j' \in H_{i+1} \cap L_{i+1}$ and $j < j'' < j'$ (the case $j' < j'' < j$ is similar). As $H_{i+1} \subseteq H_i$, both j and j' belong to H_i. Similarly as $L_i \subseteq L_{i+1}$, both j and j' belong to L_{i+1}. Consequently, also $j'' \in H_i \cap L_{i+1}$ since both H_i and L_{i+1} are intervals. Now if $j'' \in H_{i+1}$ then $j'' \in H_{i+1} \cap L_{i+1} \subseteq \rho(\{i, i+1\})$, and if $j'' \in L_i$ then $j'' \in H_i \cap L_i \subseteq \rho(\{i, i+1\})$. Hence (**) implies (***).

Finally, (***) trivially implies (*). ◀

Knowing the modes of both $h_{\rho^{-1}}$ and $-l_{\rho^{-1}}$ (in the non-decreasing case with no upper bound, ∞ can be used as the mode), $\rho^{-1}(I)$ can be computed by $O(1)$ evaluations of $l_{\rho^{-1}}$ and $h_{\rho^{-1}}$ and $O(1)$ comparisons of natural numbers for any interval I. Indeed, $\sup(\rho^{-1}(I))$

equals the value of $h_{\varrho^{-1}}$ at one of the endpoints of I or at the mode (if the mode belongs to I), or is ∞ (if $h_{\varrho^{-1}}$ is non-decreasing and unbounded); $\min(\varrho^{-1}(I))$ can be found similarly.

The set of natural numbers where h_ϱ and l_ϱ are defined is an interval. An argument similar to the proof of part (*) $\Rightarrow$ (**) of Theorem 4.2 shows that each of h_ϱ and $-l_\varrho$ is either non-decreasing or weakly unimodal within its domain of definition. Hence by symmetry, it is possible to compute $\varrho(I')$ for any interval I' by $O(1)$ evaluations of l_ϱ and h_ϱ and $O(1)$ comparisons if the modes of both h_ϱ and $-l_\varrho$ are known. Partiality of h_ϱ and l_ϱ is not an issue since Theorem 2.1 guarantees that the set of indentation baseline candidates becomes never empty. Obviously the intersection of known intervals I and $\varrho(I')$ is computable by $O(1)$ comparisons.

Consequently, by representing relations ϱ with records that consist of functions l_ϱ, h_ϱ, $l_{\varrho^{-1}}$ and $h_{\varrho^{-1}}$ together with the modes of $-l_\varrho$, h_ϱ, $-l_{\varrho^{-1}}$ and $h_{\varrho^{-1}}$, every indentation causes only an $O(1)$ time overhead (if the values of the functions are computable in $O(1)$ time). It is reasonable to expect that the parser implementer provides the right representations for all the relations in use as the number of these relations is normally quite small.

For the four relations in [3], the modes can be defined as follows (they are not unique as the functions can increase or decrease non-strictly):

$$\begin{array}{llll} (m(-l_{\triangle^{-1}}), m(h_{\triangle^{-1}})) & = & (0,\infty); & (m(-l_{\triangle}), m(h_{\triangle})) = (0,\infty); \\ (m(-l_{>^{-1}}), m(h_{>^{-1}})) & = & (0,0); & (m(-l_{>}), m(h_{>})) = (1,\infty); \\ (m(-l_{\geq^{-1}}), m(h_{\geq^{-1}})) & = & (0,0); & (m(-l_{\geq}), m(h_{\geq})) = (0,\infty); \\ (m(-l_{\circledast^{-1}}), m(h_{\circledast^{-1}})) & = & (0,0); & (m(-l_{\circledast}), m(h_{\circledast})) = (0,0). \end{array}$$

5 Related work

PEGs were first introduced and studied by Ford [6] who also showed them to be closely related with the TS system [5] and TDPL [4], as well as to their generalized forms [5, 4].

Adams [2] and Adams and Ağacan [3] provide an excellent overview of previous approaches to describing indentation-sensitive languages and attempts of building indentation features into parser libraries. Our work is a theoretical study of the approach proposed in [3] while some details of the semantics used in our paper were "corrected" in the lines of Adams' `indentation` package for Haskell [1]. This package enables specifying indentation sensitivity within the Parsec and Trifecta parser combinator libraries. A process of alignment operator elimination is previously described for CFGs by Adams [2].

Matsumura and Kuramitsu [7] develop a very general extension of PEG that also enables to specify indentation. Their framework is powerful but complicated. The approach proposed in [3] and followed by us is in contrast with [7] by focusing on indentation and aiming to maximal simplicity and convenience of usage.

6 Conclusion

We studied the extension of PEG proposed by Adams and Ağacan [3] for indentation-sensitive parsing. This extension uses operators for marking indentation and alignment besides the classic ones. Having added one more operator (position) for convenience, we found a lot of useful semantic equivalences that are valid on expressions written in the extended grammars. We applied these equivalences subsequently for defining a process that algorithmically eliminates all alignment and position operators from well-formed grammars.

We analyzed practical limitations of the indentation extension of PEG from the aspect of efficient expressibility and computability of the relations and sets needed during parsing.

We found a wide class of relations that, provided the minimum and supremum of the set of numbers related to any given number is computable in $O(1)$ time, cause only $O(1)$ overhead at each parsing step.

Acknowledgements. I thank the anonymous reviewers whose comments helped to improve the paper considerably.

References

1 Michael D. Adams. The `indentation` package. URL: `http://hackage.haskell.org/package/indentation`.

2 Michael D. Adams. Principled parsing for indentation-sensitive languages: Revisiting Landin's offside rule. In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'13, Rome, Italy - January 23 - 25, 2013*, pages 511–522. ACM, 2013. `doi:10.1145/2429069.2429129`.

3 Michael D. Adams and Ömer S. Ağacan. Indentation-sensitive parsing for Parsec. In Wouter Swierstra, editor, *Proceedings of the 2014 ACM SIGPLAN symposium on Haskell, Gothenburg, Sweden, September 4-5, 2014*, pages 121–132. ACM, 2014. `doi:10.1145/2633357.2633369`.

4 Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

5 Alexander Birman and Jeffrey D. Ullman. Parsing algorithms with backtrack. *Information and Control*, 23(1):1–34, 1973. `doi:10.1016/S0019-9958(73)90851-6`.

6 Bryan Ford. Parsing expression grammars: A recognition-based syntactic foundation. In Neil D. Jones and Xavier Leroy, editors, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, pages 111–122. ACM, 2004. `doi:10.1145/964001.964011`.

7 Tetsuro Matsumura and Kimio Kuramitsu. A declarative extension of parsing expression grammars for recognizing most programming languages. *JIP*, 24(2):256–264, 2016. `doi:10.2197/ipsjjip.24.256`.

8 Sérgio Medeiros, Fabio Mascarenhas, and Roberto Ierusalimschy. Left recursion in parsing expression grammars. In Francisco Heron de Carvalho Junior and Luís Soares Barbosa, editors, *Programming Languages - 16th Brazilian Symposium, SBLP 2012, Natal, Brazil, September 23-28, 2012. Proceedings*, volume 7554 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2012. `doi:10.1007/978-3-642-33182-4_4`.

9 Härmel Nestra. Alignment elimination from Adams' grammars, 2017. `arXiv:1706.06497`.

The Power of Linear-Time Data Reduction for Maximum Matching*

George B. Mertzios[†1], André Nichterlein[‡2], and Rolf Niedermeier[3]

1 **School of Engineering and Computing Sciences, Durham University, UK**
george.mertzios@durham.ac.uk
2 **School of Engineering and Computing Sciences, Durham University, UK, and Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
andre.nichterlein@tu-berlin.de
3 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
rolf.niedermeier@tu-berlin.de

Abstract

Finding maximum-cardinality matchings in undirected graphs is arguably one of the most central graph primitives. For m-edge and n-vertex graphs, it is well-known to be solvable in $O(m\sqrt{n})$ time; however, for several applications this running time is still too slow. We investigate how linear-time (and almost linear-time) data reduction (used as preprocessing) can alleviate the situation. More specifically, we focus on *linear-time kernelization*. We start a deeper and systematic study both for general graphs and for bipartite graphs. Our data reduction algorithms easily comply (in form of preprocessing) with every solution strategy (exact, approximate, heuristic), thus making them attractive in various settings.

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

Keywords and phrases Maximum-cardinality matching, bipartite graphs, linear-time algorithms, kernelization, parameterized complexity analysis, FPT in P.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.46

1 Introduction

"Matching is a powerful piece of algorithmic magic" [22]. In the maximum matching problem, given an undirected graph, one has to compute a maximum set of nonoverlapping edges. Maximum matching is arguably among the most fundamental graph-algorithmic primitives allowing for a polynomial-time algorithm. More specifically, on an n-vertex and m-edge graph a maximum matching can be found in $O(m\sqrt{n})$ time [20]. Improving this upper time bound resisted decades of research. Recently, however, Duan and Pettie [9] presented a linear-time algorithm that computes a $(1-\epsilon)$-approximate maximum-weight matching, where the running time dependency on ϵ is $\epsilon^{-1}\log(\epsilon^{-1})$. For the unweighted case, the $O(m\sqrt{n})$ algorithm of Micali and Vazirani [20] implies a linear-time $(1-\epsilon)$-approximation, where in this case the running time dependency on ϵ is ϵ^{-1} [9]. We take a different route: First, we do not give up the quest for optimal solutions. Second, we focus on efficient – more specifically, linear-time executable – data reduction rules, that is, not solving an instance but significantly

* A full version with all proof details is available at https://arxiv.org/abs/1609.08879
† Partially supported by the EPSRC grant EP/P020372/1.
‡ Supported by a postdoc fellowship of DAAD while at Durham University.

shrinking its size before actually solving the problem. Doing so, however, we focus here on the unweighted case. In the context of decision problems and parameterized complexity analysis this approach is known as kernelization.

The spirit behind our approach is thus closer to the identification of efficiently solvable special cases of maximum matching. There is quite some body of work in this direction. For instance, since an augmenting path can be found in linear time [11], the standard augmenting path-based algorithm runs in $O(s(n+m))$ time, where s is the number of edges in the maximum matching. Yuster [25] developed an $O(rn^2 \log n)$-time algorithm, where r is the difference between maximum and minimum degree of the input graph. Moreover, there are linear-time algorithms for computing maximum matchings in special graph classes, including convex bipartite [23], strongly chordal [8], chordal bipartite [7], and cocomparability graphs [19].

All this and the more general spirit of "parameterization for polynomial-time solvable problems" [13] (also referred to as "FPT in P" or "FPTP" for short) forms the starting point of our research. Remarkably, Fomin et al. [10] recently developed an algorithm to compute a maximum matching in graphs of treewidth k in $O(k^4 n \log n)$ randomized time.

Following the paradigm of *kernelization*, that is, provably effective and efficient data reduction, we provide a systematic exploration of the power of not only polynomial-time but actually linear-time data reduction for maximum matching. Thus, our aim (fitting within FPTP) is to devise problem kernels that are computable in linear time. In other words, the fundamental question we pose is whether there is a very efficient preprocessing that provably shrinks the input instance, where the effectiveness is measured by employing some parameters. The philosophy behind this is that if we can design linear-time data reduction algorithms, then we may employ them for free before afterwards employing any super-linear-time solving algorithm. We believe that this sort of question deserves deeper investigation and we initiate it based on the matching problem.

As kernelization is defined for decision problems, we use in the remainder of the paper the *decision version* of maximum matching. In a nutshell, a kernelization of a decision problem instance is an algorithm that produces an equivalent instance whose size can solely be upper-bounded by a function in the parameter (preferably a polynomial). The focus on decision problems is justified by the fact that all our results, although formulated for the decision version, in a straightforward way extend to the corresponding optimization version.

(Maximum-Cardinality) Matching
Input: An undirected graph $G = (V, E)$ and a nonnegative integer s.
Question: Is there a size s subset $M_G \subseteq E$ of nonoverlapping (i.e. disjoint) edges?

Note that for any polynomial-time solvable problem solving the given instance and returning a trivial yes- or no-instance always produces a constant-size kernel in polynomial time. Hence, we are looking for kernelization algorithms that are *faster* than the algorithms solving the problem. The best we usually can hope for is linear time. For NP-hard problems, each polynomial-time kernelization algorithm is faster than any solution algorithm, unless P = NP. While the focus of classical kernelization for NP-hard problems is mostly on improving the *size* of the kernel, we particularly emphasize that for polynomially solvable problems it now becomes mandatory to also focus on the *running time* of the kernelization algorithm. Indeed, we consider linear-time kernelization as the holy grail and this drives our research when studying kernelization for Matching.

Our contributions. We present three kernels for Matching (see Table 1 for an overview). All our parameterizations can be categorized as "distance to triviality" [6, 14, 18, 24]. They are motivated as follows. First, note that it is important that the parameters we exploit can

Table 1 Our kernelization results.

Parameter k	running time	kernel size	
Results for Matching			
Feedback edge number	$O(n+m)$	$O(k)$ vertices and edges	(Theorem 3)
Feedback vertex number	$O(kn)$	$2^{O(k)}$ vertices and edges	(Theorem 11)
Results for Bipartite Matching			
Distance to chain graphs	$O(n+m)$	$O(k^3)$ vertices	(Theorem 15)

be computed, or well approximated (within constant factors), in linear time regardless of the parameter value. For instance, it is not known whether this is possible for treewidth. Next, note that maximum-cardinality matchings can be trivially found in linear time on trees (or forests). So we consider the edge deletion distance (feedback edge number) and vertex deletion distance (feedback vertex number) to forests. Notably, there is a trivial linear-time algorithm for computing the feedback edge number and there is a linear-time factor-4 approximation algorithm for the feedback vertex number [1]. We mention in passing that the parameter vertex cover number, which is lower-bounded by the feedback vertex number, has been frequently studied for kernelization [3, 4]. In particular, Gupta and Peng [15] and Giannopoulou et al. [13] provided a linear-time computable quadratic-size kernel for Matching with respect to the parameter solution size (or equivalently vertex cover number). Coming to bipartite graphs, we parameterize by the vertex deletion distance to chain graphs which is motivated as follows. First, chain graphs form one of the most obvious easy cases for bipartite graphs where Matching can be solved in linear time [23]. Second, we show that the vertex deletion distance of any bipartite graph to a chain graph can be 2-approximated in linear time. Moreover, vertex deletion distance to chain graphs lower-bounds the vertex cover number of a bipartite graph, and thus gives a stronger parameterization [18] than vertex cover number.

An overview of our main results is given in Table 1. We study kernelization for Matching parameterized by the feedback vertex number, that is, the vertex deletion distance to a forest (see Section 2). As a warm-up we first show that a subset of our data reduction rules for the "feedback vertex set kernel" also yields a linear-time computable linear-size kernel for the typically much larger parameter feedback edge number (see Section 2.1). As for Bipartite Matching no faster algorithm is known than on general graphs, we kernelize Bipartite Matching with respect to the vertex deletion distance to chain graphs (see Section 3).

Seen from a high level, our two technical main results (Theorems 11 and 15, see Table 1) employ the same algorithmic strategy, namely upper-bounding (as a function of the parameter) the number of neighbors in the appropriate vertex deletion set X; that is, X being the feedback vertex set or in the deletion set to chain graphs, respectively. To achieve this we develop new *"irrelevant edge techniques"* tailored to these two kernelization problems. More specifically, whenever a vertex v of the deletion set X has large degree, we efficiently detect edges incident to v whose removal does not change the size of the maximum matching. Then the remaining graph can be further shrunk by scenario-specific data reduction rules. While this approach of removing irrelevant edges is natural, the technical details and the proofs of correctness become quite technical and combinatorially challenging.

Note that there exists a trivial $O(km)$-time solving (not only kernelization) algorithm, where k is the feedback vertex number. Our kernel has size $2^{O(k)}$. Therefore, only if $k = o(\log n)$ our kernelization algorithm *provably* shrinks the initial instance. However, our result

is still relevant: First, our data reduction rules might assist in proving a polynomial upper bound on the kernel size – so our result is a first step in this direction. Second, the running time $O(kn)$ of our kernelization algorithm is a kind of "half way" between $O(km)$ (which could be as bad as $O(k^2n)$) and $O(n+m)$ (which is best possible). Finally, note that this work focuses on theoretical and worst-case analysis; in practice, our kernelization algorithm might achieve much better upper bounds on real-world input instances.

As a technical side remark, we emphasize that in order to achieve a linear-time kernelization algorithm, we often need to use suitable data structures and to carefully design the appropriate data reduction rules to be exhaustively applicable in linear time, making this form of "algorithm engineering" much more relevant than in the classical setting of mere polynomial-time data reduction rules.

Notation and Observations. We use standard notation from graph theory. Merging two vertices u and v means to first introduce a new vertex w with $N(w) = N(u) \cup N(v)$ and then delete u and v. A feedback vertex (edge) set of a graph G is a set X of vertices (edges) such that $G - X$ is a tree or a forest. The feedback vertex (edge) number denotes the size of a minimum feedback vertex (edge) set. All paths we consider are simple paths. Two paths in a graph are called *internally vertex-disjoint* if they are either completely vertex-disjoint or they overlap only in their endpoints. A *matching* in a graph is a set of pairwise disjoint edges. Let $G = (V, E)$ be a graph and let $M \subseteq E$ be a matching in G. The degree of a vertex is denoted by $\deg(v)$. A vertex $v \in V$ is called *matched* with respect to M if there is an edge in M containing v, otherwise v is called *free* with respect to M. If the matching M is clear from the context, then we omit "with respect to M". An *alternating path* with respect to M is a path in G such that every second edge of the path is in M. An *augmenting path* is an alternating path whose endpoints are free. It is well known that a matching M is maximum if and only if there is no augmenting path for it. Let $M \subseteq E$ and $M' \subseteq E$ be two matchings in G. We denote by $G(M, M') := (V, M \triangle M')$ the graph containing only the edges in the symmetric difference of M and M', that is, $M \triangle M' := M \cup M' \setminus (M \cap M')$. Observe that every vertex in $G(M, M')$ has degree at most two.

▶ **Observation 1.** *Let $G = (V, E)$ be a graph with a maximum matching M_G, let $X \subseteq V$ be a vertex subset of size k, and let M_{G-X} be a maximum matching for $G - X$. Then, $|M_{G-X}| \leq |M_G| \leq |M_{G-X}| + k$.*

Kernelization. A *parameterized problem* is a set of instances (I, k) where $I \in \Sigma^*$ for a finite alphabet Σ, and $k \in \mathbb{N}$ is the *parameter*. We say that two instances (I, k) and (I', k') of parameterized problems P and P' are *equivalent* if (I, k) is a yes-instance for P if and only if (I', k') is a yes-instance for P'. A *kernelization* is an algorithm that, given an instance (I, k) of a parameterized problem P, computes in polynomial time an equivalent instance (I', k') of P (the *kernel*) such that $|I'| + k' \leq f(k)$ for some computable function f. We say that f measures the *size* of the kernel, and if $f(k) \in k^{O(1)}$, we say that P admits a polynomial kernel. Often, a kernel is achieved by applying polynomial-time executable data reduction rules. We call a data reduction rule $\mathcal{R}$ *correct* if the new instance (I', k') that results from applying $\mathcal{R}$ to (I, k) is equivalent to (I, k). An instance is called *reduced* with respect to some data reduction rule if further application of this rule has no effect on the instance.

2 Kernelization for Matching on General Graphs

In this section we first present as a warm-up a simple, linear-size kernel for Matching with respect to the parameter feedback edge number (see Section 2.1). Exploiting the data reduction rules and ideas used for this kernel, we then present the main result of this section: an exponential-size kernel for the typically much smaller parameter feedback vertex number (see Section 2.2).

2.1 Warm-up: Parameter feedback edge number

We provide a linear-time computable linear-size kernel for Matching parameterized by the feedback edge number, that is, the size of a minimum feedback edge set. Observe that a minimum feedback edge set can be computed in linear time via a simple depth-first search or breadth-first search. The kernel is based on the next two simple data reduction rules due to Karp and Sipser [17]. They deal with vertices of degree at most two.

▶ **Reduction Rule 2.1.** *Let $v \in V$. If $\deg(v) = 0$, then delete v. If $\deg(v) = 1$, then delete v and its neighbor and decrease the solution size s by one (v is matched with its neighbor).*

▶ **Reduction Rule 2.2.** *Let v be a vertex of degree two and let u, w be its neighbors. Then remove v, merge u and w, and decrease the solution size s by one.*

Reduction Rules 2.1 and 2.2 are correct; however, it is not clear whether Reduction Rule 2.2 can be exhaustively applied in linear time. Fortunately, for our purpose it suffices to consider the following restricted version which we can exhaustively apply in linear time.

▶ **Reduction Rule 2.3.** *Let v be a vertex of degree two and u, w be its neighbors with u and w having degree at most two. Then remove v, merge u and w, and decrease s by one.*

▶ **Lemma 2.** *Reduction Rules 2.1 and 2.3 can be exhaustively applied in $O(n + m)$ time.*

▶ **Theorem 3.** Matching *admits a linear-time computable linear-size kernel with respect to the parameter feedback edge number k.*

Proof. Apply Reduction Rules 2.1 and 2.3 exhaustively in linear time (Lemma 2). We claim that the reduced graph $G = (V, E)$ has less than $12k$ vertices and less than $13k$ edges. Denote with $X \subseteq E$ a feedback edge set for G, $|X| \leq k$. Furthermore, denote with V^1_{G-X}, V^2_{G-X}, and $V^{\geq 3}_{G-X}$ the vertices that have degree one, two, and more than two in the $G - X$. Thus, $|V^1_{G-X}| \leq 2k$ as each leaf in $G - X$ has to be incident to an edge in X. Next, since $G - X$ is a forest (or tree), we have $|V^{\geq 3}_{G-X}| < |V^1_{G-X}|$ and thus $|V^{\geq 3}_{G-X}| < 2k$. Finally, each degree-two vertex in G needs at least one neighbor of degree at least three since G is reduced with respect to Reduction Rule 2.3. Thus, the vertices in V^2_{G-X} are either incident to an edge in X or adjacent to one of the at most $|V^{\geq 3}_{G-X}| + 2k$ vertices in G that have degree at least three. Since the sum over all degrees of vertices in $V^{\geq 3}_{G-X}$ is at most $\sum_{v \in V^{\geq 3}_{G-X}} \deg_{G-X}(v) \leq 2|V^{\geq 3}_{G-X}| + |V^1_{G-X}| < 6k$, it follows that $|V^2_{G-X}| \leq 8k$. Thus, the number of vertices in G is $|V^1_{G-X}| + |V^2_{G-X}| + |V^{\geq 3}_{G-X}| \leq 12k$. Since $G - X$ is a forest, it follows that G has at most $|V| + k \leq 13k$ edges. ◀

Applying the $O(m\sqrt{n})$-time algorithm for Matching [20] on the kernel yields:

▶ **Corollary 4.** Matching *can be solved in $O(n + m + k^{1.5})$ time, where k is the feedback edge number.*

2.2 Parameter feedback vertex number

We next provide for MATCHING a kernel of size $2^{O(k)}$ computable in $O(kn)$ time where k is the feedback vertex number. Using a known linear-time factor 4-approximation algorithm [1], we can approximate feedback vertex set and use it in our kernelization algorithm.

Roughly speaking, our kernelization algorithm extends the linear-time computable kernel with respect to the parameter feedback edge set. Thus, Reduction Rules 2.1 and 2.3 play an important role in the kernelization. Compared to the other kernels presented in this paper, the kernel presented here comes at the price of higher running time $O(kn)$ and bigger kernel size (exponential size). It remains open whether MATCHING parameterized by the feedback vertex number admits a linear-time computable kernel (possibly of exponential size), and whether it admits a polynomial kernel computable in $O(kn)$ time.

Subsequently, we describe our kernelization algorithm which keeps in the kernel all vertices in the given feedback vertex set X and shrinks the size of $G - X$. Before doing so, we need some further notation. In this section, we assume that each tree is rooted at some arbitrary (but fixed) vertex such that we can refer to the parent and children of a vertex. A leaf in $G - X$ is called a *bottommost leaf* either if it has no siblings or if all its siblings are also leaves. (Here, bottommost refers to the subtree with the root being the parent of the considered leaf.) The outline of the algorithm is as follows (we assume throughout that $k < \log n$ since otherwise the input instance is already a kernel of size $O(2^k)$):

1. Reduce G with respect to Reduction Rules 2.1 and 2.3.
2. Compute a maximum matching M_{G-X} in $G - X$.
3. Modify M_{G-X} in linear time such that only the leaves of $G - X$ are free.
4. Bound the number of free leaves in $G - X$ by k^2.
5. Bound the number of bottommost leaves in $G - X$ by $O(k^2 2^k)$.
6. Bound the degree of each vertex in X by $O(k^2 2^k)$. Then, use Reduction Rules 2.1 and 2.3 to provide the kernel of size $2^{O(k)}$.

Whenever we reduce the graph at some step, we also show that the applied data reduction is *correct*. That is, the given instance is a yes-instance if and only if the reduced one is a yes-instance. The correctness of our kernelization algorithm then follows by the correctness of each step. We discuss in the following some details of each step.

2.2.1 Steps 1 to 3

By Lemma 2 we can perform Step 1 in linear time. A maximum matching in Step 2 can be computed by repeatedly matching a free leaf to its neighbor and by removing both vertices from the graph (thus effectively applying Reduction Rule 2.1 to $G - X$). By Lemma 2, this can be done in linear time. Step 3 can be done in $O(n)$ time by traversing each tree in M_{G-X} in a BFS manner starting from the root: If a visited inner vertex v is free, then observe that all children are matched since M_{G-X} is maximum. Pick an arbitrary child u of v and match it with v. The vertex w that was previously matched to u is now free and since it is a child of u, it will be visited in the future. Observe that Steps 2 and 3 do not change the graph but only the auxiliary matching M_{G-X}, and thus these steps are correct.

2.2.2 Step 4

Recall that our goal is to upper-bound the number of edges between vertices of X and $V \setminus X$, since we can then use a simple analysis as for the parameter feedback edge set. Observe that if a vertex $x \in X$ has at least k neighbors in $V \setminus X$ that are free wrt. M_{G-X}, then there exists a maximum matching where x is matched to one of these k vertices since at most $k - 1$

can be "blocked" by other matching edges. This means that we can delete all other edges incident to x. Formalizing this idea, we obtain the following data reduction rule.

▶ **Reduction Rule 2.4.** *Let $G = (V, E)$ be a graph, let $X \subseteq V$ be a subset of size k, and let M_{G-X} be a maximum matching for $G - X$. If there is a vertex $x \in X$ with at least k free neighbors $V_x = \{v_1, \ldots, v_k\} \subseteq V \setminus X$, then delete all edges from x to vertices in $V \setminus V_x$.*

To finish Step 4, we exhaustively apply Reduction Rule 2.4 in linear time. Afterwards, there are at most k^2 free (wrt. to M_{G-X}) leaves in $G - X$ that have at least one neighbor in X since each of the k vertices in X is adjacent to at most k free leaves. Thus, applying Reduction Rule 2.1 we can remove the remaining free leaves that have no neighbor in X. However, since for each degree-one vertex also its neighbor is removed, we might create new free leaves and need to again apply Reduction Rule 2.4 and update the matching (see Step 3). This process of alternating application of Reduction Rules 2.1 and 2.4 stops after at most k rounds since the neighborhood of each vertex in X can be changed by Reduction Rule 2.4 at most once. This shows the running time $O(k(n+m))$. We next show how to improve this to $O(n+m)$ time and arrive at the final lemma of this subsection.

▶ **Lemma 5.** *Given a matching instance (G, s) and a feedback vertex set X, one can compute in linear time an instance (G', s') with feedback vertex set X and a maximum matching $M_{G'-X}$ in $G' - X$ such that the following holds.*

- *There is a matching of size s in G if and only if there is a matching of size s' in G'.*
- *Each vertex that is free wrt. $M_{G'-X}$ is a leaf in $G' - X$.*
- *There are at most k^2 free leaves in $G' - X$.*

2.2.3 Step 5

Step 5 reduces the graph in $O(kn)$ time so that at most $k^2(2^k + 1)$ bottommost leaves will remain in the forest $G - X$. We restrict ourselves to consider leaves that are matched with their parent vertex in M_{G-X} and that do not have a sibling. Any sibling of a bottommost leaf is by definition also a leaf. Thus, at most one of these leaves (the bottommost leaf or its siblings) is matched with respect to M_{G-X} and all other leaves are free. Recall that in the previous step we upper-bounded the number of free leaves with respect to M_{G-X} by k^2. Hence there are at most k^2 bottommost leaves with siblings.

Our general strategy for this step is to extend the idea behind Reduction Rule 2.4: We want to keep for each pair of vertices $x, y \in X$ at most k different internally vertex-disjoint augmenting paths from x to y. (For ease of notation we keep k paths although keeping $k/2$ is sufficient.) In this step, we only consider augmenting paths of the form x, u, v, y where v is a bottommost leaf and u is v's parent in $G - X$. Assume that the parent u of v is adjacent to some vertex $x \in X$. Observe that in this case any augmenting path starting with the two vertices x and u has to continue to v and end in a neighbor of v. Thus, the edge $\{x, u\}$ can be only used in augmenting paths of length three. Furthermore, all these length-three augmenting paths are clearly internally vertex-disjoint. If we do not need the edge $\{x, u\}$ because we kept k augmenting paths from x already, then we can delete $\{x, u\}$. Furthermore, if we deleted the last edge from u to X (or u had no neighbors in X in the beginning), then u is a degree-two vertex in G and can be removed by applying Reduction Rule 2.2. As the child v of u is a leaf in $G - X$, it follows that v has at most $k + 1$ neighbors in G. Thus, an application of Reduction Rule 2.2 to remove u takes $O(k)$ time.

Counting for each pair $x \in N(u) \cap X$ and $y \in N(v) \cap X$ one augmenting path gives in a simple worst-case analysis $O(k^2)$ time per edge; this is too slow for our purposes. Instead, we

count for each vertex $x \in N(u) \cap X$ and for each *set* $Y = N(v) \cap X$ one augmenting path. In this way, we know that for each $y \in Y$ there is one augmenting path from x to y, without iterating through all $y \in Y$. We get an exponential factor in the bound of the bottommost leaves since there are 2^k subsets of X, but we can perform this step in $O(kn)$ time as follows.

▶ **Lemma 6.** *Let $(G = (V, E), s)$ be a matching instance, let $X \subseteq V$ be a feedback vertex set, and let M_{G-X} be a maximum matching for $G - X$ with at most k^2 free vertices in $G - X$ that are all leaves. Then, can compute in $O(kn)$ time an instance (G', s') with feedback vertex set X and a maximum matching $M_{G'-X}$ in $G' - X$ such that the following holds.*

- *There is a matching of size s in G if and only if there is a matching of size s' in G'.*
- *There are at most $k^2(2^k + 1)$ bottommost leaves in $G' - X$.*
- *There are at most k^2 free vertices in $G' - X$ and they are all leaves.*

2.2.4 Step 6

In this subsection, we provide the final step of our kernelization algorithm. Recall that in the previous steps we have upper-bounded the number of bottommost leaves in $G - X$ by $O(k^2 2^k)$, we computed a maximum matching M_{G-X} for $G - X$ such that at most k^2 vertices are free wrt. M_{G-X} and all free vertices are leaves in $G - X$. Using this, we next show how to reduce G to a graph of size $O(k^3 2^k)$. To this end we need some further notation. A leaf in $G - X$ that is not bottommost is called a *pendant*. We define T to be the *pendant-free tree (forest) of* $G - X$, that is, the tree (forest) obtained from $G - X$ by removing all pendants. The next observation shows that $G - X$ is not much larger than T. Together with the second observation, this allows us to restrict ourselves in the following on giving an upper bound on the size of T.

▶ **Observation 7.** *Let $G - X$ be as described above with vertex set $V \setminus X$ and let T be the pendant-free tree (forest) of $G - X$ with vertex set V_T. Then, $|V \setminus X| \leq 2|V_T| + k^2$.*

▶ **Observation 8.** *Let F be a forest, let F' be the pendant-free forest of F, and let B be the set of all bottommost leaves in F. Then, the set of leaves in F' is exactly B.*

From Observation 8 it follows that the set B of bottommost leaves in $G - X$ is exactly the set of leaves in T. In the previous step we reduced the graph such that $|B| \leq k^2(2^k + 1)$. Thus, T has at most $k^2(2^k + 1)$ vertices of degree one and, since T is a tree (a forest), T also has at most $k^2(2^k + 1)$ vertices of degree at least three. Let V_T^2 be the vertices of degree two in T and let $V_T^{\neq 2}$ be the remaining vertices in T. From the above it follows that $|V_T^{\neq 2}| \leq 2k^2(2^k + 1)$. Hence, it remains to bound the size of V_T^2. To this end, we will upper-bound the degree of each vertex in X by $O(k^2 2^k)$ and then use Reduction Rules 2.1 and 2.3. We will check for each edge $\{x, v\} \in E$ with $x \in X$ and $V \setminus X$ whether we "need" it. This check will use the idea from the previous subsection where each vertex in X needs to reach each subset $Y \in X$ at most k times via an augmenting path. Similarly as in the previous section, we want to keep "enough" of these augmenting paths. However, this time the augmenting paths might be long, while different augmenting paths might overlap. To still use the basic approach, we use the following lemma stating that we can still somehow replace augmenting paths.

▶ **Lemma 9.** *Let M_{G-X} be a maximum matching in the forest $G - X$. Let P_{uv} be an augmenting path for M_{G-X} in G from u to v. Let P_{wx}, P_{wy}, and P_{wz} be three internally vertex-disjoint augmenting paths from w to x, y, and z, respectively, such that P_{uv} intersects all of them. Then, there exist two vertex-disjoint augmenting paths with endpoints u, v, w, and one of the three vertices x, y, and z.*

Proof. Label the vertices in P_{uv} alternating as *odd* or *even* with respect to P_{uv} so that no two consecutive vertices have the same label, u is *odd*, and v is *even*. Analogously, label the vertices in P_{wx}, P_{wy}, and P_{wz} as odd and even with respect to P_{wx}, P_{wy}, and P_{wz} respectively so that w is always odd. Since all these paths are augmenting, it follows that each edge from an even vertex to its succeeding odd vertex is in the matching M_{G-X} and each edge from an odd vertex to its succeeding even vertex is not in the matching. Observe that P_{uv} intersects each of the other paths at least at two consecutive vertices, since every second edge must be an edge in M_{G-X}. Since $G-X$ is a forest and all vertices in X are free with respect to M_{G-X}, it follows that the intersection of two augmenting paths is connected and thus a path. Since P_{uv} intersects the three augmenting paths from w, it follows that at least two of these paths, say P_{wx} and P_{wy}, have a "fitting parity", that is, in the intersections of P_{uv} with P_{wx} and with P_{wy} the even vertices with respect to P_{uv} are either even or odd with respect to *both* P_{wx} and P_{wy}.

Assume w.l.o.g. that in the intersections of the paths the vertices have the same label with respect to the three paths (if the labels differ, then revert the ordering of the vertices in P_{uv}, that is, exchange the names of u and v and change all labels on P_{uv} to its opposite). Denote with v_s^1 and v_t^1 the first and the last vertex in the intersection of P_{uv} and P_{wx}. Analogously, denote with v_s^2 and v_t^2 the first and the last vertex in the intersection of P_{uv} and P_{wy}. Assume w.l.o.g. that P_{uv} intersects first with P_{wx} and then with P_{wy}. Observe that v_s^1 and v_s^2 are even vertices and v_t^1 and v_t^2 are odd vertices since the intersections have to start and end with edges in M_{G-X}. For an arbitrary path P and for two arbitrary vertices p_1, p_2 of P, denote by $p_1 - P - p_2$ the subpath of P from p_1 to p_2. Observe that $u - P_{uv} - v_t^1 - P_{wx} - x$ and $w - P_{wy} - v_t^2 - P_{uv} - v$ are vertex-disjoint augmenting paths. ◀

Algorithm description. We now provide the algorithm for Step 6 (see Algorithm 1 for a pseudocode). Algorithm 1 uses a table Tab which has an entry for each vertex $x \in X$ and each set $Y \subseteq X$. The table is filled in such a way that the algorithm detected for each $y \in Y$ at least $\mathrm{Tab}[x, Y]$ internally vertex-disjoint augmenting paths from x to y. The main part of the algorithm is the boolean function 'Keep-Edge' in Lines 13 to 22 which makes the decision on whether to delete an edge $\{x, v\}$ for $v \in V \setminus X$ and $x \in X$. The function works as follows for edge $\{x, v\}$: Starting at v the graph will be explored along possible augmenting paths until a "reason" for keeping the edge $\{x, v\}$ is found or further exploration is possible.

If the vertex v is free wrt. M_{G-X}, then $\{x, v\}$ is an augmenting path and we keep $\{x, v\}$ (see Line 14). Observe that in Step 4 we upper-bounded the number of free vertices by k^2 and all these vertices are leaves. Thus, we keep a bounded number of edges incident to x because the corresponding augmenting paths can end at a free leaf. We provide the exact bound below when discussing the size of the graph returned by Algorithm 1. In Line 14, the algorithm stops exploring the graph and keeps the edge $\{x, v\}$ if v has degree at least three in T. The reason is to keep the graph exploration simple by following only paths in T. This ensures that the running time for exploring the graph from x does not exceed $O(n)$. Since the number of vertices in T with degree at least three is bounded (see discussion after Observation 8), it follows that only a bounded number of such edges $\{x, v\}$ are kept.

If v is not free wrt. M_{G-X}, then it is matched with some vertex w. If w is adjacent to some leaf u in $G-X$ that is free wrt. M_{G-X}, then the path x, v, w, u is an augmenting path. Thus, the algorithm keeps in this case the edge $\{x, v\}$, see Line 16. Again, since the number of free leaves is bounded, only a bounded number of edges incident to x will be kept. If w has degree at least three in T, then the algorithm stops the graph exploration here and keeps the edge $\{x, v\}$, see Line 16. Again, this is to keep the running time at $O(kn)$ overall.

Algorithm 1: Algorithm for Step 6 of our kernelization.

Input: A matching instance $(G = (V, E), s)$, a feedback vertex set $X \subseteq V$ of size k for G with $k < \log n$ and at most $k^2(2^k + 1)$ bottommost leaves in $G - X$, and a maximum matching M_{G-X} for $G - X$ with at most k^2 free vertices in $G - X$ that are all leaves.

Output: An equivalent matching instance (G', s') such that G' contains at most $O(k^3 2^k)$ vertices and edges.

1 Fix an arbitrary bijection $f\colon 2^X \to \{1, \dots, 2^k\}$
2 **foreach** $v \in V \setminus X$ **do**
3 Set $f_X(v) \leftarrow f(N(v) \cap X)$ // The number $f_X(v) < n$ can be read in constant time.
4 Initialize a table Tab of size $k \cdot 2^k$ with $\mathrm{Tab}[x, f(Y)] \leftarrow 0$ for $x \in X, \emptyset \subsetneq Y \subseteq X$
5 $T \leftarrow$ pendant-free tree (forest) of $G - X$
6 $V_T^{\geq 3} \leftarrow$ vertices in T with degree ≥ 3
7 **foreach** $x \in X$ **do**
8 **foreach** $v \in N(x) \setminus X$ **do**
9 **if** *Keep-Edge*$(x, v) =$ *false* **then** // Is $\{x, v\}$ needed for an augmenting path?
10 delete $\{x, v\}$
11 exhaustively apply Reduction Rules 2.1 and 2.3
12 **return** (G, s).
13 **Function** *Keep-Edge*$(x \in X, v \in V \setminus X)$
14 **if** *v is free wrt. M_{G-X} or $v \in V_T^{\geq 3}$* **then return** *true*
15 $w \leftarrow$ matched neighbor of v in M_{G-X}
16 **if** *$w \in V_T^{\geq 3}$ or w is adjacent to free leaf in $G - X$* **then return** *true*
17 **if** *w has at least one neighbor in X and* $\mathrm{Tab}[x, f_X(w)] < 6k^2$ **then**
18 $\mathrm{Tab}[x, f_X(w)] \leftarrow \mathrm{Tab}[x, f_X(w)] + 1$
19 **return** *true*
20 **foreach** *neighbor $u \neq v$ of w that is matched wrt. M_{G-X} and fulfills $\{u, x\} \notin E$* **do**
21 **if** *Keep-Edge*$(u, x) =$ *true* **then return** *true*
22 **return** *false*

Let $Y \subseteq X$ denote the neighborhood of w in X. The partial augmenting path x, v, w can be extended to each vertex in Y. Thus, if the algorithm did not yet find $6k^2$ paths from x to vertices whose neighborhood in X is also Y, then the table entry $\mathrm{Tab}[x, f_X(w)]$ (where $f_X(w)$ encodes the set $Y = N(w) \cap X$) is increased by one and the edge $\{x, v\}$ will be kept (see Lines 18 and 19). The proof that $6k^2$ paths suffice is based on an exchange argument using Lemma 9. If the algorithm already found $6k^2$ "augmenting paths" from x to Y, then the neighborhood of w in X is irrelevant for x and the algorithm continues.

In Line 20, all above discussed cases to keep the edge $\{x, v\}$ do not apply and the algorithm extends the partial augmenting part x, v, w by considering the neighbors of w except v. Since the algorithm dealt with possible extensions to vertices in X in Lines 17 to 19 and with extensions to free vertices in $G - X$ in Line 14, it follows that the next vertex on this path has to be a vertex u that is matched wrt. M_{G-X}. Furthermore, since we want to extend a partial augmenting path from x, we require that u is not adjacent to x as otherwise x, u would be another, shorter partial augmenting path from x to u and we do not need the currently stored partial augmenting path.

The next lemma shows that Algorithm 1 is correct and runs in $O(kn)$ time.

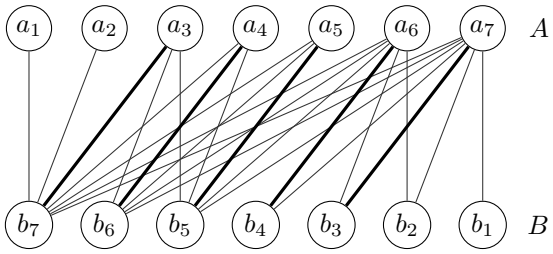


Figure 1 A chain graph. Note that the ordering of the vertices in A is going from left to right while the ordering of the vertices in B is going from right to left. The reason for these two orderings being drawn in different directions is that a maximum matching can be drawn as parallel edges, see e. g. the bold edges.

▶ **Lemma 10.** *Let $(G = (V, E), s)$ be a matching instance, let $X \subseteq V$ be a feedback vertex set of size k with $k < \log n$ and at most $k^2(2^k + 1)$ bottommost leaves in $G - X$, and let M_{G-X} be a maximum matching for $G - X$ with at most k^2 free vertices in $G - X$ that are all leaves. Then, Algorithm 1 computes in $O(kn)$ time an equivalent instance (G', s') of size $O(k^3 2^k)$.*

Simply performing Steps 1 to 6 yields the following kernel.

▶ **Theorem 11.** Matching *parameterized by the* feedback vertex number k *admits a kernel of size $2^{O(k)}$. It can be computed in $O(kn)$ time.*

Applying the $O(m\sqrt{n})$-time algorithm for Matching [20] on the kernel yields:

▶ **Corollary 12.** Matching *can be solved in $O(kn + 2^{O(k)})$ time, where k is the feedback vertex number.*

3 Kernelization for Matching on Bipartite Graphs

In this section, we investigate the possibility of efficient and effective preprocessing for Bipartite Matching. In particular, we show a linear-time computable polynomial-size kernel with respect to the parameter distance k to chain graphs. In the first part of this section, we provide the definition of chain graphs and describe how to compute the parameter. In the second part, we discuss the kernelization algorithm.

Definition and computation of the parameter. We first define chain graphs and we show that we can 4-approximate the parameter set in linear time.

▶ **Definition 13** ([5])**.** Let $G = (A, B, E)$ be a bipartite graph. Then G is a *chain graph* if each of its two color classes A, B admits a linear order w.r.t. neighborhood inclusion.

▶ **Lemma 14.** *There is a linear-time factor-4 approximation for the problem of deleting a minimum number of vertices in a bipartite graph in order to obtain a chain graph.*

Kernelization. Due to lack of space, we defer the details of our kernelization algorithm to the full version and provide in the following a high-level overview. In contrast to the kernelization in the previous section, here it is easy to bound the number of neighbors of each vertex in the deletion set X but complicated to shrink the remaining graph. Let $G = (A, B, E)$ be the given bipartite graph and let X a vertex subset such that $G - X$ is a chain graph. First compute a maximum matching M_{G-X} in $G - X$ where the edges in M_{G-X} are "parallel to each other", see Figure 1 for an illustration. Using Observation 1, we obtain the following.

▶ **Reduction Rule 3.1.** *If $|M_{G-X}| \geq s$, then return a trivial yes-instance; if $s > |M_{G-X}| + k$, then return a trivial no-instance.*

The next step of our kernelization algorithm is to bound the degree of each vertex in X. It suffices to keep for each $x \in X$ its k neighbors with smallest degree in A and in B, respectively. Due the small degree, such a vertex v is either free or matched to high-degree vertex u, see Figure 1. The correctness proof in the latter case uses the observation that there are a lot of possibilities to continue an augmenting path x, v, u as u has high degree. The reason for keeping k neighbors for each $x \in X$ is that at most $k-1$ neighbors might be "matched" (either directly or via an augmenting path) to other vertices in X.

After having bounded the degree of the vertices, we can show that for each vertex v in $G - X$ that is adjacent to a vertex in X we need to keep at most k neighbors right of v and k neighbors left of v (with respect to the linear ordering in each color class). Proving that this is indeed correct is the most technical part and relies heavily on the facts that $G - X$ is a chain graph and that the edges in M_{G-X} are "parallel".

Since for each of the k vertices in X we keep at most $2k$ neighbors in $G - X$, and for each of these neighbors, we keep $2k$ vertices, we arrive at the following.

▶ **Theorem 15.** Matching *on bipartite graphs admits a linear-time computable cubic-vertex kernel with respect to the vertex deletion distance to chain graphs.*

Applying the $O(n^{2.5})$-time algorithm for Bipartite Matching [16] on the kernel yields:

▶ **Corollary 16.** Matching *can be solved in $O(k^{7.5} + n + m)$ time, where k is the vertex deletion distance to chain graphs.*

Using the randomized $O(n^\omega)$-time bipartite matching algorithm based on matrix multiplication [21], one would obtain a randomized algorithm with running time $O(k^{3\omega} + n + m)$, where $\omega < 2.373$ is the matrix multiplication exponent.

4 Conclusion

We focused on kernelization results for Matching. In ongoing work, we are testing the practical relevance of our data reduction rules. There remain numerous challenges for future research as discussed in the second part of this concluding section. First, however, let us discuss the closely related issue of FPTP algorithms for Matching. There is a generic augmenting path-based approach to provide FPTP algorithms for Matching: One can find an augmenting path in linear time [2, 12, 20]. So the solving FPTP algorithm for Matching parameterized by some vertex deletion distance k works as follows:

1. Use a constant-factor linear-time (approximation) algorithm to compute a vertex set X such that $G - X$ is a "trivial" graph (where Matching is linear-time solvable).
2. Compute in linear time an initial maximum matching M in $G - X$.
3. Start with M as an *initial* matching in G and increase its size at most $|X| = k$ times to obtain in $O(k \cdot (n + m))$ time a *maximum* matching for G.

From this we can directly derive that Matching can be solved in $O(k(n+m))$ time, where k is one of the following parameters: feedback vertex number, feedback edge number, vertex cover number. Moreover, Bipartite Matching can be solved in $O(k(n+m))$ time, where k is the vertex deletion distance to chain graphs. Using our kernelization results, the multiplicative dependence of the running time on parameter k can now be made an additive one. For instance, in this way the running time for Bipartite Matching parameterized by vertex deletion distance to chain graphs "improves" from $O(k(n + m))$ to $O(k^{7.5} + n + m)$.

We conclude with listing some questions and tasks for future research. Can the running time of the kernelization with respect to feedback vertex set (see Section 2) be improved to linear time? Moreover, can the exponential upper bound on the kernel size be decreased to a polynomial upper bound? Is there a linear-time computable kernel for Matching parameterized by the treewidth t (assuming that t is given)? This would complement the recent randomized $O(t^4 n \log n)$ time algorithm [10]. Can one extend the kernel of Section 3 from Bipartite Matching to Matching parameterized by the distance to chain graphs?

References

1. Reuven Bar-Yehuda, Dan Geiger, Joseph Naor, and Ron M. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. *SIAM Journal on Computing*, 27(4):942–959, 1998. doi:10.1137/S0097539796305109.
2. Norbert Blum. A new approach to maximum matching in general graphs. In *Proceedings of the 17th International Colloquium on Automata, Languages, and Programming (ICALP '90)*, volume 443 of *LNCS*, pages 586–597. Springer, 1990. doi:10.1007/BFb0032060.
3. Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Preprocessing for treewidth: A combinatorial analysis through kernelization. *SIAM Journal on Discrete Mathematics*, 27(4):2108–2142, 2013. doi:10.1137/120903518.
4. Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014. doi:10.1137/120880240.
5. Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: a Survey*, volume 3 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 1999.
6. Leizhen Cai. Parameterized complexity of Vertex Colouring. *Discrete Applied Mathematics*, 127(1):415–429, 2003. doi:10.1016/S0166-218X(02)00242-1.
7. Maw-Shang Chang. Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs. In *Proceedings of the 7th International Symposium on Algorithms and Computation (ISAAC '96)*, volume 1178 of *LNCS*, pages 146–155. Springer, 1996. doi:10.1007/BFb0009490.
8. Elias Dahlhaus and Marek Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. *Discrete Applied Mathematics*, 84(1–3):79–91, 1998. doi:10.1016/S0166-218X(98)00006-7.
9. Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM*, 61(1):1:1–1:23, 2014. doi:10.1145/2529989.
10. Fedor V. Fomin, Daniel Lokshtanov, Michal Pilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '17)*, pages 1419–1432. SIAM, 2017. doi:10.1137/1.9781611974782.92.
11. Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–221, 1985. doi:10.1016/0022-0000(85)90014-5.
12. Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for general graph-matching problems. *Journal of the ACM*, 38(4):815–853, 1991. doi:10.1145/115234.115366.
13. Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical Computer Science*, 2017. Available online. doi:10.1016/j.tcs.2017.05.017.

14 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC '04)*, volume 3162 of *LNCS*, pages 162–173. Springer, 2004. doi:10.1007/978-3-540-28639-4_15.

15 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS '13)*, pages 548–557. IEEE, 2013. doi:10.1109/FOCS.2013.65.

16 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.

17 Richard M. Karp and Michael Sipser. Maximum matchings in sparse random graphs. In *Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '81)*, pages 364–375. IEEE, 1981. doi:10.1109/SFCS.1981.21.

18 Christian Komusiewicz and Rolf Niedermeier. New races in parameterized algorithmics. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS '12)*, volume 7464 of *LNCS*, pages 19–30. Springer, 2012. doi:10.1007/978-3-642-32589-2_2.

19 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. Linear-time algorithm for maximum-cardinality matching on cocomparability graphs. *CoRR*, abs/1703.05598, 2017.

20 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS '80)*, pages 17–27. IEEE, 1980. doi:10.1109/SFCS.1980.12.

21 Marcin Mucha and Piotr Sankowski. Maximum matchings via Gaussian elimination. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS '04)*, pages 248–255. IEEE, 2004. doi:10.1109/FOCS.2004.40.

22 Steven S. Skiena. *The Algorithm Design Manual*. Springer, 2010.

23 G. Steiner and J. S. Yeomans. A linear time algorithm for maximum matchings in convex bipartite graphs. *Comput. Math. Appl.*, 31:91–96, 1996. doi:10.1016/0898-1221(96)00079-X.

24 Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI '03)*, pages 1173–1178. Morgan Kaufmann, 2003.

25 Raphael Yuster. Maximum matching in regular and almost regular graphs. *Algorithmica*, 66(1):87–92, 2013. doi:10.1007/s00453-012-9625-7.

Two-Planar Graphs Are Quasiplanar

Michael Hoffmann[1] and Csaba D. Tóth*[2]

1 **Department of Computer Science, ETH Zürich, CH-8092 Zürich, Switzerland**
`hoffmann@inf.ethz.ch`
2 **Department of Mathematics, Cal. State Northridge, Los Angeles, CA, and Department of Computer Science, Tufts University, Medford, MA, USA**
`csaba.toth@csun.edu`

Abstract

It is shown that every 2-planar graph is quasiplanar, that is, if a simple graph admits a drawing in the plane such that every edge is crossed at most twice, then it also admits a drawing in which no three edges pairwise cross. We further show that quasiplanarity is witnessed by a simple topological drawing, that is, any two edges cross at most once and adjacent edges do not cross.

1998 ACM Subject Classification F.2.2 Geometrical problems, G.2.2 Graph algorithms

Keywords and phrases graph drawing, near-planar graph, simple topological plane graph

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.47

1 Introduction

For $k \in \mathbb{N}$, a graph $G = (V, E)$ is called *k-planar* if it admits a drawing in the plane such that every edge is crossed at most k times (such a drawing is called a *k-plane* drawing of G). Similarly, G is called *k-quasiplanar* if it admits a drawing in which no k edges pairwise cross each other (a *k-quasiplane* drawing). A planar graph is 0-planar and 2-quasiplanar by definition. A 3-quasiplanar graph is also called *quasiplanar*, for short. The relation between k-planarity and ℓ-quasiplanarity has been studied only recently. Angelini et al. [6] proved that for $k \geq 3$, every k-planar graph is $(k+1)$-quasiplanar. However, the case $k = 2$ was left open. In this note, we show that the result extends to $k = 2$, and prove the following.

▶ **Theorem 1.** *Every 2-planar graph is quasiplanar.*

The inclusion is proper because there exists a family of (simple) quasiplanar graphs on n vertices with $6.5n - O(1)$ edges [3], whereas every 2-planar graph on $n \geq 3$ vertices has at most $5n - 10$ edges [21]. Our proof is constructive, and allows transforming a 2-plane drawing of an n-vertex graph into a quasiplane drawing in time polynomial in n.

Simple topological drawings. The concept of k-planarity and k-quasiplanarity assumes that the drawings are *topological graphs* where the edges are represented by Jordan arcs, edges may cross each other multiple times, and adjacent edges may cross. In a *simple topological graph*, any two edges cross at most once, and no two adjacent edges cross. Excluding the crossings between adjacent edges is a nontrivial condition [14]. For example, Brandenburg et al. [8] showed that every graph that admits a 1-plane simple topological drawing also admits a 1-plane *straight-line drawing* in which crossing edges meet at a right angle.

* Research by Tóth was supported in part by the NSF awards CFF-1422311 and CFF-1423615.

Angelini et al. [6] proved that for $k \geq 3$, every k-planar graph admits a $(k+1)$-quasiplane simple topological drawing. A careful analysis of our redrawing algorithm, which transforms a 2-plane drawing of a graph into a quasiplane drawing, reveals that it produces a quasiplane simple topological drawing. Thereby we obtain the following strengthening of Theorem 1.

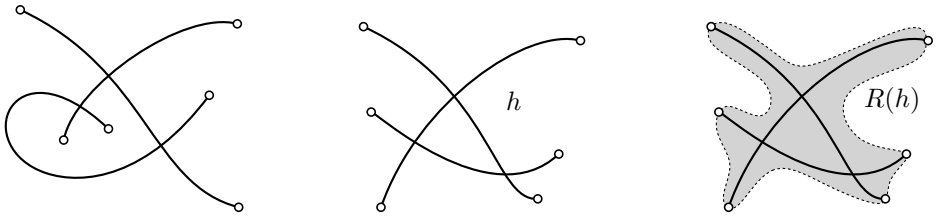
▶ **Theorem 2.** *Every 2-planar graph admits a quasiplane simple topological drawing.*

Related work. Graph planarity is a fundamental concept and a plethora of results has been obtained for planar graphs. The quest for generalizations has motivated the graph minor theory [17]. In the same vein, various notions of *near-planarity* have been studied [19]. The proximity of a graph to planarity may be measured by global parameters, such as the crossing number [22] or graph thickness and their variations [9, 10], or local parameters such as the minimum $k \in \mathbb{N}_0$ for which the graph is k-planar or k-quasiplanar. The concept of *k-planarity* plays a crucial role in proving the current best constants for the classic *Crossing Lemma* [2, 5, 18], and k-quasiplanarity is closely related to Ramsey-type properties of the intersection graph of Jordan arcs in the plane [4]. However, relations between the latter two graph classes have been studied only recently [6].

***k*-planarity.** Planar and 1-planar graphs are fairly well-understood [16]. The Crossing Lemma implies that a k-planar graph on n vertices has at most $4.1\sqrt{k} \cdot n$ edges, and this bound is the best possible apart from constant factors [21]. Tight upper bounds of $4n - 8$, $5n - 10$, and $5.5n - 11$ edges are known for $k = 1$, 2, and 3, respectively [18, 21], and an upper bound of $6n - 12$ edges is known for $k = 4$ [2]. For $k = 1, 2, 3$, so-called *optimal* k-planar graphs (which have the maximum number of edges on n vertices) have recently been completely characterized [7], however they have special properties that in general are not shared by edge-maximal k-planar graphs.

***k*-quasiplanarity.** Pach, Shahrokhi, and Szegedy [20] conjectured that for every $k \in \mathbb{N}$, an n-vertex k-quasiplanar graph has $O(n)$ edges, where the constant of proportionality depends on k. The conjecture has been verified for $k \leq 4$ [1]. The current best upper bound that holds for all $k \in \mathbb{N}$ is $n(\log n)^{O(\log k)}$ due to Fox and Pach [11]. Improvements are known in several important special cases. Suk and Walczak [23] prove that every n-vertex k-quasiplanar graph has $O(2^{\alpha(n)^c} n \log n)$ edges, where $\alpha(n)$ denotes the inverse Ackermann function and c depends only on k, if any two edges intersect in $O(1)$ points. They also show that every n-vertex k-quasiplanar graph has at most $O(n \log n)$ edges if every two edges intersect at most once. These bounds improve earlier work by Fox et al. [12, 13].

Organization. We prove Theorems 1 and 2 in Section 2: We describe a redrawing algorithm in Section 2.1, parameterized by two functions, f and g, that are defined on pairwise crossing triples of edges. In Section 2.3 we analyze local configurations that may produce a triple of pairwise crossing edges after redrawing. In Sections 2.4 and 2.5, we choose suitable functions f and g, and show that our rerouting algorithm with these parameters produces a quasiplane drawing for a 2-planar graph. In Section 2.6, we extend the analysis of our redrawing algorithm and show that it produces a simple topological quasiplane drawing. We conclude in Section 3 with a review of open problems. Due to space limitations, many proofs are omitted; they can be found in the full paper [15].



(a) tangled **(b)** untangled **(c)** region

Figure 1 Tangled and untangled 3-crossings and their associated regions.

2 Proof of Theorem 1

Let $G = (V, E)$ be a 2-planar graph. Assume without loss of generality that G is connected. We need to show that G admits a quasiplane drawing. Note that this quasiplane drawing to be constructed need not—and in general will not—be 2-plane. We may assume, without loss of generality, that G is edge-maximal, in the sense that no new edge can be added (to the abstract graph) without violating 2-planarity. Since G is 2-planar, it admits a 2-plane drawing. We show that it also admits a simple topological 2-plane drawing.

▶ **Lemma 3.** *Every 2-planar graph admits a 2-plane simple topological drawing. Specifically, a 2-plane drawing of a graph G with the minimum number of crossings (among all 2-plane drawings of G) is a simple topological graph.*

Note that a 2-plane drawing may contain a **3-crossing**, that is, a triple of pairwise crossing edges. A 3-crossing in a drawing is **untangled** if the six endpoints of the edges lie on the same face of the arrangement formed by the three edges; otherwise the 3-crossing is **tangled**, see Figure 1a and 1b for an example. Angelini et al. showed [6, Lemma 2] that every 2-planar graph admits a 2-plane drawing in which every 3-crossing is untangled. Their proof starts from a 2-plane drawing and rearranges tangled 3-crossings without introducing any new edge crossings. Therefore, in combination with our Lemma 3 we may start from a 2-plane drawing D of G with the following properties: (i) every 3-crossing is untangled, (ii) no two edges cross more than once, and (iii) no two adjacent edges cross.

If there is no 3-crossing in D, then G is quasiplanar by definition. Otherwise we construct a quasiplane drawing D' of G as described below.

Every 3-crossing in D spans a (topological) **hexagon** in the following sense. Let $\mathcal{H}$ be the set of unordered triples of edges in E that form a 3-crossing in D. In every triple $h \in \mathcal{H}$, each edge crosses both other edges of the triple, and so it cannot cross any edge in $E \setminus h$. Consequently, the triples in $\mathcal{H}$ are pairwise disjoint [6, Observation 1]. For each triple $h \in \mathcal{H}$, let $\mathrm{V}(h)$ denote the set that consists of the six endpoints of the three edges in h. Since h is untangled in D, all six vertices of $\mathrm{V}(h)$ lie on a face f_h of the arrangement induced by the edges of h as drawn in D. Any two vertices of $\mathrm{V}(h)$ that are consecutive along the boundary of f_h can be connected by a Jordan arc that closely follows the boundary of f_h and does not cross any edges in D; see Figure 1c. Together these arcs form a closed Jordan curve, which partitions the plane into two closed regions: let $R(h)$ denote the region that contains the edges of h, and let $\partial R(h)$ denote the boundary of $R(h)$. We think of $\partial R(h)$ as both a closed Jordan curve and as a graph that is a 6-cycle. As the triples in $\mathcal{H}$ are pairwise disjoint, we may assume that the regions $R(h)$, $h \in \mathcal{H}$, have pairwise disjoint interiors.

▶ **Observation 4.** *For every $h \in \mathcal{H}$, every pair of consecutive vertices of the 6-cycle $\partial R(h)$ are connected by an edge in G, and this edge is crossing-free in D.*

Proof. Let $u, v \in V$ be two consecutive vertices of a 6-cycle $\partial R(h)$ for some $h \in \mathcal{H}$.

We show that uv is an edge in G. Indeed, if uv is not an edge of G, then we can augment G with the edge $e = uv$, and insert it into the drawing D as a crossing-free Jordan arc along $\partial R(h)$ to obtain a 2-plane drawing D' of $G \cup \{e\}$. This contradicts our assumption that G is edge-maximal and no edge can be added to G without violating 2-planarity.

We then show that e is crossing free in D. Indeed, if e crosses any other edge in D, we can redraw e as a Jordan arc along $\partial R(h)$, which is crossing-free. The resulting drawing D' of G is 2-plane and has fewer crossings than D. This contradicts our assumption that D has a minimum number of crossings among all 2-plane drawings of G. ◀

By Observation 4 any two consecutive vertices along $\partial R(h)$ of a hexagon $h \in \mathcal{H}$ are connected by an edge e in G. Note that this does not necessarily imply that e is drawn along $\partial R(h)$ in D. It is possible that the cycle formed by the edge e in D and the copy of e drawn along $\partial R(h)$ (which is not part of D) contains other parts of the graph.

▶ **Observation 5.** *(a) Two distinct hexagons in $\mathcal{H}$ share at most five vertices; and (b) three distinct hexagons in $\mathcal{H}$ share at most two vertices.*

Angelini et al. proved [6, Lemma 3 and 4] that there exists an injective map $f : \mathcal{H} \to V$ that maps every hexagon $h \in \mathcal{H}$ to a vertex $v \in \mathrm{V}(h)$. For each hexagon $h \in \mathcal{H}$, exactly one edge in h is incident to the vertex $f(h)$. Let $g(h)$ be one of the two edges in h not incident to $f(h)$. Then for any such choice $g : \mathcal{H} \to E$ is an injective function (because the triples in $\mathcal{H}$ are pairwise disjoint). We complete the construction using a rerouting algorithm that for each hexagon $h \in \mathcal{H}$, reroutes the edge $g(h)$ "around" the vertex $f(h)$. The algorithm—described in detail below—is very similar to the one of Angelini et al., but with a few subtle changes to make it work for 2-planar graphs, rather than k-planar graphs, for $k \geq 3$.

2.1 Rerouting algorithm

We are given a 2-planar graph $G = (V, E)$, and a 2-plane drawing D of G with properties (i)–(iii), as described above. Let the functions $f : \mathcal{H} \to V$ and $g : \mathcal{H} \to E$ be given. (We will determine suitable choices for f and g later.) The algorithm consists of two phases.

Phase 1. For each hexagon $h \in \mathcal{H}$, we perform the following changes in D. Let $h = \{a, b, c\}$ such that the edge a is incident to $f(h)$ and $b = g(h) = uv$, where u is adjacent to $f(h)$ along $\partial R(h)$. Keep the original drawing of the edges a and c. Then arrange (possibly redraw) the edge b inside $R(h)$ so that the oriented Jordan arc uv crosses a before c. Finally, redraw the edge $b = g(h) = uv$ to go around vertex $f(h)$ as follows. See Figure 2.

1. Erase the portion of b in a small neighborhood of the crossing $a \cap b$ to split b into two Jordan arcs: an arc γ_v from v to a point x close to $a \cap b$, and another arc γ_u from x to u.
2. Keep γ_v as part of the new arc representing b, but discard γ_u and replace it by a new Jordan arc from x to u. This arc first closely follows the edge a towards $f(h)$, then goes around the endpoint $f(h)$ of a until it reaches the edge $f(h)u$ (which exists by Observation 4 and is crossing-free in D). The arc then closely follows the edge $f(h)u$ without crossing it to reach u.

As a result, edges a and b no longer cross and the 3-crossing induced by h is eliminated. However, the rerouting may create new crossings between $g(h)$ and edges incident to $f(h)$ (but not a and $uf(h)$). These new crossings are of no consequence, unless they create a 3-crossing. Hence we have to analyze under which circumstances 3-crossings can arise as a

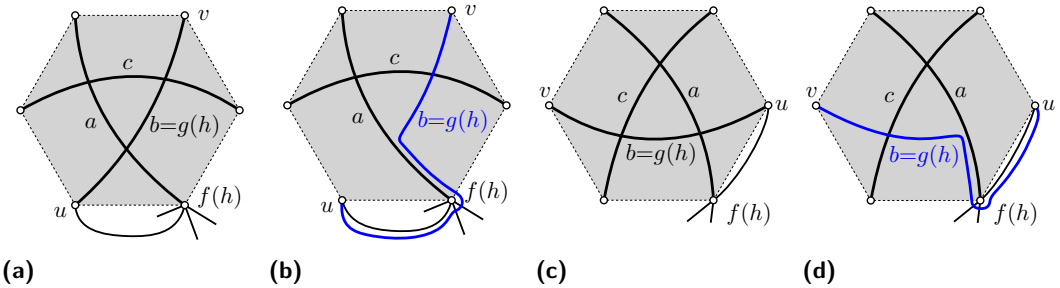


Figure 2 Rerouting $g(h)$ around $f(h)$; $g(h)$ can be either of the two edges not incident to $f(h)$.

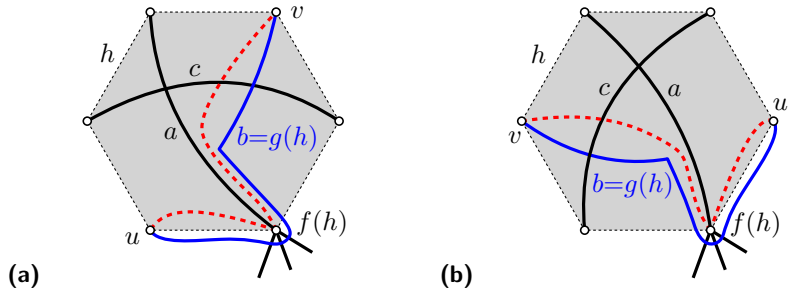


Figure 3 The hexagon h is a home for the two edges that are shown by a dashed red arc. These edges (if present in G) can be safely drawn inside $R(h)$.

result of the reroutings. But first we eliminate some potentially troublesome edge crossings in a second phase of the algorithm.

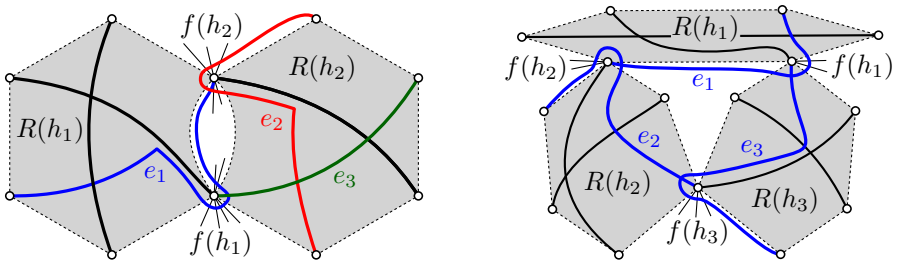
For an edge $e \in E$ a hexagon $h \in \mathcal{H}$ is a **home** for e if e is both incident to $f(h)$ and adjacent to $g(h)$. If h is a home for e, then e can be drawn inside $R(h)$ so that it has at most one crossing, with the edge $c \in h$ (see Figure 3).

Phase 2. As long as there exists an edge $e \in E$ so that (1) e has a home $h \in \mathcal{H}$, (2) there is no home $h' \in \mathcal{H} \setminus \{h\}$ of e so that e is drawn inside $R(h')$, and (3) e has at least one crossing in the current drawing, we reroute e to be drawn inside $R(h)$.

Note that each $h \in \mathcal{H}$ is a home for at most two edges and conversely an edge can have at most two homes (one for each endpoint because f is injective). Also note that an edge may be rerouted in both Phase 1 and Phase 2. This completes the description of the rerouting algorithm. Let $D(f, g)$ denote the drawing that results from applying both phases of the rerouting algorithm to the original drawing D of G.

2.2 Properties of $D(f, g)$

The edges of G fall into three groups, depending on how they are represented in $D(f, g)$ with respect to D: (1) **nonrerouted** edges have not been rerouted in either phase and remain the same as in D; (2) edges that have been rerouted in Phase 2 we call **safe** (regardless of whether or not they have also been rerouted in Phase 1); and (3) edges that have been rerouted in Phase 1 but not in Phase 2 we call **critical**. An edge is **rerouted** if it is either safe or critical. Let us start by classifying the *new* crossings that are introduced by the rerouting algorithm. Without loss of generality we may assume that in every hexagon $h \in \mathcal{H}$



(a) twin **(b)** fan

Figure 4 The redrawing may produce 3-crossings in form of twins or fans.

of D the edge $g(h)$ intersects the other two edges of h in the order described in the first paragraph of Phase 1 above. (If not, then redraw the edge $g(h)$ within $R(h)$ accordingly.)

▶ **Lemma 6.** *Consider a crossing c of two edges e_1 and e_2 in $D(f,g)$ that is not a crossing in D. After possibly exchanging the roles of e_1 and e_2, the crossing c is of exactly one of the following two types: (a) e_1 is safe and drawn in $R(h)$ for a home $h \in \mathcal{H}$ with $e_2 \in h$ nonrerouted; or (b) e_1 is critical and rerouted around an endpoint of e_2.*

▶ **Lemma 7.** *Consider a safe edge e in $D(f,g)$, and let $h \in \mathcal{H}$ denote the home of e so that $e = f(h)z$, for $z \in \mathrm{V}(h)$, is drawn inside $R(h)$. Then*

(i) *e is not part of a 3-crossing;*

(ii) *e does not cross any edge more than once; and*

(iii) *e crosses an adjacent edge e' only if e' is critical, incident to $f(h)$, and rerouted around $z = f(h')$, for some hexagon $h' \in \mathcal{H} \setminus \{h\}$, with $g(h') = e'$.*

▶ **Lemma 8.** *No two adjacent critical edges cross in $D(f,g)$.*

We are ready to completely characterize the 3-crossings in $D(f,g)$. The characterization allows us to then eliminate these 3-crossings by selecting the functions f and g suitably.

▶ **Definition 9.** Let $D(f,g)$ be a drawing of a graph $G = (V,E)$ with functions $f : \mathcal{H} \to V$ and $g : \mathcal{H} \to E$ as defined above. Three edges $e_1, e_2, e_3 \in E$ form a ...

- **twin** configuration in $D(f,g)$ if they are in two distinct hexagons $h_1, h_2 \in \mathcal{H}$, where $e_1 = g(h_1)$, $e_2 = g(h_2)$ and $e_3 \in h_2 \setminus \{e_2\}$, such that edge e_1 is incident to $f(h_2)$, edge e_3 is incident to $f(h_1)$ but not to $f(h_2)$, and e_3 is drawn inside $R(h_2)$. See Figure 4a.
- **fan** configuration in $D(f,g)$ if they are in three pairwise distinct hexagons $h_1, h_2, h_3 \in \mathcal{H}$, where $e_1 = g(h_1)$, $e_2 = g(h_2)$, and $e_3 = g(h_3)$, such that edge e_1 is incident to $f(h_2)$, edge e_2 is incident to $f(h_3)$, and edge e_3 is incident to $f(h_1)$. See Figure 4b.

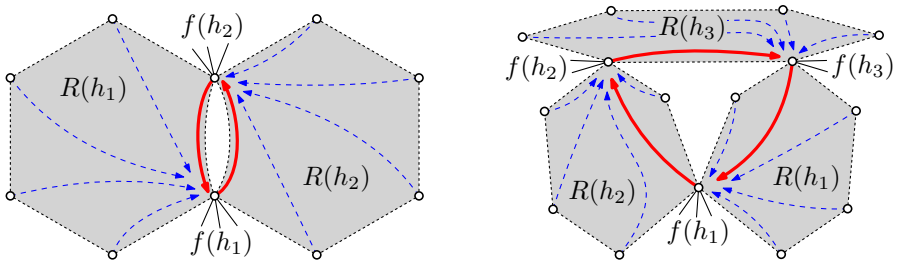
▶ **Lemma 10.** *Every 3-crossing in $D(f,g)$ forms a twin or a fan configuration.*

Theorem 1 is an immediate corollary of the following lemma, which we prove in Section 2.5.

▶ **Lemma 11.** *There exist functions $f : \mathcal{H} \to V$ and $g : \mathcal{H} \to E$ for which $D(f,g)$ is a quasiplane drawing of G.*

2.3 Conflict digraph

We define a plane digraph $K = (V,A)$ that represents the interactions between the hexagons in $\mathcal{H}$. The conflict graph depends on G, on the initial drawing D, and on the function



(a) twin **(b)** fan

Figure 5 Twin and fan configurations induce cycles in the conflict graph.

$f : \mathcal{H} \to V$, but it does not depend on the function g. For every hexagon $h \in \mathcal{H}$, we create five directed edges that are all directed towards $f(h)$ and drawn inside $R(h)$. These edges start from the five vertices on $\partial R(h)$ other than $f(h)$; see Figure 5. Note that two vertices in V may be connected by two edges with opposite orientations lying in two different hexagons (for instance, in a twin configuration as shown in Figure 5a). However, K contains neither loops nor parallel edges with the same orientation because f is injective and so every vertex can have incoming edges from at most one hexagon.

▶ **Observation 12.** *Let K be the conflict graph for $G = (V, E)$ and the drawing $D(f, g)$.*

(i) *K is a directed plane graph.*

(ii) *At every vertex $v \in V$, the incoming edges in K are consecutive in the rotation order of incident edges around v.*

(iii) *If $e_1 = v_1v_2$, $e_2 = v_2v_3$, and $e_3 = v_3v_1$ form a fan configuration in $D(f, g)$, then the conflict digraph contains a 3-cycle (v_1, v_2, v_3).*

(iv) *If $e_1 = g(h_1)$, $e_2 = g(h_2)$, and $e_3 \in h_2$ form a twin configuration in $D(f, g)$, then the conflict digraph contains a 2-cycle $(f(h_1), f(h_2))$.*

Proof. (i) The edges of K lie in the regions $R(h)$, $h \in \mathcal{H}$. Since these regions are interior-disjoint, edges from different regions do not cross. All edges in the same region $R(h)$, $h \in \mathcal{H}$, are incident to $f(h)$; so they do not cross, either. (ii) For each vertex $v \in V$, there is at most one $h \in \mathcal{H}$ such that $v = f(h)$. All incoming edges of v lie in the region $R(h)$, and all edges lying in $R(h)$ are directed towards $v = f(h)$ by construction. (iii–iv) Both claims follow directly from the definition of fan and twin configurations and the definition of K. ◀

Relations between cycles in K. We observed that K is a plane digraph, where every twin configuration induces a 2-cycle and every fan configuration induces a 3-cycle. So in order to control the appearance of twin and fan configurations in the drawing $D(f, g)$, we need to understand the structure of 2- and 3-cycles in the conflict digraph K. In the following paragraphs we introduce some terminology and prove some structural statements about cycles in K.

For a cycle c in K, let $\mathsf{int}(c)$ denote the **interior** of c, let $\mathsf{ext}(c)$ denote the **exterior** of c, let $R(c)$ denote the closed bounded **region** bounded by c, and let $\mathrm{V}(c)$ denote the **vertex set** of c. We use the notation $i \oplus 1 := 1 + (i \bmod k)$ and $i \ominus 1 := 1 + ((k + i - 2) \bmod k)$ to denote successors and predecessors, respectively, in a circular sequence of length k that is indexed $1, \ldots, k$. Let c_1 and c_2 be two cycles in the conflict graph K. We say that c_1 and c_2 are **interior-disjoint** if $\mathsf{int}(c_1) \cap \mathsf{int}(c_2) = \emptyset$. We say that c_1 **contains** c_2 if $R(c_2) \subseteq R(c_1)$. See Figure 6a for an example. In both cases, c_1 and c_2 may share vertices and edges, but they may also be vertex-disjoint.

(a) A smooth 3-cycle contains a smooth 2-cycle. **(b)** A nonsmooth 3-cycle.

Figure 6 Examples: smooth cycles and containment.

▶ **Lemma 13.** *If a vertex $v \in V$ is incident to two interior-disjoint cycles in K, then these cycles have opposite orientations (clockwise vs. counterclockwise). Consequently, every vertex $v \in V$ is incident to at most two interior-disjoint cycles in K.*

Ghosts. A cycle in the conflict digraph K is **short** if it has length two or three. We say that a 3-cycle in K is a **ghost** if two of its vertices induce a 2-cycle in K. Let $\mathcal{C}$ be the set of all short cycles in K that are not ghosts. Intuitively, we do not worry about a ghost cycle c so much. It will turn out later that by taking care of the 2-cycle c' that makes c a ghost, we also take care of c at the same time.

▶ **Lemma 14.** *A short cycle in K is uniquely determined by its vertex set.*

▶ **Lemma 15.** *Let $c_1, c_2 \in \mathcal{C}$. If $\mathrm{V}(c_1) \cap \mathsf{int}(c_2) \neq \emptyset$, then c_2 contains c_1.*

Proof. Suppose to the contrary that there exist short cycles $c_1, c_2 \in \mathcal{C}$ such that $v_1 \in \mathrm{V}(c_1) \cap \mathsf{int}(c_2)$ but c_2 does not contain c_1. Then some point along c_1 lies in $\mathsf{ext}(c_2)$. Since K is a plane graph, an entire edge of c_1 must lie in $\mathsf{ext}(c_2)$. Denote this edge by (v_2, v_3). Recall that c_1 is short (that is, it has at most three vertices), consequently, $c_1 = (v_1, v_2, v_3)$. Since c_1 has points in both $\mathsf{int}(c_2)$ and $\mathsf{ext}(c_2)$, the two cycles intersect in at least two points. In a plane graph, the intersection of two cycles consists of vertices and edges. Consequently $\mathrm{V}(c_1) \cap \mathrm{V}(c_2) = \{v_2, v_3\}$. Recall that c_2 is also short, and so it has a directed edge between any two of its vertices. However, (v_2, v_3) lies in $\mathsf{ext}(c_2)$, so the reverse edge (v_3, v_2) is present in c_2. That is, $\{v_2, v_3\}$ induces a 2-cycle in K. Hence both c_1 and c_2 are ghosts, contrary to our assumption. ◀

Smooth cycles. In order to avoid twin and fan configurations in $D(f, g)$, we would like to choose an injective function $f : \mathcal{H} \to V$, with $f(h) \in \mathrm{V}(h)$, that avoids short cycles in K, except for a special type of cycles (called *smooth*) to be defined next.

▶ **Definition 16.** Let $c = (v_1, \ldots, v_k)$ be a simple short cycle in the conflict graph K. Recall that every edge in K lies in a region $R(h)$, $h \in \mathcal{H}$, and is directed to $f(h)$. So the cycle c corresponds to a cycle of hexagons $(h_1, \ldots, h_k)$, such that the vertex $v_i = f(h_i)$ lies on the boundary of hexagons h_i and $h_{i\oplus 1}$, for $i = 1, \ldots, k$. We say that the hexagons $h_1, \ldots, h_k$ are **associated** with c. The cycle c is **smooth** if none of the associated hexagons has a vertex in $\mathsf{int}(c)$. (For example, the cycles in Figure 6a are smooth, but the 3-cycle in Figure 6b is not.)

Note that a smooth cycle in K may contain many vertices of various hexagons in its interior; the restrictions apply only to those (two or three) hexagons that are associated with the cycle. For instance, there could be several hexagons in the white regions between the hexagons in Figure 6. Let $\mathcal{C}_\mathrm{s}$ denote the set of all smooth cycles in $\mathcal{C}$, that is, the set of all short smooth nonghost cycles in K. In Section 2.4, we show how to choose f such that all cycles in $\mathcal{C}$ are smooth, that is, $\mathcal{C} = \mathcal{C}_\mathrm{s}$.

Properties of smooth cycles. The following three lemmata formulate some important properties of smooth cycles that hold for any injective function $f : \mathcal{H} \to V$, where $f(h) \in \mathrm{V}(h)$ for all $h \in \mathcal{H}$.

▶ **Lemma 17.** *Let $c \in \mathcal{C}_\mathrm{s}$ and let $u \in \mathsf{int}(c)$ be a vertex of G. Then there is no edge (u, v) in K for any $v \in \mathrm{V}(c)$.*

Proof. Suppose for the sake of a contradiction that (u, v) is an edge of K with $v \in \mathrm{V}(c)$. Let h be the hexagon with $f(h) = v$. All edges towards v are drawn inside h so that, in particular, $u \in \mathrm{V}(h)$. As h is associated with c, this contradicts the assumption that c is smooth. ◀

▶ **Lemma 18.** *Let $c_1, c_2 \in \mathcal{C}_\mathrm{s}$ so that $c_1 \neq c_2$ and c_2 contains c_1. Then $\mathrm{V}(c_1) \cap \mathrm{V}(c_2) = \emptyset$.*

Proof. Suppose to the contrary that there exists a vertex $u \in \mathrm{V}(c_1) \cap \mathrm{V}(c_2)$. We claim that $\mathrm{V}(c_1) \cap \mathsf{int}(c_2) = \emptyset$. To see this, consider a vertex $v \in \mathrm{V}(c_1) \cap \mathsf{int}(c_2)$. Then following c_1 from v to u we find an edge (x, y) of K so that $x \in \mathsf{int}(c_2)$ and $y \in \mathrm{V}(c_2)$. However, such an edge does not exist by Lemma 17. Hence there is no such vertex v and $\mathrm{V}(c_1) \cap \mathsf{int}(c_2) = \emptyset$. Given that c_2 contains c_1, it follows that $\mathrm{V}(c_1) \subseteq \mathrm{V}(c_2)$.

If c_1 is a 3-cycle, then so is c_2 and Lemma 14 contradicts our assumption $c_1 \neq c_2$. Hence c_1 is a 2-cycle and c_2 is a 3-cycle. But then c_2 is a ghost, in contradiction to $c_2 \in \mathcal{C}_\mathrm{s}$. ◀

▶ **Lemma 19.** *Any two cycles in $\mathcal{C}_\mathrm{s}$ are interior-disjoint or vertex disjoint.*

Proof. Let $c_1, c_2 \in \mathcal{C}_\mathrm{s}$ with $c_1 \neq c_2$. Suppose, to the contrary, that $\mathsf{int}(c_1) \cap \mathsf{int}(c_2) \neq \emptyset$ and $V(c_1) \cap V(c_2) \neq \emptyset$. Without loss of generality, an edge (u_1, u_2) of c_2 lies in the interior of c_1.

We may assume that u_1 and u_2 are common vertices of c_1 and c_2. Indeed, if u_1 and u_2 were not common vertices of the cycles, then a vertex of c_2 would lie in the interior of c_1. Then c_1 contains c_2 by Lemma 15, and $\mathrm{V}(c_1) \cap \mathrm{V}(c_2) = \emptyset$ by Lemma 18.

We may further assume that both c_1 and c_2 are 3-cycles. Indeed, if the vertex set of one of them contains that of the other, then one of them is a 3-cycle and the other is a 2-cycle. Since both c_1 and c_2 are present in $\mathcal{C}$, one of them would be a ghost cycle in $\mathcal{C}$, contradicting the definition of $\mathcal{C}$.

Since (u_1, u_2) is a directed edge of c_2 that lies in the interior of c_1, and c_1 is a 3-cycle that has an edge between any two of its vertices, the edge (u_2, u_1) is present in c_1. This implies that $c_3 = (u_1, u_2)$ is a 2-cycle in K. Therefore $c_3 \in \mathcal{C}$, and both c_1 and c_2 are ghost cycles in $\mathcal{C}$, contradicting the definition of $\mathcal{C}$, $\mathcal{C} \supseteq \mathcal{C}_\mathrm{s}$. This confirms that $c_1, c_2 \in \mathcal{C}_\mathrm{s}$, $c_1 \neq c_2$, are interior-disjoint or vertex disjoint, as claimed. ◀

2.4 Choosing the special vertices $f(h)$

As noted above, Angelini et al. proved [6, Lemmata 3 and 4] that there exists an injective map $f : \mathcal{H} \to V$ that maps every hexagon $h \in \mathcal{H}$ to a vertex $v \in \mathrm{V}(h)$. We review their argument (using Hall's matching theorem), and then strengthen the result to establish some additional properties of the function $f : \mathcal{H} \to V$.

Hall's condition. Let $\mathcal{A} \subseteq \mathcal{H}$ be a subset of hexagons, and let $\mathrm{V}(\mathcal{A}) \subseteq V$ be the set of vertices incident to the hexagons in $\mathcal{A}$. Following Angelini et al. [6, Lemma 4] we obtain Hall's condition via double counting.

▶ **Lemma 20.** *For every subset $\mathcal{A} \subseteq \mathcal{H}$, we have $|\mathrm{V}(\mathcal{A})| \geq 2|\mathcal{A}| + 2$.*

▶ **Corollary 21.** *There exists an injective map $f : \mathcal{H} \to V$ that maps every hexagon $h \in \mathcal{H}$ to a vertex $v \in \mathrm{V}(h)$.*

▶ **Corollary 22.** *For every nonempty subset $\mathcal{A} \subseteq \mathcal{H}$, we have $|\mathrm{V}(\mathcal{A})| \geq |\mathcal{A}| + 5$.*

Proof. If $|\mathcal{A}| = 1$, then $|\mathcal{A}| + 5 = 6$ and a single hexagon has 6 distinct vertices. If $|\mathcal{A}| = 2$, then $|\mathcal{A}|+5 = 7$; and two distinct hexagons have at least 7 distinct vertices by Observation 5a. Otherwise $|\mathcal{A}| \geq 3$, and Lemma 20 yields $|\mathrm{V}(\mathcal{A})| \geq 2|\mathcal{A}| + 2 \geq |\mathcal{A}| + 5$. ◀

▶ **Lemma 23.** *There exists an injective function $f : \mathcal{H} \to V$ such that $f(h) \in \mathrm{V}(h)$, for every $h \in \mathcal{H}$, and every cycle in $\mathcal{C}$ is smooth.*

2.5 Choosing the special edges $g(h)$

Let $f : \mathcal{H} \to V$ be a function as described in Lemma 23. That is, in the following we assume $\mathcal{C} = \mathcal{C}_{\mathrm{s}}$ (all short nonghost cycles in K are smooth). We use Hall's theorem to show that there is a matching of the cycles in $\mathcal{C}$ to the vertices in V such that each cycle is matched to an incident vertex. For a subset $\mathcal{B} \subseteq \mathcal{C}$, let $\mathrm{V}(\mathcal{B})$ denote the set of all vertices incident to some cycle in $\mathcal{B}$.

▶ **Lemma 24.** *For every set $\mathcal{B}_0 \subseteq \mathcal{C}$ of pairwise interior-disjoint cycles, $|\mathcal{B}_0| \leq |\mathrm{V}(\mathcal{B}_0)|$.*

Proof. We use double counting. Let I be the set of all pairs $(v, c) \in V \times \mathcal{B}_0$ such that v is incident to c. Every cycle is incident to ≥ 2 vertices, hence $|I| \geq 2|\mathcal{B}_0|$. By Lemma 13, every vertex is incident to at most two interior-disjoint cycles. Consequently, $|I| \leq 2|\mathrm{V}(\mathcal{B}_0)|$. The combination of the upper and lower bounds for $|I|$ yields $|\mathcal{B}_0| \leq |\mathrm{V}(\mathcal{B}_0)|$. ◀

▶ **Lemma 25.** *For every set $\mathcal{B} \subseteq \mathcal{C}$ of cycles, we have $|\mathcal{B}| \leq |\mathrm{V}(\mathcal{B})|$.*

▶ **Lemma 26.** *There exists an injective function $s : \mathcal{C} \to V$ that maps every cycle in $\mathcal{C}$ to one of its vertices.*

We are ready to define the function $g : \mathcal{H} \to E$, that maps every hexagon $h \in \mathcal{H}$ to one of its edges.

▶ **Lemma 27.** *There is a function $g : \mathcal{H} \to E$ such that*
- *for every $h \in \mathcal{H}$, $g(h) \in h$ and $g(h)$ is not incident to $f(h)$;*
- *for every 2-cycle $(f(h_1), f(h_2))$ in K, the edges $g(h_1)$ and $g(h_2)$ do not cross in $D(f, g)$;*
- *for every 3-cycle $(f(h_1), f(h_2), f(h_3))$ in K, at least two of the edges $g(h_1)$, $g(h_2)$, and $g(h_3)$ do not cross in $D(f, g)$.*

Proof. By Lemma 26, there is an injective function $s : \mathcal{C} \to V$ that maps every cycle $c \in \mathcal{C}$ to one of its vertices. For each cycle $c \in \mathcal{C}$, vertex $s(c)$ is the endpoint of some directed edge $(q(c), s(c))$ in the conflict graph. Consequently, there is a hexagon $h \in \mathcal{H}$ such that $s(c) = f(h)$ and $q(c) \in \mathrm{V}(h)$. We say that h is *assigned* to the cycle c. We distinguish two types of hexagons, depending on whether or not they are assigned to a 2-cycle of $\mathcal{C}$.

Hexagons that are not assigned to 2-cycles. For every hexagon h that is not assigned to any cycle, choose $g(h)$ to be an arbitrary edge in h that is not incident to the vertex $f(h)$. For every hexagon h that is assigned to a 3-cycle $c \in \mathcal{C}$, choose $g(h)$ to be the (unique) edge in h that is incident to neither $q(c)$ nor $s(c)$. If $c = (f(h_1), f(h_2), f(h_3))$ and without loss of generality $s(c) = f(h_2)$, then $g(h_2)$ is not incident to $f(h_1) = q(c)$, consequently $g(h_1)$ is disjoint from $g(h_2)$. (Note that $g(h_1)$ is not incident to $f(h_2) = s(c)$ because this would induce a 2-cycle in K, making c a ghost.)

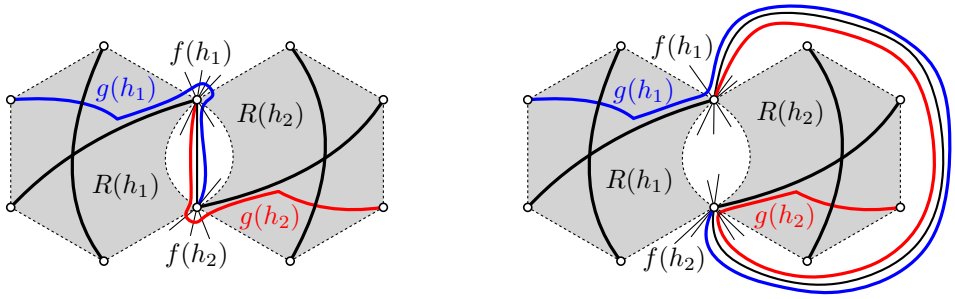


Figure 7 In Case 1 of Lemma 27, the edge $g(h_2)$ is incident to $f(h_1)$. We set $g(h_1)$ so that it is incident to $f(h_2)$. Regardless of how $f(h_1)f(h_2)$ is drawn, the edge separates $g(h_1)$ and $g(h_2)$ and ensures that they are disjoint.

Hexagons assigned to 2-cycles. Consider a 2-cycle $c \in \mathcal{C}$, and let h_1 and h_2 denote the associated hexagons so that without loss of generality $s(c) = f(h_1)$. Suppose without loss of generality that c is oriented clockwise. We distinguish three cases.

Case 1: $g(h_2)$ has already been selected and $g(h_2)$ is incident to $f(h_1)$. Then let $g(h_1)$ be the unique edge in h_1 incident to $f(h_2)$ (Figure 7). We claim that $g(h_1)$ and $g(h_2)$ do not cross in $D(f, g)$. As both edges are critical, by Lemma 6 they can only cross in the neighborhood of $f(h_1)$ or $f(h_2)$. Let a_i be the edge of h_i incident to $f(h_i)$, for $i \in \{1, 2\}$. The edge $g(h_i)$, for $i \in \{1, 2\}$, follows a_i towards the neighborhood of $f(h_i)$ and then crosses the edges incident to $f(h_i)$ following a_i in clockwise order (the orientation of c) until reaching the edge $f(h_1)f(h_2)$. Then $g(h_i)$ follows $f(h_1)f(h_2)$ to its other endpoint, without crossing the edge. Therefore, the path formed by the edges a_1, $f(h_1)f(h_2)$, and a_2 splits the neighborhoods of $f(h_1)$ and $f(h_2)$ into two components so that $g(h_1)$ and $g(h_2)$ are in different components. Thus $g(h_1)$ and $g(h_2)$ do not cross, as claimed.

Case 2: $g(h_2)$ has already been selected and $g(h_2)$ is not incident to $f(h_1)$. Then let $g(h_1)$ be the unique edge in h_1 incident to neither $f(h_1)$ nor $f(h_2)$ (Figure 8a). We claim that $g(h_1)$ and $g(h_2)$ do not cross in $D(f, g)$. As both edges are critical, by Lemma 6 they can only cross in the neighborhood of $f(h_1)$ or $f(h_2)$. But as $g(h_1)$ is not incident to $f(h_2)$, there is a neighborhood of $f(h_2)$ that is disjoint from $g(h_1)$, and so $g(h_1)$ and $g(h_2)$ do not cross there. Similarly, there is a neighborhood of $f(h_1)$ that is disjoint from $g(h_2)$, and so $g(h_1)$ and $g(h_2)$ do not cross there, either. Thus $g(h_1)$ and $g(h_2)$ do not cross in $D(f, g)$.

Case 3: no hexagon h_1 is assigned to a 2-cycle so that $g(h_2)$ has already been selected. Then we are left with hexagons that correspond to 2-cycles and form *cycles* $L = (h_1, \ldots, h_k)$ such that $(f(h_i), f(h_{i\oplus 1}))$ is a 2-cycle in $\mathcal{C}$, for $i = 1 \ldots, k$. These cycles are interior-disjoint by Lemma 19, and any two consecutive cycles in L have opposite orientations by Lemma 13. It follows that k is even.

Since every 2-cycle in L is smooth, the three vertices $f(h_{i\ominus 1})$, $f(h_i)$, and $f(v_{i\oplus 1})$ are consecutive along $\partial R(h_i)$. For every odd $i \in \{1, \ldots, k\}$, let $g(h_i)$ be the (unique) edge in h_i incident to $f(h_{i\ominus 1})$ (and incident to neither $f(h_i)$ nor $f(h_{i\oplus 1})$). Similarly, for every even $i \in \{1, \ldots, k\}$, let $g(h_i)$ be the edge in h_i incident to $f(h_{i\oplus 1})$ (and incident to neither $f(h_i)$ nor $f(h_{i\ominus 1})$). Refer to Figure 8b.

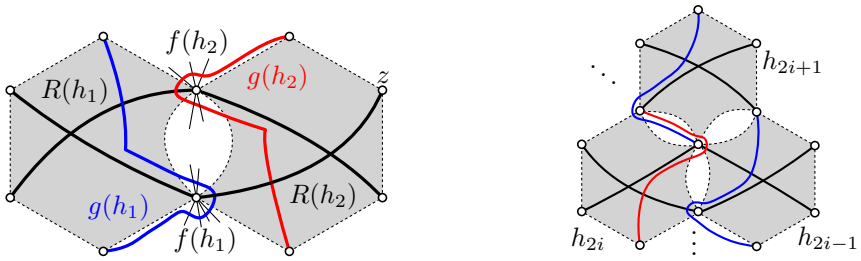


Figure 8 (a) In Case 2 of Lemma 27, the edge $g(h_2)$ is not incident to $f(h_1)$. We set $g(h_1)$ so that it is not incident to $f(h_2)$, to ensure that $g(h_1)$ and $g(h_2)$ are disjoint. (b) In Case 3 we face a cycle of 2-cycles. We consistently select edges to be rerouted in even (red edge) and odd (blue edge) hexagons so that they are pairwise disjoint.

For every odd index $i \in \{1, \ldots, k\}$, the rerouted edges $g(h_i)$ and $g(h_{i\oplus 1})$ are incident to neither $f(h_{i\oplus 1})$ nor $f(h_i)$. Similarly, for every even index $i \in \{1, \ldots, k\}$, the rerouted edges $g(h_i)$ and $g(h_{i\oplus 1})$ are incident to $f(h_{i\oplus 1})$ and $f(h_i)$, respectively. In both cases, the rerouted edges $g(h_i)$ and $g(h_{i\oplus 1})$ are disjoint.

Ghost cycles. It remains to consider ghost cycles. Let c_1 be a ghost cycle in K. Without loss of generality, assume that $c_1 = (v_1, v_2, v_3)$, where $v_1 = f(h_1)$, $v_2 = f(h_2)$, and $v_3 = f(h_3)$, and $c_2 = (v_1, v_2)$ is a 2-cycle in $\mathcal{C}$. Recall that c_2 is smooth (cf. Lemma 23). By construction, $g(h_1)$ and $g(h_2)$ do not cross in $D(f, g)$. Hence at least two of the edges in $\{g(h_1), g(h_2), g(h_3)\}$ do not cross in $D(f, g)$, as required. ◀

The combination of Lemma 10, Lemma 23, and Lemma 27 proves Lemma 11, which completes the proof of Theorem 1.

2.6 Quasiplane simple topological drawings

The redrawing algorithm in Section 2.1 transformed a 2-plane drawing D with properties (i)–(iii), and rerouted some of the edges in two phases to obtain a quasiplane drawing $D(f, g)$. In this section, we show that the algorithm produces a simple topological drawing, that is, any two edges cross at most once, and no two adjacent edges cross.

▶ **Theorem 2.** *Every 2-planar graph admits a quasiplane simple topological drawing.*

3 Conclusions

We have proved that every 2-planar graph is quasiplanar (Theorem 1) by showing that a 2-plane topological graph can be transformed into a quasiplane topological graph, in which no three edges pairwise cross. Theorem 2 strengthens the result to produce a quasiplane *simple* topological graph (any two edges cross at most once and adjacent edges do not cross).

In Section 2.4, we have shown that we can choose one vertex $f(h)$ for each hexagon $h \in \mathcal{H}$ such that all 2- and 3-cycles in the conflict graph K have some special properties. It is unclear, however, whether 2- and 3-cycles can be avoided altogether by a suitable choice of the function f. We formulate an open problem to this effect: Given a set $\mathcal{H}$ of interior-disjoint (topological) hexagons in the plane on a vertex set V, is there an injective function $f : \mathcal{H} \to V$

such that the conflict digraph K contains no 2-cycles (alternatively, neither 2- nor 3-cycles)? Several fundamental problems remain open for k-quasiplanar graphs:

- What is the computational complexity of recognizing k-quasiplanar graphs? Is there a polynomial-time algorithm that decides whether a given graph is quasiplanar (or k-quasiplanar for a given constant k)?
- Is there a constant c_k for every $k \in \mathbb{N}$ such that an n-vertex k-quasiplanar graph has at most $c_k n$ edges [20]? Affirmative answers are known for $k \leq 4$ only [1].
- By Theorem 1 and the main result in [6], every k-planar graph is $(k+1)$-quasiplanar, where $k \in \mathbb{N}$, $k \geq 2$. Angelini et al. [6] ask whether this result can be improved for large k: Denote by $\ell(k) \in \mathbb{N}$ the minimum integer such that every k-planar graph is ℓ-quasiplanar. Prove or disprove that $\ell(k) = o(k)$.

Acknowledgements. This work began at the *Fifth Annual Workshop on Geometry and Graphs*, March 6–10, 2017, at the Bellairs Research Institute of McGill University. We thank the organizers and all participants for the productive and positive atmosphere.

References

1 Eyal Ackerman. On the maximum number of edges in topological graphs with no four pairwise crossing edges. *Discrete Comput. Geom.*, 41(3):365–375, 2009. doi:10.1007/s00454-009-9143-9.

2 Eyal Ackerman. On topological graphs with at most four crossings per edge. *CoRR*, abs/1509.01932:1–41, 2015. URL: https://arxiv.org/abs/1509.01932.

3 Eyal Ackerman and Gábor Tardos. On the maximum number of edges in quasi-planar graphs. *J. Combin. Theory Ser. A*, 114(3):563–571, 2007. doi:10.1016/j.jcta.2006.08.002.

4 Pankaj K. Agarwal, Boris Aronov, János Pach, Richard Pollack, and Micha Sharir. Quasi-planar graphs have a linear number of edges. *Combinatorica*, 17:1–9, 1997. doi:10.1007/BF01196127.

5 Miklós Ajtai, Václav Chvátal, Monroe Newborn, and Endre Szemerédi. Crossing-free subgraphs. *Ann. Discrete Math.*, 12:9–12, 1982. doi:10.1016/S0304-0208(08)73484-4.

6 Patrizio Angelini, Michael A. Bekos, Franz J. Brandenburg, Giordano Da Lozzo, Giuseppe Di Battista, Walter Didimo, Giuseppe Liotta, Fabrizio Montecchiani, and Ignaz Rutter. On the relationship between k-planar and k-quasi planar graphs. In *Proc. 43rd Internat. Workshop Graph-Theoret. Concepts Comput. Sci.*, Lecture Notes Comput. Sci. Springer, 2017. to appear; preliminary version available at arXiv:1702.08716. URL: http://arxiv.org/abs/1702.08716.

7 Michael Bekos, Michael Kaufmann, and Chrysanthi Raftopoulou. On optimal 2- and 3-planar graphs. In *Proc. 33rd Internat. Sympos. Comput. Geom.*, LIPIcs. Schloß Dagstuhl, 2017. to appear; preliminary version available at arXiv:1703.06526. URL: http://arxiv.org/abs/1702.08716.

8 Franz J. Brandenburg, Walter Didimo, William S. Evans, Philipp Kindermann, Giuseppe Liotta, and Fabrizio Montecchiani. Recognizing and drawing IC-planar graphs. *Theoret. Comput. Sci.*, 636:1–16, 2016. doi:10.1016/j.tcs.2016.04.026.

9 Christian A. Duncan. On graph thickness, geometric thickness, and separator theorems. *Comput. Geom. Theory Appl.*, 44(2):95–99, 2011. doi:10.1016/j.comgeo.2010.09.005.

10 David Eppstein, Philipp Kindermann, Stephen G. Kobourov, Giuseppe Liotta, Anna Lubiw, Aude Maignan, Debajyoti Mondal, Hamideh Vosoughpour, Sue Whitesides, and Stephen K.

Wismath. On the planar split thickness of graphs. In *Proc. 12th Latin Amer. Sympos. Theoretical Informatics*, volume 9644 of *Lecture Notes Comput. Sci.*, pages 403–415. Springer, 2016. doi:10.1007/978-3-662-49529-2_30.

11 Jacob Fox and János Pach. Applications of a new separator theorem for string graphs. *Combinatorics, Probability and Computing*, 23(1):66–74, 2012. doi:10.1017/S0963548313000412.

12 Jacob Fox and János Pach. Coloring K_k-free intersection graphs of geometric objects in the plane. *European J. Combin.*, 33(5):853–866, 2012. doi:10.1016/j.ejc.2011.09.021.

13 Jacob Fox, János Pach, and Andrew Suk. The number of edges in k-quasi-planar graphs. *SIAM J. Discrete Math.*, 27(1):550–561, 2013. doi:10.1137/110858586.

14 Radoslav Fulek, Michael J. Pelsmajer, Marcus Schaefer, and Daniel Štefankovič. Adjacent crossings do matter. *J. Graph Algorithms Appl.*, 16(3):759–782, 2012. doi:10.7155/jgaa.00266.

15 Michael Hoffmann and Csaba D. Tóth. Two-planar graphs are quasiplanar. *CoRR*, abs/1705.05569:1–22, 2017. URL: https://arxiv.org/abs/1705.05569.

16 Stephen G. Kobourov, Giuseppe Liotta, and Fabrizio Montecchiani. An annotated bibliography on 1-planarity. *CoRR*, abs/1703.02261:1–38, 2017. URL: https://arxiv.org/abs/1703.02261.

17 László Lovász. Graph minor theory. *Bulletin of the American Mathematical Society*, 43(1):75–86, 2006. doi:10.1090/S0273-0979-05-01088-8.

18 János Pach, Radoš Radoičić, Gábor Tardos, and Géza Tóth. Improving the crossing lemma by finding more crossings in sparse graphs. *Discrete Comput. Geom.*, 36(4):527–552, 2006. doi:10.1007/s00454-006-1264-9.

19 János Pach, Radoš Radoičić, and Géza Tóth. Relaxing planarity for topological graphs. In *More Graphs, Sets and Numbers*, volume 15 of *Bolyai Soc. Math. Studies*, pages 285–300. Springer, 2006. doi:10.1007/978-3-540-32439-3_12.

20 János Pach, Farhad Shahrokhi, and Mario Szegedy. Applications of the crossing number. *Algorithmica*, 16(1):111–117, 1996. doi:10.1007/BF02086610.

21 János Pach and Géza Tóth. Graphs drawn with few crossings per edge. *Combinatorica*, 17(3):427–439, 1997. doi:10.1007/BF01215922.

22 Marcus Schaefer. The graph crossing number and its variants: a survey. *Electronic J. Combinatorics*, #DS21:1–100, 2014. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/DS21.

23 Andrew Suk and Bartosz Walczak. New bounds on the maximum number of edges in k-quasi-planar graphs. *Comput. Geom. Theory Appl.*, 50:24–33, 2015. doi:10.1016/j.comgeo.2015.06.001.

The Shortest Identities for Max-Plus Automata with Two States*

Laure Daviaud[1] and Marianne Johnson[2]

1 **MIMUW, University of Warsaw, Poland**
ldaviaud@mimuw.edu.pl
2 **School of Mathematics, University of Manchester, UK**
Marianne.Johnson@manchester.ac.uk

Abstract

Max-plus automata are quantitative extensions of automata designed to associate an integer with every non-empty word. A pair of distinct words is said to be an identity for a class of max-plus automata if each of the automata in the class computes the same value on the two words. We give the shortest identities holding for the class of max-plus automata with two states. For this, we exhibit an interesting list of necessary conditions for an identity to hold. Moreover, this result provides a counter-example of a conjecture of Izhakian, concerning the minimality of certain identities.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Max-plus automata, Weighted automata, Identities, Tropical matrices

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.48

1 Introduction

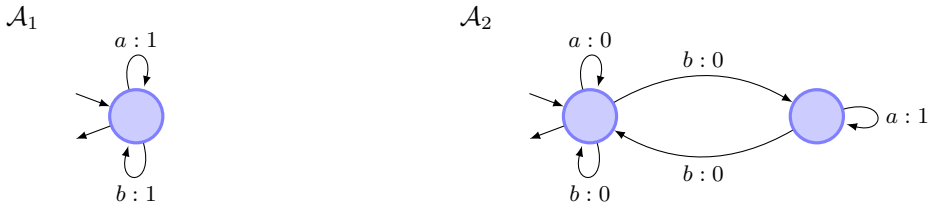
A natural question when dealing with computational models is to understand which pairs of inputs can be separated by the model, *i.e.* lead to different results. Or conversely, which pairs of distinct inputs will give the same computation. These pairs are called identities for the model. Regarding finite automata, two words are said to be separated by a given automaton if one is accepted and the other is rejected. When fixing an automaton, or even considering the class of automata with at most a certain number of states, we know that some pairs of distinct words are not separated. It is a simple argument of cardinality: the number of automata with a bounded number of states is finite and each of them computes a boolean value on a given word, while the number of words is infinite. However, when considering the full class, for every pair of distinct words, it is easy to construct an automaton accepting one and rejecting the other.

When dealing with quantitative extensions of automata, namely weighted automata, the situation is much more intricate. Weighted automata were introduced by Schützenberger in [12]. They compute functions from the set of words to the set of values of a semiring, allowing one to model quantities such as costs, gains or probabilities. The question of separating words (*i.e.* computing different values on the words) highly depends on the semiring. For probabilistic automata, or automata on the usual semiring $(\mathbb{R}, +, \times)$, it is known that there is an automaton (with two states) which separates every pair of distinct words.

* This work was partially supported by the LIPA project, funded by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No 683080).

In this paper we are interested in max-plus automata, which are weighted automata over the tropical semiring that compute functions from the set of *non-empty* words to the values of the semiring $\mathbb{Z}_{\max} = (\mathbb{Z} \cup \{-\infty\}, \max, +)$. From now on, for simplicity, we may use the notation $\mathbb{Z}_{\max}$ to denote the semiring or the set of its values $\mathbb{Z} \cup \{-\infty\}$. We note that some authors prefer to work with min-plus automata, which compute values in the min-plus semiring $\mathbb{Z}_{\min} = (\mathbb{Z} \cup \{+\infty\}, \min, +)$. Since the two semirings ($\mathbb{Z}_{\max}$ and $\mathbb{Z}_{\min}$) are isomorphic, the results presented here can be easily translated to the min-plus case.

A *max-plus automaton* is a finite automaton whose transitions are weighted by integers. An easy way to think about these weights is to consider them as amounts of money that you win when you go through a transition. Along a run, you accumulate this money (you sum the amounts; this sum is called the *weight of the run*), and your purpose is, given a word w, to go from an initial state to a final state, by reading w and grabbing the maximal amount of money you can. The *value (or weight) associated with* w is the maximum possible amount that you could win by reading the word w. Max-plus automata are thus particularly suitable to model gain maximisation, study the worst-case complexity of a program [2] or evaluate performance of discrete event systems [4, 5]. Let us give two examples on the alphabet $\{a, b\}$ (where initial and final states are denoted by ingoing and outgoing arrows respectively):



The automaton $\mathcal{A}_1$ associates with each word its length. The function computed by the automaton $\mathcal{A}_2$ is more complicated and we begin by describing its behaviour in particular cases. Consider a word of the form $ba^{k_1}ba^{k_2}b\cdots ba^{k_\ell}b$ where all the k_i are positive integers. Then, the value computed by the automaton is the maximum of the sums $k_{i_1} + k_{i_2} + \cdots + k_{i_m}$ where no two i_j are consecutive, that is to say, $i_{j+1} \geqslant i_j + 2$ for all j.

Two distinct words u and v are separated by a class of max-plus automata if there is an automaton in the class that associates two different values on the two words, otherwise they form an identity for the class, usually denoted $u = v$. But this question is much more intricate than in the previous cases of boolean automata and automata weighted over the usual semiring. We can show that a single max-plus automaton cannot separate all pairs of distinct words. It is again a simple cardinality argument: if the weights of the transitions of an automaton are between $-m$ and m, then the value associated to a word of length n is between $-mn$ and mn, while the number of words of length n on a finite alphabet Σ is $|\Sigma|^n$. For n large enough, there must exist two distinct words having the same value. It is also clear that given two distinct words, one can construct a max-plus automaton (with an arbitrarily large number of states) separating them. A major open question is the following:

Given a bound d, does there exist an identity for the set of max-plus automata with at most d states?

In that case, a simple cardinality argument fails. This question was first considered in [9] where it was answered positively for $d = 2$. The known identity for two states consists of a pair of words of length 20, but the problem seems very difficult to tackle in the general case. Shitov [13] proposed an identity for $d = 3$ consisting of a pair of words of length 1795308.

Currently no generalisation of these results seems conceivable, the ultimate (very far) goal being to characterise the complete set of identities. This paper is motivated by the fact that a better understanding of the identities in the case $d = 2$ is already a first step for a better understanding of the general case.

Contribution. We focus on the class of max-plus automata with two states, denoted $\mathcal{C}$. It is easy to see that if $u = v$ is an identity which holds in $\mathcal{C}$ then u and v have the same length (see for example $\mathcal{A}_1$), defining the *length of the identity*. We give the unique two identities of minimal length (17) which hold in $\mathcal{C}$.

▶ **Theorem 1.** *There are two identities (up to a renaming of the letters) of minimal length which hold in the class of max-plus automata with two states:*

$$a^2b^3a^3babab^3a^2 = a^2b^3ababa^3b^3a^2 \quad \textit{and} \quad ab^3a^4baba^2b^3a = ab^3a^2baba^4b^3a$$

To achieve this goal, we give a rather short list of necessary conditions for an identity to hold which together eliminate all the other possible candidates of length shorter than 18 (Proposition 10, Section 3). This list is short enough that it can be tested by computer. We also prove that this list is minimal in the sense that each of the conditions eliminates at least one pair of words, that cannot be eliminated using the other conditions alone. However, this list is probably not complete, and future works will consist in trying to extend it to fully characterise the identities holding in $\mathcal{C}$. We then prove that the identities given in the statement of Theorem 1 hold in $\mathcal{C}$ (Proposition 11, Section 4).

Link with matrices. This topic is closely related with the question of identities on semigroups of matrices over the tropical semiring. We consider matrices with entries in $\mathbb{Z} \cup \{-\infty\}$ and the product AB for two matrices A, B (provided the number of columns of A and the number of rows of B coincide, denoted here d) is defined as $(AB)_{i,j} = \max_{1 \leqslant k \leqslant d}(A_{i,k} + B_{k,j})$.

An identity $u = v$ is said to be satisfied by a semigroup of matrices if for all substitutions of the letters by matrices in the semigroup, the equality holds.

A max-plus automaton with d states can equivalently be represented by a semigroup of matrices of dimension d: the states are numbered $\{1, \dots, d\}$ and for each letter, a square matrix $\mu(a)$ of dimension d is defined such that the (i, j)-coefficient contains the weight of the transition from state i to state j labelled by a or $-\infty$ if there is no such transition. Then μ extends to give a semigroup morphism $\mu : \Sigma^+ \to M_n(\mathbb{Z}_{\max})$. For a non-empty word w, it is straightforward to verify that $\mu(w)_{i,j}$ is the maximum of the weights of the runs from state i to state j, labelled by w. An initial vector I (*resp.* final vector F) with 1 row and d columns (*resp.* d rows and 1 column) and entries in $\{0, -\infty\}$ is defined by $I_i = 0$ (*resp.* $F_i = 0$) if and only if state i is initial (*resp.* final). The weight of a word w in $\mathcal{A}$ is exactly the value given by $I\mu(w)F \in \mathbb{Z}_{\max}$ (see for example [11] for more explanations). The max-plus automaton $\mathcal{A}_1$ illustrated on the previous page is represented by $\mu(a) = (1)$, $\mu(b) = (1)$ and $I = F = (0)$, while $\mathcal{A}_2$ is represented by:

$$\mu(a) = \begin{pmatrix} 0 & -\infty \\ -\infty & 1 \end{pmatrix} \qquad \mu(b) = \begin{pmatrix} 0 & 0 \\ 0 & -\infty \end{pmatrix} \qquad I = \begin{pmatrix} 0 & -\infty \end{pmatrix} \qquad F = \begin{pmatrix} 0 \\ -\infty \end{pmatrix}$$

Using this representation, it can be easily shown that $u = v$ is an identity which holds in $\mathcal{C}$ if and only if $u = v$ holds for the semigroup of square matrices of dimension 2. Then Theorem 1 implies the following theorem.

▶ **Theorem 2.** *There are two identities (up to a renaming of the letters) of minimal length which hold in the semigroup of tropical square matrices of dimension* 2*:*

$$a^2b^3a^3babab^3a^2 = a^2b^3ababa^3b^3a^2 \quad \textit{and} \quad ab^3a^4baba^2b^3a = ab^3a^2baba^4b^3a$$

Using the fact that the identities which hold in the semigroup of tropical square matrices of dimension 2 are the same as those which hold in the semigroup of tropical square matrices of dimension 2 with real entries (as explained in Section 2 below), Theorem 2 gives a counter-example to a conjecture of Izhakian concerning the structure of the identities of minimal length. Indeed, in [8, Conjecture 5.1] he provides a method of constructing identities satisfied by every subsemigroup of the semigroup of the tropical square matrices of dimension d consisting of matrices with maximal (tropical) rank (see [8] for detailed definitions), conjecturing that certain amongst these are of minimal length, but for $d = 2$, the shortest identities produced by this method have length greater than 17.

Organisation of the paper. In Section 2, we give first properties. In particular, we make some comments about working with weights in $\mathbb{Z}$ rather than $\mathbb{N}$, $\mathbb{Q}$ or $\mathbb{R}$, restricting the automata to have only one initial and one final state and considering only 2-letter alphabets. In Section 3, we give the list of conditions allowing us to eliminate all the pairs of words up to length 17 except two (up to renaming of the letters). In Section 4, we prove that these pairs do indeed form identities.

2 First properties

Given a word w and a letter a, we write $|w|$ to denote the length of w and $|w|_a$ to denote the number of occurrences of the letter a in w. If $w = w_0w_1\cdots w_\ell$ with $w_0, w_1, \ldots, w_\ell$ letters, the positions of w are $0, 1, \ldots \ell$ and w_i is said to be the letter at position i. In a max-plus automaton $\mathcal{A}$, a run labelled by w from a state p to a state q with weight α will be denoted $p \xrightarrow{w\,:\,\alpha} q$. We denote by $[\![\mathcal{A}]\!]$ the function (from the set of non-empty words over a finite alphabet Σ to $\mathbb{Z}_{\max}$) computed by $\mathcal{A}$. Let us recall that $\mathcal{C}$ denotes the class of all the max-plus automata with two states. More generally, for any positive integer d, we denote by $\mathcal{C}_d$ the class of all the max-plus automata with d states (so that $\mathcal{C}_2 = \mathcal{C}$). An identity over Σ, that is to say a pair of two distinct non-empty words over Σ, denoted $u = v$, holds in $\mathcal{C}_d$ if and only if for all $\mathcal{A} \in \mathcal{C}_d$, $[\![\mathcal{A}]\!](u) = [\![\mathcal{A}]\!](v)$. For now, we fix an integer $d \geqslant 2$.

Content. If $\Sigma = \{a_1, \ldots a_n\}$, the *content* of a word w is the n-tuple $(|w|_{a_1}, \ldots, |w|_{a_n})$ of the number of occurrences of each of the letters in w.

▶ **Lemma 3.** *If $u = v$ holds in $\mathcal{C}_d$, then u and v have the same content. In particular u and v have the same length.*

Proof. The number of occurrences of a letter a can be computed by a max-plus automaton with one state (both initial and final) with one transition for each letter of Σ, where the transition labelled by a has weight 1 and all other transitions have weight 0. (Note that this can be seen as a max-plus automaton with d states by simply adding states, and possibly transitions with weight 0). Thus, if the content of u and v differs then there exist an automaton in $\mathcal{C}_d$ computing two different values on these two words. ◀

Initial and final states. In the rest of the paper, we will freely use the following fact:

▶ **Lemma 4.** *An identity $u = v$ holds in $\mathcal{C}_d$ if and only if it holds in the class of max-plus automata with d states having exactly one initial and one final state.*

Proof. Denote by $\mathcal{C}'_d$ the class of max-plus automata with d states and exactly one initial and one final state. Clearly, if $u = v$ holds in $\mathcal{C}_d$, it must also hold in $\mathcal{C}'_d$. Conversely, suppose $u = v$ holds in $\mathcal{C}'_d$ and let $\mathcal{A} \in \mathcal{C}_d$. Consider now the set $\mathcal{S}$ of the max-plus automata in $\mathcal{C}'_d$ obtained from $\mathcal{A}$ with a unique initial state chosen from amongst the initial states of $\mathcal{A}$ and a unique final state chosen from amongst the final states of $\mathcal{A}$. Since $u = v$ holds in $\mathcal{C}'_d$, we get:

$$[\![\mathcal{A}]\!](u) = \max_{\mathcal{B}\in\mathcal{S}} ([\![\mathcal{B}]\!](u)) = \max_{\mathcal{B}\in\mathcal{S}} ([\![\mathcal{B}]\!](v)) = [\![\mathcal{A}]\!](v)$$

and thus $u = v$ holds in $\mathcal{C}_d$. ◀

Weights. The set of identities which hold in $\mathcal{C}_d$ does not change when restricting the weights to have values in $\mathbb{N}$ or when allowing them to take values in $\mathbb{Q}$ or $\mathbb{R}$. Some directions are clear by definitions. We give ideas for the others.

From $\mathbb{Z}$ to $\mathbb{N}$. Consider an identity $u = v$ which holds in the class of d-state max-plus automata with weights in $\mathbb{N}$. It follows from the proof of Lemma 3 that $|u| = |v|$. Now let $\mathcal{A} \in \mathcal{C}_d$ and consider the max-plus automaton $\mathcal{A}_k$ obtained from $\mathcal{A}$ by adding the same integer k to the weight of all transitions in $\mathcal{A}$. Since $\mathcal{A}$ has finitely many transitions it is clear that we can choose k large enough so that $\mathcal{A}_k$ has weights in $\mathbb{N}$. Then we get, $[\![\mathcal{A}]\!](u) = [\![\mathcal{A}_k]\!](u) - k|u| = [\![\mathcal{A}_k]\!](v) - k|v| = [\![\mathcal{A}]\!](v)$, from which it follows that $u = v$ holds for all $\mathcal{A} \in \mathcal{C}_d$.

From $\mathbb{Q}$ to $\mathbb{Z}$. Consider an identity $u = v$ that holds in $\mathcal{C}_d$ and let $\mathcal{A}$ be a d-state max-plus automaton with weights in $\mathbb{Q}$. By multiplying all the weights on the transitions of $\mathcal{A}$ by a suitable non-zero integer k (*e.g.* the *lcm* of the denominators), we get a max-plus automaton $\mathcal{A}_k$ with weights in $\mathbb{Z}$, such that $[\![\mathcal{A}]\!](u) = \frac{1}{k}[\![\mathcal{A}_k]\!](u) = \frac{1}{k}[\![\mathcal{A}_k]\!](v) = [\![\mathcal{A}]\!](v)$.

From $\mathbb{R}$ to $\mathbb{Q}$. Consider an identity $u = v$ that holds in the class of d-state max-plus automata with weights in $\mathbb{Q}$ and let $\mathcal{A}$ be a d-state max-plus automaton with weights in $\mathbb{R}$. Let $(\mathcal{A}_m)_{m\in\mathbb{N}}$ be a sequence of max-plus automata constructed from $\mathcal{A}$ by changing all the real weights to rational weights in such a way that for every transition of $\mathcal{A}$ weighted by α, the sequence of weights $\alpha_m \in \mathbb{Q}$ of the corresponding transitions in $\mathcal{A}_m$ tends to α. Since limits can be commuted with maximum and sum over finite sets, we have:

$$[\![\mathcal{A}]\!](u) = \lim_{m\to\infty} [\![\mathcal{A}_m]\!](u) = \lim_{m\to\infty} [\![\mathcal{A}_m]\!](v) = [\![\mathcal{A}]\!](v)$$

Finally, we show that we need only to consider *full automata*. An automaton is said to be full if for every pair of states p, q and every letter a, there is a transition from p to q labelled by a.

▶ **Lemma 5.** *An identity $u = v$ holds in $\mathcal{C}_d$ if and only if it holds in the subclass of $\mathcal{C}_d$ consisting of full automata.*

Proof. The if direction is clear by definition. For the converse direction, consider an identity $u = v$ which holds in the subclass of $\mathcal{C}_d$ consisting of full automata. Suppose for contradiction that $\mathcal{A} \in \mathcal{C}_d$ is an automaton falsifying the identity. For each integer k, construct the full automaton $\mathcal{A}_k$ from $\mathcal{A}$ by adding in any missing transitions and weighting these by k. If $[\![\mathcal{A}]\!](u) = -\infty$ (meaning that there is no accepting run on u) then for all k, the accepting runs on u in $\mathcal{A}_k$ necessarily take a transition weighted by k (we suppose that $\mathcal{A}$ has at least one

initial and one final state). By assumption, $[\![\mathcal{A}]\!](v)$ must be finite (otherwise $\mathcal{A}$ does not falsify the identity) and by construction, necessarily for all k, $[\![\mathcal{A}_k]\!](v) \geqslant [\![\mathcal{A}]\!](v)$ (since $\mathcal{A}$ is contained in $\mathcal{A}_k$). Let us denote by m the maximal weight on a transition of $\mathcal{A}$. Then, consider k less than $[\![\mathcal{A}]\!](v) - (|u| - 1)m$. We get $[\![\mathcal{A}_k]\!](v) = [\![\mathcal{A}_k]\!](u) \leqslant k + (|u| - 1)m < [\![\mathcal{A}]\!](v)$, which leads to a contradiction. The same reasoning holds if $[\![\mathcal{A}]\!](v) = -\infty$. Otherwise, if $[\![\mathcal{A}]\!](u)$ and $[\![\mathcal{A}]\!](v)$ are both finite, and by considering k large and negative enough, $[\![\mathcal{A}]\!](u) = [\![\mathcal{A}_k]\!](u) = [\![\mathcal{A}_k]\!](v) = [\![\mathcal{A}]\!](u)$, since each maximal accepting run will avoid the transitions weighted by k. ◀

Number of letters. An identity on a 2-letter alphabet can be seen as an identity over a larger alphabet and it is easy to see that for all $k \geqslant 2$ the identity holds in the class of d-state max-plus automata over two letters if and only if it holds in the class of d-state max-plus automata over k letters. Suppose now that $u = v$ is an identity holding in $\mathcal{C}_d$ over an alphabet Σ containing at least three letters. Since u and v are distinct, they must differ in some position, i say. Suppose then that $u_i \neq v_i$. Now, consider $\bar{u}$ and $\bar{v}$ obtained from u and v by replacing every letter, except v_i, by u_i. By construction $\bar{u}$ and $\bar{v}$ are distinct. We are going to prove that $\bar{u} = \bar{v}$ holds in the class of max-plus automata over Σ. Indeed, consider a d-state max-plus automaton $\mathcal{A}$ over Σ. Construct first an automaton $\mathcal{A}'$ obtained from $\mathcal{A}$ by removing all the transitions not labelled by u_i or v_i. Then construct an automaton $\mathcal{B}$ over Σ obtained from $\mathcal{A}'$, by adding copies of the transitions labelled by u_i for all the other letters, except v_i, *i.e.* for every transition $p \xrightarrow{u_i \,:\, \alpha} q$, and every letter $c \neq v_i$, add the transition $p \xrightarrow{c \,:\, \alpha} q$. Then,

$$
\begin{aligned}
[\![\mathcal{A}]\!](\bar{u}) &= [\![\mathcal{A}']\!](\bar{u}) && \text{since } \bar{u} \text{ contains only } u_i\text{'s and } v_i\text{'s} \\
&= [\![\mathcal{B}]\!](\bar{u}) && \text{since } \bar{u} \text{ contains only } u_i\text{'s and } v_i\text{'s} \\
&= [\![\mathcal{B}]\!](u) && \text{since every letter } c \neq v_i \text{ mimics } u_i \text{ in } \mathcal{B} \\
&= [\![\mathcal{B}]\!](v) && \text{since } u = v \text{ is an identity over } \Sigma \text{ holding in } \mathcal{C}_d \\
&= [\![\mathcal{B}]\!](\bar{v}) && \text{since every letter } c \neq v_i \text{ mimics } u_i \text{ in } \mathcal{B} \\
&= [\![\mathcal{A}]\!](\bar{v}) && \text{since } \bar{v} \text{ contains only } u_i\text{'s and } v_i\text{'s}
\end{aligned}
$$

Thus, if an identity over Σ holds in $\mathcal{C}_d$ then an identity of the same length using just two letters must also hold in $\mathcal{C}_d$. Since we are interested in minimal length identities, in the rest of the paper we will consider only 2-letter alphabets.

3 Minimality

As explained at the end of the previous section, from now on we fix a 2-letter alphabet $\Sigma = \{a, b\}$. In this section, we provide a list of conditions which must all be satisfied by the identities holding in $\mathcal{C}$. Thanks to this list and aided by a computer, we are left with exactly two pairs of words (up to exchanging a and b) of length shorter than 18 which are still candidates to be identities in $\mathcal{C}$. In the next section, we prove that they are indeed identities in $\mathcal{C}$.

3.1 Triangular identities

A max-plus automaton with two states p and q is said to be *triangular* if there is no transition either from p to q or from q to p. We denote by $\mathcal{C}_\mathcal{T}$ this class of automata. An identity holding in $\mathcal{C}$ must also hold in $\mathcal{C}_\mathcal{T}$. Identities holding in the class of triangular automata

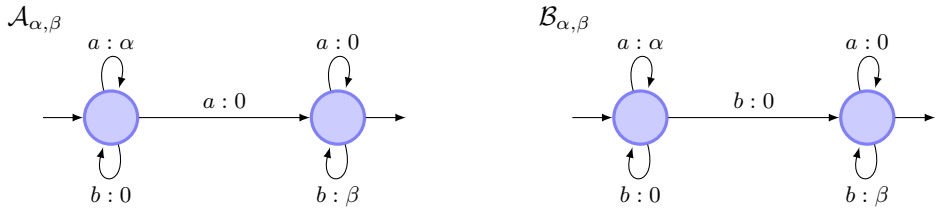
are much easier to study. They are fully characterised in [3], where it is proved that they are exactly the identities holding in the bicyclic monoid. More generally, several works [6, 7, 10, 1, 3] study identities holding in the class of triangular max-plus automata with d states, which correspond to the semigroup of upper-triangular matrices, where it has been proved that such an identity always exists.

Let us recall that if $u = v$ holds in $\mathcal{C}_\mathcal{T}$, then u and v have the same content (since the automata constructed in Lemma 3 are indeed triangular).

Beginning and end of a word. The *first* (*resp. second*, *last*, *penultimate*) *block* of a word w is the first (*resp.* second, last, penultimate) maximal block of the same consecutive letter of w. For example, for $w = a^3b^2a^6b^7a^4b$, these blocks are respectively a^3, b^2, b and a^4.

▶ **Lemma 6.** *If $u = v$ is an identity holding in $\mathcal{C}_\mathcal{T}$, then u and v have the same first, second, last and penultimate blocks respectively.*

Triangular identities. We give here a variant of a property in [3, Th 3.3 and Cor 3.4]. We show that the identities $u = v$ which hold in $\mathcal{C}_\mathcal{T}$ are exactly those such that u and v have the same content, the same first and last blocks, and which hold in the class of max-plus automata of one of the following shape, where α and β are integers either both positive or both negative:



If an identity holds for the class of all the max-plus automata of the form $\mathcal{A}_{\alpha,\beta}$ and $\mathcal{B}_{\alpha,\beta}$ for all integers α, β either both positive or both negative, the identity is said to be a *triangular identity*.

Checking triangular identities. Checking if a given identity $u = v$ is triangular can be done by symbolic computation using the shape of the automata above. More precisely, for any position i in a word w, we denote by $w_{<i}$ (*resp.* $w_{>i}$) the prefix of w strictly before position i (*resp.* the suffix of w strictly after position i). We get:

$$[\![\mathcal{A}_{\alpha,\beta}]\!](w) = \max_{w_i=a} (\alpha|w_{<i}|_a + \beta|w_{>i}|_b) \quad \text{and} \quad [\![\mathcal{B}_{\alpha,\beta}]\!](w) = \max_{w_i=b} (\alpha|w_{<i}|_a + \beta|w_{>i}|_b)$$

The identity $u = v$ is triangular if and only if for all integers α, β of the same sign, $[\![\mathcal{A}_{\alpha,\beta}]\!](u) = [\![\mathcal{A}_{\alpha,\beta}]\!](v)$ and $[\![\mathcal{B}_{\alpha,\beta}]\!](u) = [\![\mathcal{B}_{\alpha,\beta}]\!](v)$. It is proved in [3, Th 8.3] that it can be checked in polynomial time with respect to the sum of the lengths of u and v (even for a larger number of states). Another easy way to check a triangular identity in a reasonable time for identities of small length is to note that the parameters can be bounded:

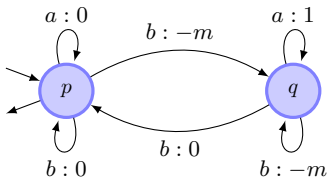
▶ **Lemma 7.** *Given two words u and v of the same length ℓ, $[\![\mathcal{A}_{\alpha,\beta}]\!](u) = [\![\mathcal{A}_{\alpha,\beta}]\!](v)$ (resp. $[\![\mathcal{B}_{\alpha,\beta}]\!](u) = [\![\mathcal{B}_{\alpha,\beta}]\!](v)$) holds for all integers α, β either both positive or both negative if and only if it holds for all such α, β with $|\alpha|$, $|\beta|$ bounded by $2\ell^2$.*

3.2 Block-permutation

Two words u and v are said to be *block-permuted* if u and v are composed of the same maximal blocks of the same consecutive letter but possibly in a different order. For example, $a^3b^2a^4b$ and $b^2a^3ba^4$ are block-permuted but $a^3b^2a^4b$ and a^2baba^4b are not.

▶ **Lemma 8.** *If $u = v$ is an identity which holds in $\mathcal{C}$, then u and v are block-permuted.*

Proof. Consider an identity $u = v$ which holds in $\mathcal{C}$. Suppose that u and v are not block-permuted, and that the maximal blocks of occurrences of the letter a are different (the proof for the letter b is similar). Let us write $n_1 \geqslant n_2 \geqslant \ldots \geqslant n_\ell$ for the lengths (with multiplicities) of the maximal blocks of consecutive a in u (*resp.* $m_1 \geqslant m_2 \geqslant \ldots \geqslant m_{\ell'}$ for v). By Lemma 3, u and v have the same content and so there must exist an index $i \in \{1, \ldots, \min(\ell, \ell')\}$ such that $n_j = m_j$ for all $j < i$, whilst $n_i \neq m_i$. Without loss of generality, suppose that $n_i > m_i$ and consider the following automaton where $m = n_i - 1$.



There are four options to read a word of the form ba^k (ignoring initial and final states for the moment): (1) around p with weight 0, (2) from p to q with weight $-m + k$, (3) around q with weight $-m + k$, or (4) from q to p with weight 0. Thus, if a maximal block of a is of length greater than m (except possibly the first or the last one), it should be read around q, otherwise, it should be read around p. By Lemma 6, u and v must have the same first and last blocks. For each $k = 1, \ldots, \ell$, let $N(k)$ be the set of indices from $1 \leqslant t \leqslant k$ such that a^{n_t} is not the first block of u and v, nor the last block of u and v. It is now easy to see that the weight of u must be greater than or equal to $\sum_{j \in N(i)}(n_j - m)$, while the weight of v is $\sum_{j \in N(i-1)}(n_j - m)$, which is smaller than the weight of u. Since this contradicts the fact that $u = v$ holds in $\mathcal{C}$, we conclude that $n_i = m_i$ for all i; or in other words, u and v are block-permuted. ◀

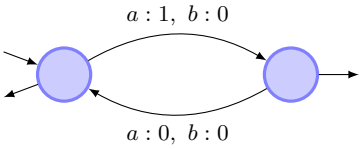
▶ **Corollary 9.** *If $u = v$ is an identity that holds in $\mathcal{C}$, then u and v each contain at least 7 maximal blocks of the same consecutive letter.*

Proof. Consider an identity $u = v$ with $u = a^{k_1}b^{k_2}a^{k_3}b^{k_4}a^{k_5}b^{k_6}$. If it holds in $\mathcal{C}$, then by Lemma 6, v must start with $a^{k_1}b^{k_2}$ and end with $a^{k_5}b^{k_6}$. Finally, by Lemma 8, necessarily u and v are the same word. ◀

3.3 Counting and parity conditions

Finally the last conditions we consider involve a finite number of max-plus automata with weights within $\{0, 1\}$ dealing in some sense with parity and counting conditions.

(C1). *The number of occurrences of the letter a in an even position.* This value is computed by the following automaton:



Note that, if two words have the same content, then the equality of this parameter for the two words implies the equality of all the other variants (number of b's in an odd position...). Indeed, the number of a's in an odd position is equal to the difference between the total number of a's and the number of a's in an even position. The number of b's in an even position is the difference between the total number of even positions and the number of a's in an even position.

(C2). *The number of occurrences of the letter a after an even number of b, and the number of occurrences of the letter b after an even number of a.* These values are computed respectively by the following automata:

As in the previous condition, providing two words have the same content, the equality on these parameters implies the equality on the other variants (number of a's after an odd number of b's, etc.). Indeed, the number of a's after an odd number of b's is equal to the difference between the total number of a's and the number of a's after an even number of b's...

The two last conditions are more difficult to explain.

(C3). We consider the following automata, and in each case the automata obtained such that exactly one of the states is both initial and final, as well as those obtained by exchanging a and b.

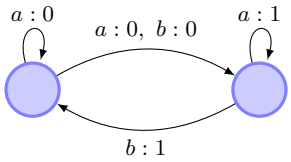
It can be checked that the words $ab^3ababa^3b^3a^2$ and $ab^3a^3babab^3a^2$ cannot be separated by any of the previously discussed conditions. However the automaton on the right, taking p to be both initial and final is able to do so, as we shall now show. The beginning of the two words are read deterministically until reaching the factor a^3. There, a non-deterministic choice is made to optimise the weight obtained by reading the end of the word. This choice is made at different positions in the two words leading to two different weights. More precisely, the maximal run for the word $ab^3ababa^3b^3a^2$ is as follows:

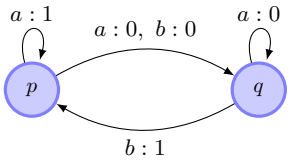
$$p \xrightarrow{ab^3\ :\ 1} p \xrightarrow{ababa\ :\ 0} \underbrace{q \xrightarrow{a^2\ :\ 0} p}_{\text{non-det choice}} \xrightarrow{b^3\ :\ 2} q \xrightarrow{a^2\ :\ 0} p$$

while the one for $ab^3a^3babab^3a^2$ is as follows:

$$p \xrightarrow{ab^3\,:\,1} \underbrace{p \xrightarrow{a^2\,:\,0} q}_{\text{non-det choice}} \xrightarrow{ababa\,:\,2} p \xrightarrow{b^3\,:\,2} q \xrightarrow{a^2\,:\,0} p$$

(C4). We consider the following automata, and in each case the automata obtained such that exactly one of the states is both initial and final, as well as those obtained by exchanging a and b.





The words $ab^2a^2ba^2ba^4b^3a$ and $ab^2a^4ba^2ba^2b^3a$ cannot be separated by any of the previously discussed conditions, whilst the automaton on the right, taking q to be both initial and final is able to do so. The beginning of the two words are read deterministically until reaching the factor a^2 in the middle of the two words. This determinism forces to read the two first blocks of a with weight 0, while the other ones will be read with weight 1. This leads to different results because of the commutation of the blocks a^2 and a^4 in the two words. More precisely, a maximal run for the word $ab^2a^2ba^2ba^4b^3a$ is as follows:

$$q \xrightarrow{ab^2a^2b\,:\,2} \underbrace{p \xrightarrow{a^2\,:\,1} q}_{\text{non-det choice}} \xrightarrow{b\,:\,1} p \xrightarrow{a^4\,:\,4} p \xrightarrow{b^3a\,:\,1} q$$

while the one for $ab^2a^4ba^2ba^2b^3a$ is as follows:

$$q \xrightarrow{ab^2a^4b\,:\,2} \underbrace{p \xrightarrow{a^2\,:\,1} q}_{\text{non-det choice}} \xrightarrow{b\,:\,1} p \xrightarrow{a^2\,:\,2} p \xrightarrow{b^3a\,:\,1} q$$

An identity $u = v$ is said to satisfy **(C1)**, **(C2)**, **(C3)** or **(C4)** if the same values is computed on u and v by the automata given above.

▶ **Proposition 10.** *There are exactly four triangular identities $u = v$ of length shorter than* 18 *satisfying* **(C1)**, **(C2)**, **(C3)** *and* **(C4)** *in which u and v are block-permuted and have the same first and last blocks.*

We have checked all these conditions assisted by a computer, with a program listing all the pairs of words not eliminated by one of these conditions.

Moreover, this list of conditions is in some sense minimal since for each of them, there are examples of pairs that are not eliminated when removing the condition from the list. These examples are also exhibited by our program.

We remark that if the block-permutation condition holds then the only automata we need to consider which involve weights not within $\{0, 1\}$ are the ones corresponding to the triangular conditions. This list of conditions is probably not sufficient to characterise fully the identities which hold in $\mathcal{C}$, however, one can ask if we can extend it and keep this distinction between the triangular conditions with arbitrary weights and the other conditions involving only weights in $\{0, 1\}$.

There are exactly four remaining candidates:

$$a^2b^3a^3babab^3a^2 = a^2b^3ababa^3b^3a^2, \quad ab^3a^4baba^2b^3a = ab^3a^2baba^4b^3a$$

and the ones obtained by exchanging the roles of a and b. In the next section, we prove that they indeed hold in $\mathcal{C}$.

4 The shortest identities

In this section, we conclude the proof of Theorem 1 by proving that the remaining candidate identities hold in $\mathcal{C}$. By exchanging the role of a and b, it is sufficient to prove the following proposition:

▶ **Proposition 11.** *The following two identities hold in $\mathcal{C}$:*

(I1) $a^2b^3a^3babab^3a^2 = a^2b^3ababa^3b^3a^2$ *and* **(I2)** $ab^3a^4baba^2b^3a = ab^3a^2baba^4b^3a$

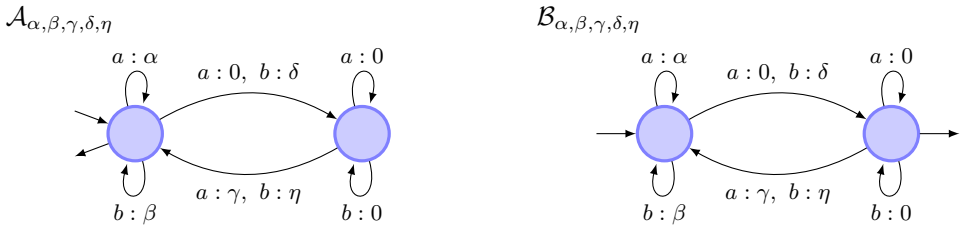
For a word $u = a_0 \cdots a_\ell$ of length $\ell + 1$, let us denote by $\tilde{u} = a_\ell \cdots a_0$ the *reverse* of u.

▶ **Lemma 12.** *Let $u \in \Sigma^+$. If $[\![\mathcal{A}]\!](u) \geqslant [\![\mathcal{A}]\!](\tilde{u})$ for all $\mathcal{A}$ in $\mathcal{C}$, then $u = \tilde{u}$ is an identity which holds in $\mathcal{C}$.*

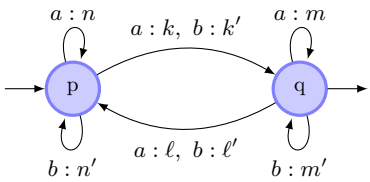
Proof. Consider an automaton $\mathcal{A}$ in $\mathcal{C}$. By hypothesis, $[\![\mathcal{A}]\!](u) \geqslant [\![\mathcal{A}]\!](\tilde{u})$. Construct now $\mathcal{B}$ obtained from $\mathcal{A}$ by reversing the transitions, *i.e.* there is a transition $p \xrightarrow{c\,:\,\alpha} q$ in $\mathcal{A}$ if and only if there is a transition $q \xrightarrow{c\,:\,\alpha} p$ in $\mathcal{B}$. Moreover the initial (*resp.* final) states of $\mathcal{B}$ are defined from the final (*resp.* initial) states of $\mathcal{A}$. By this construction, $[\![\mathcal{A}]\!](\tilde{u}) = [\![\mathcal{B}]\!](u) \geqslant [\![\mathcal{B}]\!](\tilde{u}) = [\![\mathcal{A}]\!](u)$. Thus $[\![\mathcal{A}]\!](\tilde{u}) = [\![\mathcal{A}]\!](u)$ for all $\mathcal{A}$ in $\mathcal{C}$ and hence $u = \tilde{u}$ holds in $\mathcal{C}$. ◀

Remark that the two identities **(I1)** and **(I2)** are of the form $u = \tilde{u}$.

▶ **Lemma 13.** *Given two words u and v of the same content, $[\![\mathcal{A}]\!](u) \geqslant [\![\mathcal{A}]\!](v)$ for all $\mathcal{A}$ in $\mathcal{C}$ if and only if $[\![\mathcal{B}]\!](u) \geqslant [\![\mathcal{B}]\!](v)$ for all $\mathcal{B}$ of one of the following two forms, where α, β, γ, δ, η are integers:*



Proof. The if direction is clear by definition. Conversely, denote by $\mathcal{C}'$ the class of automata described in the statement of the proposition. Suppose that $[\![\mathcal{B}]\!](u) \geqslant [\![\mathcal{B}]\!](v)$ for all $\mathcal{B} \in \mathcal{C}'$. Consider $\mathcal{A} \in \mathcal{C}$. First, by the proof of Lemma 5, we can suppose that $\mathcal{A}$ is full and by Lemma 4, that $\mathcal{A}$ has exactly one initial and one final state. Suppose that these two states are different. If not, a similar reasoning will hold, involving $\mathcal{A}_{\alpha,\beta,\gamma,\delta,\eta}$ instead of $\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}$. We represent $\mathcal{A}$ in the following picture:



First, construct $\mathcal{A}'$ from $\mathcal{A}$ by removing m (*resp.* m') from all the weights of the transitions labelled by a (*resp.* b). Then construct $\mathcal{B}$ from $\mathcal{A}'$ by removing $k - m$ from the weights of

the transitions labelled by a and b from p to q and adding $k-m$ from the weights of the transitions labelled by a and b from q to p. By construction, $\mathcal{B}$ is in $\mathcal{C}'$. We get:

$$\begin{aligned}
[\![\mathcal{A}]\!](u) &= [\![\mathcal{A}']\!](u) + m'|u|_b + m|u|_a && \text{by construction}\\
&= [\![\mathcal{B}]\!](u) + (k-m) + m'|u|_b + m|u|_a && \text{since } p \text{ is initial and } q \text{ final, thus on an}\\
&&& \text{accepting run, the transitions from}\\
&&& p \text{ to } q \text{ are taken (in total) exactly once}\\
&&& \text{more than the transitions from } q \text{ to } p\\
&\geqslant [\![\mathcal{B}]\!](v) + (k-m) + m'|v|_b + m|v|_a && \text{since } \mathcal{B} \text{ is in } \mathcal{C}' \text{ and}\\
&&& u \text{ and } v \text{ have the same content}\\
&\geqslant [\![\mathcal{A}']\!](v) + m'|v|_b + m|v|_a && \text{for the same reason as above}\\
&\geqslant [\![\mathcal{A}]\!](v) && \text{by construction}
\end{aligned}$$

◀

Let us consider **(I1)** and denote $u = a^2b^3a^3babab^3a^2$. By Lemmas 12 and 13, for proving that **(I1)** holds in $\mathcal{C}$, it is sufficient to prove that for all integers $\alpha, \beta, \gamma, \delta, \eta$, $[\![\mathcal{A}_{\alpha,\beta,\gamma,\delta,\eta}]\!](u) \geqslant [\![\mathcal{A}_{\alpha,\beta,\gamma,\delta,\eta}]\!](\tilde{u})$ and $[\![\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}]\!](u) \geqslant [\![\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}]\!](\tilde{u})$. Consider $\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}$. Aided by computer, we compute symbolically the values on u and on its reverse in $\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}$ as a tropical polynomial in α, β, γ, δ, η. Each monomial term corresponds to the weight of an accepting (but not necessarily maximal weight) run. For example, the monomial term $8\alpha + 8\beta$ corresponds to an accepting run on u (when reading u around the initial state and going to the final state on the last transition) and in fact on $\tilde{u}$ also. We denote by M_u, $M_{\tilde{u}}$ and M the set of monomials appearing only in the computation of u, only in the computation of $\tilde{u}$ or for both, respectively. We compute these three sets aided by a computer.

Finally, we prove that for each monomial in $M_{\tilde{u}}$ and each choice of parameters $\alpha, \beta, \gamma, \delta, \eta \in \mathbb{Z}$, there is a monomial in $M_u \cup M$ which is greater on the values $\alpha, \beta, \gamma, \delta, \eta$. This concludes the proof that $[\![\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}]\!](u) \geqslant [\![\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}]\!](\tilde{u})$ for all integers $\alpha, \beta, \gamma, \delta, \eta$. To do so, there is no need to consider the monomials in $M_{\tilde{u}}$ in which neither γ nor η appears. Indeed, we already checked in the previous section that **(I1)** satisfies the triangular conditions. Thus, $u = \tilde{u}$ holds for triangular automata. Consider $\mathcal{B}'_{\alpha,\beta,\delta}$ constructed from $\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}$ by removing the transitions from the final state to the initial state. A monomial in $M_{\tilde{u}}$ in which neither γ nor η appears corresponds to an accepting run in $\mathcal{B}'_{\alpha,\beta,\delta}$, and hence is bounded above by $\mathcal{B}'_{\alpha,\beta,\delta}(\tilde{u}) = \mathcal{B}'_{\alpha,\beta,\delta}(u)$, since this automaton is triangular. The latter is clearly bounded above by $\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}(u)$. So for all monomials in $M_{\tilde{u}}$ in which neither γ nor η and each choice of parameters $\alpha, \beta, \gamma, \delta, \eta \in \mathbb{Z}$, there is necessarily a monomial in $M_u \cup M$ which is greater on the values $\alpha, \beta, \gamma, \delta, \eta$. Finally, the set $M_{\tilde{u}}$ without these monomials is of reasonable size and we are able to complete the computations by hand.

Similar computations hold for $\mathcal{A}_{\alpha,\beta,\gamma,\delta,\eta}$ and for **(I2)**.

5 Conclusion

In this paper, we give the shortest identities which hold in the class of max-plus automata with two states. We hope that a better understanding of this case is a first step towards a better understanding of the general case. In particular, we give an interesting list of conditions which are sufficient to achieve this goal. Future works will consist in trying to understand better these conditions and how to extend this list to fully characterise the sets of identities for max-plus automata with two states. In particular, we remark that under the

block-permutation condition, the only automata we need to consider which involve weights not within $\{0, 1\}$ are the ones corresponding to the triangular conditions. We ask if we can generalise this list of conditions to get the shortest identities for a larger number of states, and keep this distinction between the triangular conditions with arbitrary weights and the other conditions involving only weights in $\{0, 1\}$.

References

1. Y. Chen, X. Hu, Y. Luo, and O. Sapir. The finite basis problem for the monoid of two-by-two upper triangular tropical matrices. *Bull. Aust. Math. Soc.*, 94(1):54–64, 2016. doi:10.1017/S0004972715001483.
2. T. Colcombet, L. Daviaud, and F. Zuleger. Size-change abstraction and max-plus automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2014. doi:10.1007/978-3-662-44522-8_18.
3. L. Daviaud, M. Johnson, and M. Kambites. Identities in upper triangular tropical matrix semigroups and the bicyclic monoid, 2017. preprint, http://arxiv.org/abs/1612.04219.
4. S. Gaubert. Performance evaluation of $(\max, +)$ automata. *IEEE Trans. Automat. Control*, 40(12):2014–2025, 1995. doi:10.1109/9.478227.
5. S. Gaubert and J. Mairesse. Modeling and analysis of timed Petri nets using heaps of pieces. *IEEE Trans. Automat. Control*, 44(4):683–697, 1999. doi:10.1109/9.754807.
6. Z. Izhakian. Semigroup identities in the monoid of triangular tropical matrices. *Semigroup Forum*, 88(1):145–161, 2014. doi:10.1007/s00233-013-9507-6.
7. Z. Izhakian. Erratum to: Semigroup identities in the monoid of triangular tropical matrices [MR3164156]. *Semigroup Forum*, 92(3):733, 2016. doi:10.1007/s00233-016-9790-0.
8. Z. Izhakian. Semigroup identities of tropical matrix semigroups of maximal rank. *Semigroup Forum*, 92(3):712–732, 2016. doi:10.1007/s00233-015-9765-6.
9. Z. Izhakian and S. W. Margolis. Semigroup identities in the monoid of two-by-two tropical matrices. *Semigroup Forum*, 80(2):191–218, 2010. doi:10.1007/s00233-009-9203-8.
10. J. Okniński. Identities of the semigroup of upper triangular tropical matrices. *Comm. Algebra*, 43(10):4422–4426, 2015. doi:10.1080/00927872.2014.946141.
11. J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
12. M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
13. Y. Shitov. A semigroup identity for tropical 3×3 matrices, 2014. To appear in Ars Mathematica Contemporanea 14 (2018), 15–23.

On the Upward/Downward Closures of Petri Nets*

Mohamed Faouzi Atig[1], Roland Meyer[†2], Sebastian Muskalla[3], and Prakash Saivasan[4]

1 **Uppsala University, Sweden**
mohamed_faouzi.atig@it.uu.se
2 **TU Braunschweig, Germany**
roland.meyer@tu-bs.de
3 **TU Braunschweig, Germany**
s.muskalla@tu-bs.de
4 **TU Braunschweig, Germany**
p.saivasan@tu-bs.de

Abstract

We study the size and the complexity of computing finite state automata (FSA) representing and approximating the downward and the upward closure of Petri net languages with coverability as the acceptance condition. We show how to construct an FSA recognizing the upward closure of a Petri net language in doubly-exponential time, and therefore the size is at most doubly exponential. For downward closures, we prove that the size of the minimal automata can be non-primitive recursive. In the case of BPP nets, a well-known subclass of Petri nets, we show that an FSA accepting the downward/upward closure can be constructed in exponential time. Furthermore, we consider the problem of checking whether a simple regular language is included in the downward/upward closure of a Petri net/BPP net language. We show that this problem is EXPSPACE-complete (resp. NP-complete) in the case of Petri nets (resp. BPP nets). Finally, we show that it is decidable whether a Petri net language is upward/downward closed.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Petri nets, BPP nets, downward closure, upward closure

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.49

1 Introduction

Petri nets are a popular model of concurrent systems [14]. Petri net languages (with different acceptance conditions) have been extensively studied during the last years, including deciding their emptiness (which can be reduced to reachability) [31, 23, 25, 26], their regularity [39, 11], their context-freeness [38, 27], and many other decision problems (e.g. [17, 2, 15]). In this paper, we consider the class of Petri net languages with coverability as the acceptance condition (i.e. the set of sequences of transition labels occurring in a computation reaching a marking greater than or equal to a given final marking).

We address the problem of computing the *downward* and the *upward* closure of Petri net languages. The downward closure of a language L, denoted by $L{\downarrow}$, is the set of all subwords, all words that can be obtained from words in L by deleting letters. The upward closure of L, denoted by $L{\uparrow}$, is the set of all superwords, all words that can be obtained from words in

* The full version is available as technical report [7].
† A part of this work was carried out when the author was at Aalto University.

L by inserting letters. It is well-known that, for any language, the downward and upward closure are regular and can be described by a *simple regular expression (SRE)*. However, such an expression is in general not computable, e.g. for example, it is not possible to compute the downward closure of languages recognized by lossy channel systems [33].

In this paper, we first consider the problem of constructing a finite state automaton (FSA) accepting the upward/downward closure of a Petri net language. We give an algorithm that computes an FSA of doubly-exponential size for the upward closure in doubly-exponential time. This is done by showing that every minimal word results from a computation of length at most doubly exponential in the size of the input. Our algorithm is also optimal since we present a family of Petri net languages for which the minimal finite state automata representing their upward closure are of doubly-exponential size.

Our second contribution is a family of Petri net languages for which the size of the minimal finite state automata representing the downward closure is non-primitive recursive: The languages contain Ackermann many words. The downward closure of Petri net languages has been shown to be effectively computable [17]. The algorithm is based on the Karp-Miller tree [22], which has non-primitive recursive complexity.

Furthermore, we consider the SRE inclusion problem which asks whether the language of a simple regular expression is included in the downward/upward closure of a Petri net language. The idea behind SRE inclusion is to stratify the problem of computing the downward/upward closure: Rather than having an algorithm computing all information about the language, we imagine to have an oracle (e.g. an enumeration) making proposals for SREs that could be included in the downward/upward closure. The task of the algorithm is merely to check whether a proposed inclusion holds. We show that this problem is EXPSPACE-complete in both cases. In the case of upward closures, we prove that SRE inclusion boils down to checking whether the set of minimal words of the given SRE is included in the upward closure. In the case of downward closures, we reduce the problem to the simultaneous unboundedness problem for Petri nets, which is EXPSPACE-complete [11].

We also study the problem of checking whether a Petri net language actually is upward or downward closed. This is interesting as it means that an automaton for the closure, which we can compute with the aforementioned methods, is a precise representation of the system's behavior. We show that the problem of being upward/downward closed is decidable for Petri nets. The result is a consequence of a more general decidability that we believe is of independent interest. We show that checking whether a regular language is included in a Petri net language (with coverability as the acceptance condition) is decidable. Here, we rely on a decision procedure for trace inclusion due to Esparza et al. [21].

Finally, we consider *BPP*[1] *nets* [13], a subclass of Petri nets defined by a syntactic restriction: Every transition is allowed to consume at most one token in total. We show that we can compute finite state automata accepting the upward and the downward closure of a BPP net language in exponential time. The size of the FSA is also exponential. Our algorithms are optimal as we present a family of BPP net languages for which the minimal FSA representing their upward/downward closure have exponential size. Furthermore, we consider the SRE inclusion problem. We show that, in the case of BPP nets, it is NP-complete for both, inclusion in the upward and in the downward closure. To prove the upper bound, we reduce to the satisfiability problem for existential Presburger arithmetic (which is known to be NP-complete [37]). The hardness is by a reduction from SAT to the emptiness of BPP net languages, which in turn reduces to SRE inclusion.

[1] BPP stands for *basic parallel processes*, a notion from process algebra.

Related Work. Several constructions have been proposed in the literature to compute finite state automata recognizing the downward/upward closure. In the case of Petri net languages (with various acceptance conditions including reachability), it has been shown that the downward closure is effectively computable [17]. With the results in this paper, the computation and the state complexity have to be non-primitive recursive. For the languages generated by context-free grammars, effective computability of the downward closure is due to [40, 16, 10, 8]. For the languages recognized by one-counter automata, a strict subclass of the context-free languages, it has been shown how to compute in polynomial time a finite state automaton accepting the downward/upward closure of the language [6]. The effective computability of the downward closure has also been shown for stacked counter automata [43]. In [42], Zetzsche provides a characterization for a class of languages to have an effective downward closure. It has been used to prove the effective computability of downward closures of higher-order pushdown automata and higher-order recursion schemes [18, 9]. The downward closure of the languages of lossy channel systems is not computable [33].

The computability results discussed above have been used to prove the decidability of verification problems and to develop approximation-based program analysis methods (see e.g. [5, 4, 3, 24, 30, 44]). Throughout the paper, we will give hints to applications in verification.

2 Preliminaries

In this section, we fix some basic definitions and notations that will be used throughout the paper. For every $i, j \in \mathbb{N}$ with $i \leq j$, we use $[i..j]$ to denote the set $\{k \in \mathbb{N} \mid i \leq k \leq j\}$ (resp. $[i..j[$ for $\{k \in \mathbb{N} \mid i \leq k < j\}$). Let Σ be a finite alphabet. We use Σ_ε to denote $\Sigma \cup \{\varepsilon\}$. The length of a word u over Σ is denoted by $|u|$, where $|\varepsilon| = 0$. Let $k \in \mathbb{N}$ be a natural number, we use Σ^k (resp. $\Sigma^{\leq k}$) to denote the set of all words of length equal (resp. smaller or equal) to k. A language L over Σ is a (possibly infinite) set of finite words.

Let Γ be a subset of Σ. Given a word $u \in \Sigma^*$, we denote by $\pi_\Gamma(u)$ the projection of u over Γ, i.e. the word obtained from u by erasing all the letters that are not in Γ.

The *Parikh image* of a word [34] counts the number of occurrences of all letters while forgetting about their positioning. Formally, the function $\Psi : \Sigma^* \mapsto \mathbb{N}^\Sigma$ takes a word $w \in \Sigma^*$ and gives the function $\Psi(w) : \Sigma \to \mathbb{N}$ defined by $(\Psi(w))(a) = \left|\pi_{\{a\}}(w)\right|$ for all $a \in \Sigma$.

The *subword relation* $\preceq \subseteq \Sigma^* \times \Sigma^*$ [20] between words is defined as follows: A word $u = a_1 \dots a_n$ is a subword of v, denoted $u \preceq v$, if u can be obtained by deleting letters from v or, equivalently, if $v = v_0 a_1 v_1 \dots a_n v_n$ for some $v_0, \dots, v_n \in \Sigma^*$.

Let L be a language over Σ. The *upward closure* of L consists of all words that have a subword in the language, $L{\uparrow} = \{v \in \Sigma^* \mid \exists u \in L : u \preceq v\}$. The *downward closure* of L contains all words that are dominated by a word in the language, $L{\downarrow} = \{u \in \Sigma^* \mid \exists v \in L : u \preceq v\}$. Higman showed that the subword relation is a well-quasi ordering [20], which means that every set of words $S \subseteq \Sigma^*$ has a finite *basis* — a finite set of *minimal elements* $v \in S$ such that $\nexists u \in S : u \neq v, u \preceq v$. With finite bases, $L{\uparrow}$ and $L{\downarrow}$ are guaranteed to be regular for every language $L \subseteq \Sigma^*$ [19]. Indeed, they can be expressed using the subclass of simple regular languages defined by so-called *simple regular expressions* [1]. These SREs are choices among *products* p, $sre ::= p \mid sre + sre$. Products interleave single letters a or $(a + \varepsilon)$ with iterations over letters from subsets $\Gamma \subseteq \Sigma$ of the alphabet: $p ::= a \mid (a + \varepsilon) \mid \Gamma^* \mid p.p$. The syntactic size of an SRE sre is denoted by $|sre|$ and defined as expected.

Finite State Automata. A *finite state automaton (FSA)* A is a tuple $(Q, \to, q_0, Q_f, \Sigma)$ where Q is a finite non-empty set of states, Σ is the finite input alphabet, $\to \subseteq Q \times \Sigma_\varepsilon \times Q$ is the non-deterministic transition relation, $q_0 \in Q$ is the initial state, and $Q_f \subseteq Q$ is the set of

final states. We represent a transition $(q, a, q') \in \rightarrow$ by $q \xrightarrow{a}_A q'$ and generalize the relation to words in the expected way. The language of finite words accepted by A is denoted by $L(A)$. The size of A, denoted $|A|$, is defined by $|Q| + |\Sigma|$. An FSA is minimal for its language $L(A)$ if there is no FSA B with $L(A) = L(B)$ with a strictly smaller number of states.

Petri Nets. A *(labeled) Petri net* is a tuple $N = (\Sigma, P, T, F, \lambda)$ [36]. Here, Σ is a finite alphabet, P a finite set of *places*, T a finite set of *transitions*, $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ a *flow function*, and $\lambda : T \mapsto \Sigma_\varepsilon$ a labelling function. When convenient, we will assume that the places are ordered, $P = [1..\ell]$ for some $\ell \in \mathbb{N}$. For a place or transition $x \in P \cup T$, we define the *preset* to consist of the elements that have an arc to x, ${}^\bullet x = \{y \in P \cup T \mid F(y, x) > 0\}$. The *postset* is defined similarly, $x^\bullet = \{y \in P \cup T \mid F(x, y) > 0\}$.

To define the semantics of Petri nets, we use *markings* $M : P \to \mathbb{N}$ that assign to each place a number of *tokens*. A marking M *enables* a transition t, denoted $M[t\rangle$, if $M(p) \geq F(p, t)$ for all $p \in P$. A transition t that is enabled may be *fired*, leading to the new marking M' defined by $M'(p) = M(p) - F(p, t) + F(t, p)$ for all $p \in P$, i.e. t consumes $F(p, t)$ many tokens and produces $F(t, p)$ many tokens in p. We write the firing relation as $M[t\rangle M'$. A *computation* $\pi = M_0[t_1\rangle M_1 \cdots [t_m\rangle M_m$ consists of markings and transitions. We extend the firing relation to transition sequences $\sigma \in T^*$ in the straightforward manner and also write $\pi = M_0[\sigma\rangle M_m$. A marking M is *reachable* from an initial marking M_0 if $M_0[\sigma\rangle M$ for some $\sigma \in T^*$. A marking M covers another marking M_f, denoted $M \geq M_f$, if $M(p) \geq M_f(p)$ for all $p \in P$. A marking M_f is *coverable* from M_0 if there is a marking M reachable from M_0 that covers M_f, $M_0[\sigma\rangle M \geq M_f$ for some $\sigma \in T^*$.

Given a Petri net N, an initial marking M_0, and a final marking M_f, the associated *covering language* is $L(N, M_0, M_f) = \{\lambda(\sigma) \mid \sigma \in T^*,\ M_0[\sigma\rangle M \geq M_f\}$, where the labeling function λ is extended to sequences of transitions in the straightforward manner. Given a natural number $k \in \mathbb{N}$, we define $L_k(N, M_0, M_f) = \left\{\lambda(\sigma) \;\middle|\; \sigma \in T^{\leq k},\ M_0[\sigma\rangle M \geq M_f\right\}$ to be the set of words accepted by computations of length at most k.

Let $max(F)$ denote the maximum of the range of F. The size of the Petri net N is $|N| = |\Sigma| + \cdot |P| \cdot |T| \cdot (1 + \lceil log_2(1 + max(F)) \rceil)$, i.e. we assume that the flow function is encoded in binary. Similarly, the size of a marking M is $|M| = |P| \cdot (1 + \lceil log_2(1 + max(M)) \rceil)$, where $max(M)$ denotes the maximum of the range of M. We define the *token count* $tc(M) = \Sigma_{p \in P} M(p)$ of a marking M to be the sum of all tokens assigned by M.

A Petri net N is said to be a *BPP net* if every transition consumes at most one token from one place (i.e. $\Sigma_{p \in P} F(p, t) \leq 1$ for every $t \in T$).

3 Upward Closures

We consider the problem of constructing a finite state automaton accepting the upward closure of a Petri net and a BPP net language, respectively. The upward closure offers an over-approximation of the system behavior that is useful for verification purposes [30].

Petri Nets. We prove a doubly-exponential upper bound on the size of the finite state automaton representing the upward closure of a Petri net language. Then, we present a family of Petri net languages for which the minimal finite state automata representing their upward closure have a size doubly exponential in the size of the input.

Upper Bound. Fix the Petri net $N = (\Sigma, P, T, F, \lambda)$ and let M_0 and M_f be the initial and the final marking of interest. Define $n = |N| + |M_0| + |M_f|$.

▶ **Theorem 1.** *One can construct an FSA of size* $O(2^{2^{\text{poly}(n)}})$ *for* $L(N, M_0, M_f)\uparrow$.

The remainder of the section is devoted to proving the theorem. We will show that every minimal word results from a computation of length at most $O(2^{2^{\text{poly}(n)}})$. Let us call such computations the minimal ones. Let k be a bound on the length of the minimal computations. This means the language $L_k(N, M_0, M_f)$ contains all minimal words of $L(N, M_0, M_f)$. Furthermore, $L_k(N, M_0, M_f) \subseteq L(N, M_0, M_f)$ and therefore the equality $L_k(N, M_0, M_f)\uparrow = L(N, M_0, M_f)\uparrow$ holds. Now we can use the following lemma to construct a finite automaton whose size is $O(2^{2^{\text{poly}(|n|)}})$ and that accepts $L_k(N, M_0, M_f)$. Without an increase in size, this automaton can be modified to accept $L_k(N, M_0, M_f)\uparrow$.

▶ **Lemma 2.** *Consider* N, M_0, *and* M_f. *For every* $k \in \mathbb{N}$, *we can construct an FSA of size* $O((k+2)^{\text{poly}(n)})$ *that accepts* $L_k(N, M_0, M_f)$, *where* $n = |N| + |M_0| + |M_f|$.

It remains to show that every minimal word results from a computation of length at most doubly exponential in the size of the input. This is the following proposition.

▶ **Proposition 3.** *For every computation* $M_0[\sigma\rangle M \geq M_f$, *there is* $M_0[\sigma'\rangle M' \geq M_f$ *with* $\lambda(\sigma') \preceq \lambda(\sigma)$ *and* $|\sigma'| \leq 2^{2^{cn \log n}}$, *where* c *is a constant.*

Our proof is an adaptation of Rackoff's technique to show that coverability can be solved in EXPSPACE [35]. Rackoff derives a bound (similar to ours) on the length of the shortest computations that cover a given marking. Rackoff's proof has been generalized to different settings, e.g. to BVAS in [12]. Lemma 5.3 in [28] claims that Rackoff's original proof already implies Proposition 3. This is not true as we provide a counterexample in the full version of this paper [7]. To handle labeled Petri nets, his proof needs two amendments. First, it is not sufficient to consider the shortest covering computations. Instead, we have to consider computations long enough to generate all minimal words. Second, Rackoff's proof splits a firing sequence into two parts and replaces the second part by a shorter one. In our case, we need that the shorter word is a subword of the original one.

We now elaborate on Rackoff's proof strategy and give the required definitions, then we explain in more detail our adaptation, and finally give the technical details.

We assume that the places are ordered, i.e. $P = [1..\ell]$. Rackoff's idea is to relax the definition of the firing relation and allow for negative token counts on the last $i+1$ to ℓ places. With a recurrence over the number of places, he then obtains a bound on the length of the computations that keep the first i places positive. Formally, an *unrestricted marking* of N is a function $M : P \to \mathbb{Z}$. An unrestricted marking M *i-enables* a transition $t \in T$ if $M(j) \geq F(j,t)$ for all $j \in [1..i]$. Firing t yields a new unrestricted marking M', denoted $M[t\rangle_i M'$, with $M'(p) = M(p) - F(p,t) + F(t,p)$ for all $p \in P$. A computation $\pi = M_0[t_1\rangle_i M_1 \dots [t_m\rangle_i M_m$ is *i-bounded* with $i \in [1..\ell]$ if for each marking M_k with $k \in [0..m]$ and each place $j \in [1..i]$, we have $M_k(j) \geq 0$. We assume a fixed marking M_f to be covered. The computation π is *i-covering* (wrt. M_f) if $M_m(j) \geq M_f(j)$ for all $j \in [1..i]$. Given two computations $\pi_1 = M_0[t_1\rangle_i \cdots [t_k\rangle_i M_k$ and $\pi_2 = M'_0[t'_1\rangle_i \cdots [t'_s\rangle_i M'_s$ such that $M_k(j) = M'_0(j)$ for all $j \in [1..i]$, we define their *i-concatenation* $\pi_1 \cdot_i \pi_2$ to be the computation $M_0[t_1\rangle_i \cdots [t_k\rangle_i M_k[t'_1\rangle_i M''_{k+1} \cdots [t'_s\rangle_i M''_{k+s}$.

Rackoff's result provides a bound on the length of the shortest i-covering computations. Since we have to generate all minimal words, we will specify precisely which computations to consider (not only the shortest ones). Moreover, Rackoff's bound holds independent of the initial marking. This is needed, because the proof of the main lemma splits a firing sequence into two parts and then considers the starting marking of the second part as the new initial

marking. The sets we define in the following will depend on some unrestricted initial marking M, but we then quantify over all possible markings to get rid of the dependency.

Let $\mathit{Paths}(M,i)$ be the set of all paths of all i-covering and i-bounded computations starting at M, i.e. $\mathit{Paths}(M,i) = \{\sigma \in T^* \mid \pi = M[\sigma\rangle_i M', \pi \text{ is } i\text{-bounded and } i\text{-covering}\}$. Let $\mathit{Words}(M,i) = \{\lambda(\sigma) \mid \sigma \in \mathit{Paths}(M,i)\}$ be the corresponding set of words, and let $\mathit{Basis}(M,i) = \{w \in \mathit{Words}(M,i) \mid w \text{ is } \preceq\text{-minimal}\}$ be its minimal elements. The central definitions is $\mathit{SPath}(M,i)$, the set of shortest paths yielding the minimal words in $\mathit{Basis}(M,i)$,

$$\mathit{SPath}(M,i) = \left\{ \sigma \in \mathit{Paths}(M,i) \;\middle|\; \begin{array}{l} \lambda(\sigma) \in \mathit{Basis}(M,i), \\ \nexists\, \sigma' \in \mathit{Paths}(M,i)\colon \quad |\sigma'| < |\sigma|, \lambda(\sigma') = \lambda(\sigma) \end{array} \right\} .$$

Define $m(M,i) = \max\{|\sigma| + 1 \mid \sigma \in \mathit{SPath}(M,i)\}$ to be the length (+1) of the longest path in $\mathit{SPath}(M,i)$, or $m(M,i) = 0$ if $\mathit{SPath}(M,i)$ is empty. Note that $\mathit{Basis}(M,i)$ is finite and therefore only finitely many different lengths occur for sequences in SPath, i.e. $m(M,i)$ is well-defined. To remove the dependency on M, define $f(i) = \max\{m(M,i) \mid M \in \mathbb{Z}^\ell\}$ to be the maximal length of an i-covering computation, where the maximum is taken over all unrestricted initial markings. The well-definedness of $f(i)$ is not clear yet and will be a consequence of the next lemma. A bound on $f(\ell)$ will give us a bound on the maximum length of a computation accepting a minimal word from $L(N, M_0, M_f)$. To derive the bound, we prove that $f(i+1) \leq (2^n f(i))^{i+1} + f(i)$ using Rackoff's famous case distinction [35].

▶ **Lemma 4.** *$f(0) = 1$ and $f(i+1) \leq (2^n f(i))^{i+1} + f(i)$ for all $i \in [0..\ell[$.*

Lower Bound. We present a family of Petri net languages for which the minimal finite state automata representing the upward closure are of size doubly exponential in the size of the input. We rely on a construction due to Lipton [29] that shows how to calculate in a precise way (including zero tests) with values up to 2^{2^n} in Petri nets.

▶ **Lemma 5.** *For every number $n \in N$, we can construct a Petri net $N(n) = (\{a\}, P, T, F, \lambda)$ and markings M_0, M_f of size polynomial in n such that $L(N(n), M_0, M_f) = \{a^{2^{2^n}}\}$.*

BPP Nets. We establish an exponential upper bound on the size of the finite automata representing the upward closure of BPP net languages. Then, we present a family of BPP net languages for which the minimal finite automata representing their upward closure are of size at least exponential in the size of the input.

Upper Bound. Fix the BPP net $N = (\Sigma, P, T, F, \lambda)$ and assume M_0 and M_f to be the initial and the final marking of interest. Let $n = |N| + |M_0| + |M_f|$.

▶ **Theorem 6.** *One can construct an FSA of size $O(2^{\mathrm{poly}(n)})$ for $L(N, M_0, M_f)\uparrow$.*

We will show that every minimal word results from a computation whose length is polynomially dependent on the number of transitions and on the number of tokens in the final marking (which may be exponential in the size of the input). Let k be a bound on the length of the minimal computations. With the same argument as before and using Lemma 2, we can construct a finite state automaton of size $O(2^{\mathrm{poly}(n)})$ that accepts $L_k(N, M_0, M_f)\uparrow$.

▶ **Proposition 7.** *Consider a BPP net N. For every computation $M_0[\sigma\rangle M \geq M_f$ there is $M_0[\sigma'\rangle M' \geq M_f$ with $\lambda(\sigma') \preceq \lambda(\sigma)$ and $|\sigma'| \leq tc(M_f)^2|T|$.*

The key to proving the lemma is to consider a structure that makes the concurrency among transitions in the BPP computation of interest explicit. Phrased differently, we give a true concurrency semantics (also called partial order semantics and similar to Mazurkiewicz traces) to BPP computations. Since BPPs do not synchronize, the computation yields a forest where different branches represent causally independent transitions. To obtain a subcomputation that covers the final marking, we select from the forest a set of leaves that corresponds exactly to the final marking. We then show that the number of transitions in the minimal forest that generates the selected set of leaves is polynomial in the number of tokens in the final marking and in the number of transitions.

Lower Bound. We present a family of BPP net languages for which the minimal finite state automata representing the upward closure are exponential in the size of the input. The idea is to rely on the final marking, which is encoded in binary and hence can require 2^n tokens.

▶ **Lemma 8.** *For all numbers $n \in \mathbb{N}$, we can construct a BPP net $N(n) = (\{a\}, P, T, F, \lambda)$ and markings M_0, M_f of size polynomial in n such that $L(N(n), M_0, M_f) = \{a^{2^n}\}$.*

4 Downward Closures

We consider the problem of constructing a finite state automaton accepting the downward closure of a Petri net and a BPP net language, respectively. The downward closure often has the property of being a precise description of the system behavior, namely as soon as asynchronous communication comes into play: If the components are not tightly coupled, they may overlook commands of the partner and see precisely the downward closure of the other's computation. As a result, having a representation of the downward closure gives the possibility to design exact or under-approximate verification algorithms.

Petri Nets. The downward closure of Petri net languages has been shown to be effectively computable in [17]. The algorithm is based on the Karp-Miller tree [22], which can be of non-primitive recursive size. We now present a family of Petri net languages that are already downward closed and for which the minimal finite automata have to be of non-primitive recursive size in the size of the input. Our result relies on a construction due to Mayr and Meyer [32]. It gives a family of Petri nets whose computations all terminate but, upon halting, may have produced Ackermann many tokens on a distinguished place.

▶ **Lemma 9.** *For all $n, x \in \mathbb{N}$, there is a Petri net $N(n) = (\{a\}, P, T, F, \lambda)$ and markings $M_0^{(x)}, M_f$ of size polynomial in $n + x$ such that $L(N(n), M_0^{(x)}, M_f) = \{a^k \mid k \leq Acker_n(x)\}$.*

Our lower bound is an immediate consequence of this lemma.

▶ **Theorem 10.** *There is a family of Petri net languages for which the minimal finite automata representing the downward closure are of non-primitive recursive size.*

This hardness result relies on a weak computation mechanism of very large numbers that is unlikely to show up in practical examples. The SRE inclusion problem studied in the following section can be understood as a refined analysis of the computation problem for downward closures.

BPP Nets. We prove an exponential upper bound on the size of the finite automata representing the downward closure of BPP languages. Then, we present a family of BPP languages for which the minimal finite automata representing their downward closure are exponential in the size of the input BPP nets.

Upper Bound. Fix the BPP net $N = (\Sigma, P, T, F, \lambda)$ and let M_0 and M_f be the initial and the final marking of interest. Let $n = |N| + |M_0| + |M_f|$.

▶ **Theorem 11.** *We can construct a finite automaton of size* $O(2^{\text{poly}(n)})$ *for* $L(N, M_0, M_f)\downarrow$.

The key insight for simulating N by a finite automaton is the following: If during a firing sequence a marking occurs that has more than c tokens (where c is specified below) in some place p, then there has to be a *pump*, a subsequence of the firing sequence that can be repeated to produce arbitrarily many tokens in p. The precise statement is this, where we use $m = max(F)$ to refer to the maximal multiplicity of an edge.

▶ **Lemma 12.** *Let* $M_0[\sigma\rangle M$ *such that for some place* $p \in P$*, we have* $M(p) > c$ *with* $c = tc(M_0)(|P| \cdot m)^{(|T|+1)}$ *Then for each* $j \in \mathbb{N}$*, there is* $M_0[\sigma_j\rangle M_j$ *such that (1)* $\sigma \preceq \sigma_j$*, (2)* $M \leq M_j$*, and (3)* $M_j(p) > j$*.*

The automaton for $L(N, M_0, M_f)\downarrow$ is the state space of N with token values beyond c set to ω. For every transition, we also have an ε-variant to obtain the downward closure. The language of this automaton is the downward closure of the language of the given BPP net.

Lower Bound. Consider the family of BPP net languages from the Lemma 8: $L(N(n), M_0, M_f) = \{a^{2^n}\}$, for all $n \in \mathbb{N}$. Its downward closure is $\{a^i \mid i \leq 2^n\}$. The minimal finite state automata recognising the downward closure have at least 2^n states.

5 SRE Inclusion in Downward Closure

The downward closure of a Petri net language is hard to compute. We therefore propose to under-approximate it by an SRE as follows. Assumev we have a heuristic coming up with a candidate SRE that is supposed to be an under-approximation in the sense that its language is included in the downward closure of interest. The problem we study is the algorithmic task of checking whether the inclusion indeed holds. If so, the SRE provides reliable (must) information about the system's behavior, behavior that is guaranteed to occur. This information is useful for finding bugs.

SRE Inclusion in Downward Closure (SRED)	
Given:	A Petri net (N, M_0, M_f), an SRE sre.
Question:	$L(sre) \subseteq L(N, M_0, M_f)\downarrow$?

Petri Nets.

▶ **Theorem 13.** SRED *is* EXPSPACE*-complete for Petri nets.*

Hardness is due to the hardness of coverability [29]. Indeed, marking M_f is coverable from M_0 in N iff $L(N, M_0, M_f) \neq \emptyset$ iff $\{\varepsilon\} \subseteq L(N, M_0, M_f)\downarrow$. Note that $\{\varepsilon\} = L(\emptyset^*)$ and therefore is a simple regular language.

For the upper bound, we take inspiration from a recent result of Zetzsche [42]. He has shown that, for a large class of models, computing the downward closure is equivalent to deciding an unboundedness problem. We use a variant of this problem that comes with a complexity result. The *simultaneous unboundedness problem for Petri nets* (SUPPN) is, given a Petri net N, an initial marking M_0, and a subset $X \subseteq P$ of places, decide whether for each $n \in \mathbb{N}$, there is a computation σ_n such that $M_0[\sigma_n\rangle M_{\sigma_n}$ with $M_{\sigma_n}(p) \geq n$ for all places $p \in X$. In [11], Demri has shown that this problem is EXPSPACE-complete.

▶ **Theorem 14** ([11]). SUPPN *is* EXPSPACE*-complete.*

We turn to the reduction of the inclusion problem SRED to the unboundedness problem SUPPN. Since SREs are choices among products, an inclusion $L(sre) \subseteq L(N, M_0, M_f)\downarrow$ holds iff $L(p) \subseteq L(N, M_0, M_f)\downarrow$ holds for all products p in *sre*. Since the Petri net language is downward closed, we can further simplify the products by removing choices. Fix a total ordering on the alphabet Σ. Such an ordering can be represented by a word w_Σ. We define the *linearization operation* that takes a product and returns a regular expression:

$$lin(a + \varepsilon) = a \qquad lin(a) = a$$
$$lin(\Gamma^*) = (\pi_\Gamma(w_\Sigma))^* \qquad lin(p_1 p_2) = lin(p_1)lin(p_2)\ .$$

For example, if $\Sigma = \{a, b, c\}$ and we take $w_\Sigma = abc$, then $p = (a + c)^*(a + \varepsilon)(b + c)^*$ is turned into $lin(p) = (ac)^* a (bc)^*$. The discussion justifies the following lemma.

▶ **Lemma 15.** *$L(sre) \subseteq L(N, M_0, M_f)\downarrow$ if and only if for all products p in sre we have $L(lin(p)) \subseteq L(N, M_0, M_f)\downarrow$.*

We will reduce $L(lin(p)) \subseteq L(N, M_0, M_f)\downarrow$ to SUPPN. To this end, we first understand $lin(p)$ as a Petri net $N_{lin(p)}$. We modify this Petri net by adding one place p_Γ for each block $(\pi_\Gamma(w_\Sigma))^* = a_i \dots a_j$. Each transition that repeats or leaves the block is modified to generate a token in p_Γ. As a result, p_Γ counts how often the word $\pi_\Gamma(w_\Sigma)$ has been executed.

The second step is to define an appropriate product of $N_{lin(p)}$ with the Petri net of interest. Intuitively, the product synchronizes with the downward closure of N.

▶ **Definition 16.** Consider two Petri nets $N_i = (\Sigma, P_i, T_i, F_i, \lambda)$, $i = 1, 2$, with $P_1 \cap P_2 = \emptyset$ and $T_1 \cap T_2 = \emptyset$. Their *right-synchronized product* $N_1 \rtimes N_2$ is the labeled Petri net $N_1 \rtimes N_2 = (\Sigma, P_1 \cup P_2, T_1 \cup T, F, \lambda)$, where for the transitions $t_1 \in T_1$, λ and F remain unchanged. The new transitions are $T = \{merge(t_1, t_2) \mid t_1 \in T_1, t_2 \in T_2, \lambda_1(t_1) = \lambda_2(t_2)\}$ with $\lambda(merge(t_1, t_2)) = \lambda_1(t_1) = \lambda_2(t_2)$ and similarly $F(p_i, merge(t_1, t_2)) = F_i(p_i, t_i)$, $F(merge(t_1, t_2), p_i) = F_i(t_i, p_i)$ for $p_i \in P_i$, $i = 1, 2$.

As indicated by the name *right-synchronized*, the transitions of N_1 can be fired without synchronization, while the transitions of N_2 can only be fired if a transition of N_1 with the same label is fired simultaneously.

Consider a Petri net N with initial marking M_0. We compute the right-synchronized product $N' = N \rtimes N_{lin(p)}$, take the initial marking M_0' that coincides with M_0 but puts a token on the initial place of $N_{lin(p)}$, and focus on the counting places $X = \{p_\Gamma \mid (\pi_\Gamma(w_\Sigma))^*\}$ is a block in p. The following correspondence holds.

▶ **Lemma 17.** *$L(lin(p)) \subseteq L(N, M_0, M_\emptyset)\downarrow$ if and only if the places in X are simultaneously unbounded in N' from M_0'. Here $M_\emptyset$ is the zero marking, i.e. $M_\emptyset(p) = 0$ for all p.*

The lemma does not yet involve the final marking M_f. We modify N' and X such that simultaneous unboundedness implies $L(lin(p)) \subseteq L(N, M_0, M_f)\downarrow$. The idea is to introduce a new place p_f that can become unbounded only after M_f has been covered. We furthermore add a transition t_f that consumes $M_f(p)$ tokens from each place p of N and produces one token in p_f. We add another transition t_{pump} that consumes one token in p_f and creates two tokens in p_f. Call the resulting net N''. The new initial marking M_0'' coincides with M_0' and assigns no token to p_f.

Note that we do not enforce that t_f is only fired after all the rest of the computation has taken place. We can rearrange the transitions in any valid firing sequence of N'' to obtain a sequence of the shape $\sigma.{t_f}^k.{t_{pump}}^{k'}$, where σ contains neither t_f nor t_{pump}.

▶ **Lemma 18.** *$L(lin(p)) \subseteq L(N, M_0, M_f)\downarrow$ iff the places in $X \cup \{p_f\}$ are simultaneously unbounded in N'' from M_0''.*

To conclude the proof of Theorem 13, it remains to argue that the generated instance for SUPPN is polynomial in the input, i.e. in N, M_0, M_f and p. The expression $lin(p)$ is certainly linear in p, and the net $N_{lin(p)}$ is polynomial in $lin(p)$. The blow-up caused by the right-synchronized product is at most quadratic, and adding the transitions and the places to deal with M_f is polynomial. The size of M_0'' is polynomial in the size of M_0 and p. Altogether, the size of N'', $X \cup \{p_f\}$, and M_0'' (which together form the generated instance for SUPPN) is polynomial in the size of the original input.

BPP Nets. We show that the problem of deciding whether the language of an SRE is included in the downward closure of a BPP net language is NP-complete.

▶ **Theorem 19.** SRED *for BPP nets is* NP*-complete.*

Hardness is by a reduction from SAT to BPP coverability, which in turn reduces to deciding whether the language of an SRE is included in the downward closure of a BPP language. For the reverse direction, we give a reduction to satisfiability of an existential formula in *Presburger arithmetic*, the first-order theory of the natural numbers with addition, subtraction, and order. An *existential* Presburger formula takes the form $\exists x_1 \dots \exists x_n . \varphi$ where φ is a quantifier-free formula. We shall also write positive Boolean combinations of existential formulas. By an appropriate renaming of the quantified variables, any such formula can be converted into an equivalent existential Presburger formula. We write $\varphi(\vec{x})$ to indicate that (at most) the variables $\vec{x} = x_1 \dots x_k$ occur free in φ. Given a function M from $\vec{x}$ to $\mathbb{N}$, the meaning of M *satisfies* φ is as usual and we write $M \models \varphi$ to denote this. We rely on the following complexity result:

▶ **Theorem 20** ([37])**.** *Satisfiability in existential Presburger arithmetic is* NP*-complete.*

Note that $L(sre) \subseteq L(N, M_0, M_f)\downarrow$ iff the inclusion holds for every product p in sre. Given such a product, we construct a new BPP net N' and an existential Presburger formula $\psi(P')$ such that $L(p) \subseteq L(N, M_0, M_f)\downarrow$ iff there is a marking M' reachable in N' from a modified initial marking M_0' with $M' \models \psi$. This concludes the proof with the help of the following characterization of reachability in BPP nets in terms of existential Presburger arithmetic.

▶ **Theorem 21** ([41, 13])**.** *Given a BPP net $N = (\Sigma, P, T, F, \lambda)$ and an initial marking M_0, one can compute in polynomial time an existential Presburger formula $\Psi(P)$ so that for all markings M: $M \models \Psi(P)$ if and only if $M_0[\sigma\rangle M$ for some $\sigma \in T^*$.*

Key to the construction of N' is a characterization of the computations that need to be present in the BPP net for the inclusion $L(p) \subseteq L(N, M_0, M_f)\downarrow$ to hold. Wlog., in the following we will assume that the product takes the shape

$$(a_1 + \varepsilon)\Sigma_1^*(a_2 + \varepsilon) \dots \Sigma_{n-1}^*(a_n + \varepsilon),$$

where $\Sigma_1, \dots, \Sigma_{n-1} \subseteq \Sigma$ and $a_1, \dots, a_n \in \Sigma$. For this language to be included in $L(N, M_0, M_f)\downarrow$, the BPP should have a computation with parts σ_i containing a_i and parts ρ_i between the σ_i that contain all letters in Σ_i and that can be repeated. To formalize the requirement, recall that we use w_Σ for a total order on the alphabet and $\pi_{\Sigma_i}(w_\Sigma)$ for the projection to $\Sigma_i \subseteq \Sigma$.

Moreover, we define $M \leq^c M'$, with c the constant defined in Lemma 12, if for all places $p \in P$ we have $M'(p) < c$ implies $M(p) \leq M'(p)$.

▶ **Definition 22.** Let p be a product. The BPP net N together with the markings M_0, M_f admits a *p-witness* if there are markings $M_1, M_1', \ldots, M_n, M_n'$ and computations σ_i, ρ_i that satisfy $M_i[\sigma_i\rangle M_i'$ for all $i \in [1..n]$, $M_i'[\rho_i\rangle M_{i+1}$ for all $i \in [1..n[$, and moreover: (1) $a_i \preceq \lambda(\sigma_i)$, for all $i \in [1..n]$, (2) $\pi_{\Sigma_i}(w_\Sigma) \preceq \lambda(\rho_i)$ and $M_i' \leq^c M_{i+1}$, for all $i \in [1..n-1]$, and (3) $M_1 = M_0$ and $M_f \leq^c M_n'$.

In a p-witness, (1) enforces that the a_i occur in the desired order, and the first part of (2) requires that $\pi_{\Sigma_i}(w_\Sigma)$ occurs in between. The second part of (2) means that each ρ_i (and thus $\pi_{\Sigma_i}(w_\Sigma)$) can be repeated. Property (3) enforces that the computation still starts in the initial marking and can be extended to cover the final marking.

The following proposition reduces the problem SRED for BPP nets to checking whether the BPP admits a p-witness.

▶ **Proposition 23.** *$L(p) \subseteq L(N, M_0, M_f)\downarrow$ holds iff (N, M_0, M_f) admits a p-witness.*

We now reduce the problem of finding such a p-witness to finding in another BPP net $N' = (\emptyset, P', T', F', \lambda')$ a reachable marking that satisfies a Presburger formula $\Psi_{M_f}^{M_0}(P')$. The task is to identify $2n$ markings that are related by $2n-1$ computations as required by a p-witness. The idea is to create $2n-1$ replicas of the BPP net and run them independently to guess the corresponding computations σ_i resp. ρ_i. The Presburger formula $\Psi_{M_f}^{M_0}$ will check that the target marking reached with σ_i coincides with the initial marking for ρ_i, and the target marking reached with ρ_i is the initial marking of σ_{i+1}. To this end, the net N' remembers the initial marking that each replica started from in a full copy (per replica) of the set of places of the BPP net. Furthermore $\Psi_{M_f}^{M_0}$ checks that each ρ^i can be repeated by ensuring that the final marking in the corresponding replica is larger than the initial marking. As initial marking for N', we consider the marking $M_\emptyset$ with $M_\emptyset(p) = 0$ for all p.

▶ **Proposition 24.** *There are σ' and M' so that $M_\emptyset[\sigma'\rangle M'$ in N' and $M' \models \Psi_{M_f}^{M_0}$ if and only if (N, M_0, M_f) admits a p-witness.*

6 SRE Inclusion in Upward Closure

Rather than computing the upward closure of a Petri net language we now check whether a given SRE under-approximates it. Formally, the problem is defined as follows.

SRE Inclusion in Upward Closure (SREU)	
Given:	A Petri net (N, M_0, M_f), SRE sre.
Question:	$L(sre) \subseteq L(N, M_0, M_f)\uparrow$?

▶ **Theorem 25.** SREU *is* EXPSPACE*-complete for Petri nets.*

The EXPSPACE lower bound is immediate by hardness of coverability for Petri nets [29]. The upper bound is due to the following fact: We only need to check whether the set of minimal words in the language of the given SRE is included in the upward closure of the Petri net language. Since the number of minimal words in the SRE language is less than the size of the SRE, and since checking whether a word is included in the upward closure of the language of the Petri net N can be reduced in polynomial time to coverability in Petri nets (which is well-known to be in EXPSPACE [35]), we obtain our EXPSPACE upper bound.

▶ **Theorem 26.** SREU *is* NP*-complete for BPP nets.*

The hardness is by a reduction of the coverability problem for BPP nets. For the upper bound, the algorithm is similar to the one for checking the inclusion of an SRE in the downward closure of a BPP language. Consider a product p of the given SRE. The inclusion $L(p) \subseteq L(N, M_0, M_f)\uparrow$ holds iff the minimal subwords $\min(p) = a_1 \dots a_n$ belong to $L(N, M_0, M_f)\uparrow$. Word $a_1 \dots a_n$ belongs to the upward closure iff one of its subwords is in $L(N, M_0, M_f)$. We reduce this check for an accepted subword to checking whether a reachable marking M in a different net N' satisfies a Presburger formula Ψ.

7 Being Upward/Downward Closed

We now study the problem to decide whether a Petri net language actually is upward or downward closed, i.e. whether the closure that we can compute is actually a precise representation of the system's behavior. Formally, the problem BUC is defined as follows:

Being upward closed (BUC)	
Given:	A Petri net (N, M_0, M_f).
Question:	$L(N, M_0, M_f) = L(N, M_0, M_f)\uparrow$?

The problem of being downward closed (BDC) replaces $\uparrow$ by $\downarrow$ in the above definition.

▶ **Theorem 27.** BUC *and* BDC *are decidable for Petri nets.*

Note that $L(N, M_0, M_f) \subseteq L(N, M_0, M_f)\uparrow$ trivially holds. It remains to decide the converse. First, we show how to decide $L(A) \subseteq L(N, M_0, M_f)$ for any given FSA A. This regular inclusion should be a problem of independent interest. Then, we can use the automaton for the upward closure constructed by Theorem 1 (resp. the automaton for the downward closure that can be constructed by [17]) to decide BUC (resp. BDC).

We rely on a result of Esparza et. al [21] that involves the *traces* of an FSA (resp. Petri net), labelings of computations that start from the initial state (resp. initial marking), regardless of whether they end in a final state (resp. covering marking). For a finite automaton A , we define $\mathcal{T}(A) = \{w \in \Sigma^* \mid q_{init} \xrightarrow{w} q \text{ for some } q \in Q\}$. Similarly, for a Petri net, we define $\mathcal{T}(N, M_0) = \{w \in \Sigma^* \mid \exists\, \sigma \in T^* \colon \lambda(\sigma) = w, M_0[\sigma\rangle M \text{ for some marking } M\}$.

▶ **Theorem 28** ([21]). *The inclusion $\mathcal{T}(A) \subseteq \mathcal{T}(N, M_0)$ is decidable.*

The algorithm constructs a computation tree of A and N. This tree determinizes N in that it tracks sets of incomparable markings reachable with the same trace. The construction terminates if either the set of markings becomes empty and the inclusion fails or (the automaton deadlocks or) we find a set of markings that covers a predecessor and the inclusion holds. The latter is guaranteed to happen due to the well-quasi ordering (wqo) of sets of markings. This dependence on wqos does not allow us to derive a complexity result.

We now show how to reduce checking the inclusion $L(A) \subseteq L(N, M_0, M_f)$ to deciding an inclusion among trace languages. Theorem 28 can be used to decide this inclusion. Let (N, M_0, M_f) be the Petri net of interest together with its initial and final marking, and let A be the given FSA. As language $L(N, M_0, M_f)$ is not prefix-closed in general, we consider the zero marking $M_\emptyset$ as the new final marking. This yields a prefix-closed language with $\mathcal{T}(N, M_0) = L(N, M_0, M_\emptyset)$, since now all valid firing sequences give a word in the language, and prefixes of valid firing sequences are again valid firing sequences. We still need to take the original final marking M_f into account. To do so, we modify the net by adding a new transition that can only be fired after M_f has been covered.

Let $a \notin \Sigma$ be a fresh letter. Let $N.a$ be the Petri net that is obtained from N and the given final marking M_f by adding a new transition t_{final} that consumes $M_f(p)$ many tokens from every place p of N and that is labeled by a. For the automaton, we use a similar trick. Let $A.a$ be an automaton for $L(A).a$ that is reduced so that the unique final state is reachable from every state.

▶ **Lemma 29.** *$L(A) \subseteq L(N, M_0, M_f)$ holds iff $\mathcal{T}(A.a) \subseteq \mathcal{T}(N.a, M_0)$ holds.*

The second inclusion is decidable using Theorem 28. This yields the desired result.

▶ **Theorem 30.** *$L(A) \subseteq L(N, M_0, M_f)$ is decidable.*

References

1. P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *FMSD*, 25(1), 2004.
2. P. A. Abdulla, G. Delzanno, and L. V. Begin. Comparing the expressive power of well-structured transition systems. In *CSL*, LNCS. Springer, 2007.
3. M. F. Atig, A. Bouajjani, K. Narayan Kumar, and P. Saivasan. On bounded reachability analysis of shared memory systems. In *FSTTCS*, LIPIcs. Dagstuhl, 2014.
4. M. F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *LMCS*, 7(4), 2011.
5. M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, LNCS. Springer, 2008.
6. M. F. Atig, D. Chistikov, P. Hofman, K. N. Kumar, P. Saivasan, and G. Zetzsche. Complexity of regular abstractions of one-counter languages. In *LICS*, pages 207–216. ACM, 2016.
7. M. F. Atig, R. Meyer, S. Muskalla, and P. Saivasan. On the upward/downward closures of petri nets. *CoRR*, abs/1701.02927, 2017. URL: `http://arxiv.org/abs/1701.02927`.
8. G. Bachmeier, M. Luttenberger, and M. Schlund. Finite automata for the sub- and superword closure of CFLs: Descriptional and computational complexity. In *LATA*, LNCS. Springer, 2015.
9. L. Clemente, P. Parys, S. Salvati, and I. Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In *LICS*, pages 96–105. ACM, 2016.
10. B. Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 1991.
11. S. Demri. On selective unboundedness of VASS. *JCSS*, 79(5), 2013.
12. S. Demri, M. Jurdziński, O. Lachish, and R. Lazić. The covering and boundedness problems for branching vector addition systems. *Journal of Computer and System Sciences*, 79(1):23–38, 2013.
13. J. Esparza. Petri Nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inf.*, 31(1), 1997.
14. J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Bulletin of the EATCS*, 52, 1994.
15. A. Finkel, G. Geeraerts, J. F. Raskin, and L. V. Begin. On the omega-language expressive power of extended Petri nets. *ENTCS*, 2005.
16. H. Gruber, M. Holzer, and M. Kutrib. More on the size of Higman-Haines sets: Effective constructions. In *MCU*, LNCS. Springer, 2007.
17. P. Habermehl, R. Meyer, and H. Wimmel. The downward-closure of Petri net languages. In *ICALP*, LNCS. Springer, 2010.
18. M. Hague, J. Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In *POPL*. ACM, 2016.

19 L. H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1), 1969.

20 G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.* (3), 2(7), 1952.

21 P. Jančar, J. Esparza, and F. Moller. Petri nets and regular processes. *J. Comput. Syst. Sci.*, 59(3):476–503, December 1999.

22 R. M. Karp and R. E. Miller. Parallel program schemata. *JCSS*, 3(2):147–195, 1969.

23 S. R. Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *STOC*. ACM, 1982.

24 S. La Torre, A. Muscholl, and I. Walukiewicz. Safety of parametrized asynchronous shared-memory systems is almost always decidable. In *CONCUR*, LIPIcs. Dagstuhl, 2015.

25 J. L. Lambert. A structure to decide reachability in Petri nets. *TCS*, 99(1), 1992.

26 J. Leroux. Vector addition system reachability problem: a short self-contained proof. In *POPL*. ACM, 2011.

27 J. Leroux, V. Penelle, and G. Sutre. On the context-freeness problem for vector addition systems. In *LICS*. IEEE, 2013.

28 J. Leroux, M. Praveen, and G. Sutre. A relational trace logic for vector addition systems with application to context-freeness. In *CONCUR*, pages 137–151. Springer, 2013.

29 R. J. Lipton. The reachability problem requires exponential space. Technical report, Yale University, Department of Computer Science, 1976.

30 Z. Long, G. Calin, R. Majumdar, and R. Meyer. Language-theoretic abstraction refinement. In *FASE*, LNCS. Springer, 2012. doi:10.1007/978-3-642-28872-2_25.

31 E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comp.*, 13(3), 1984.

32 E. W. Mayr and A. R. Meyer. The complexity of the finite containment problem for Petri nets. *JACM*, 28(3), 1981.

33 R. Mayr. Undecidable problems in unreliable computations. *TCS*, 1-3(297), 2003.

34 R. Parikh. On context-free languages. *JACM*, 13(4), 1966.

35 C. Rackoff. The covering and boundedness problems for vector addition systems. *TCS*, 6(2), 1978.

36 W. Reisig. *Petri nets: An Introduction*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1985.

37 B. Scarpellini. Complexity of subcases of Presburger arithmetic. *Transactions of the AMS*, 284(1), 1984.

38 S. R. Schwer. The context-freeness of the languages associated with vector addition systems is decidable. *TCS*, 1992.

39 R. Valk and G. Vidal-Naquet. Petri nets and regular languages. *JCSS*, 23(3), 1981.

40 J. van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21(3), 1978.

41 K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational Horn clauses. In *CADE*, pages 337–352. Springer, 2005.

42 G. Zetzsche. An approach to computing downward closures. In *ICALP*, LNCS. Springer, 2015.

43 G. Zetzsche. Computing downward closures for stacked counter automata. In *STACS*, LIPIcs. Dagstuhl, 2015.

44 G. Zetzsche. The complexity of downward closure comparisons. In *ICALP*, volume 55 of *LIPIcs*, pages 123:1–123:14. Dagstuhl, 2016.

On Multidimensional and Monotone k-SUM

Chloe Ching-Yun Hsu[1] and Chris Umans*[2]

1 **Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, USA**
chhsu@caltech.edu

2 **Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, USA**
umans@cms.caltech.edu

Abstract

The well-known k-SUM conjecture is that integer k-SUM requires time $\Omega(n^{\lceil k/2\rceil - o(1)})$. Recent work has studied multidimensional k-SUM in $\mathbb{F}_p^d$, where the best known algorithm takes time $\tilde{O}(n^{\lceil k/2\rceil})$. Bhattacharyya et al. [ICS 2011] proved a $\min(2^{\Omega(d)}, n^{\Omega(k)})$ lower bound for k-SUM in $\mathbb{F}_p^d$ under the Exponential Time Hypothesis. We give a more refined lower bound under the standard k-SUM conjecture: for sufficiently large p, k-SUM in $\mathbb{F}_p^d$ requires time $\Omega(n^{k/2-o(1)})$ if k is even, and $\Omega(n^{\lceil k/2\rceil - 2k\frac{\log k}{\log p} - o(1)})$ if k is odd.

For a special case of the multidimensional problem, *bounded monotone d-dimensional 3SUM*, Chan and Lewenstein [STOC 2015] gave a surprising $\tilde{O}(n^{2-2/(d+13)})$ algorithm using additive combinatorics. We show this algorithm is essentially optimal. To be more precise, bounded monotone d-dimensional 3SUM requires time $\Omega(n^{2-\frac{4}{d}-o(1)})$ under the standard 3SUM conjecture, and time $\Omega(n^{2-\frac{2}{d}-o(1)})$ under the so-called strong 3SUM conjecture. Thus, even though one might hope to further exploit the structural advantage of monotonicity, no substantial improvements beyond those obtained by Chan and Lewenstein are possible for bounded monotone d-dimensional 3SUM.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems

Keywords and phrases 3SUM, kSUM, monotone 3SUM, strong 3SUM conjecture

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.50

1 Introduction

The k-SUM problem and the k-SUM conjecture are related to a large number of problems in computational geometry [11], dynamic data structures, and graph theory. For example, Pătraşcu [17] showed lower bounds for dynamic problems under the 3SUM conjecture, and Kopelowitz, Pettie, and Porat [16] improved Pătraşcu's framework to give better reductions from 3SUM to SetIntersection, SetDisjointness, and triangle enumeration. Goldstein, Kopelowitz, Lewenstein, and Porat [13] showed several reporting problems are 3SUM-hard. Vassilevska and Williams [19], and Jafargholi and Viola [15] used 3SUM to study triangle problems. Abboud and Lewi [1] proved tight lower and upper bounds for the exact-weight subgraph finding problem under the k-SUM conjecture.

▶ **Definition 1** (k-SUM)**.** Given subsets $A_1, \ldots, A_k$ of size n of an abelian group G, the k-SUM problem asks whether there are $a_1 \in A_1, \ldots, a_k \in A_k$ such that $\sum_{i=1}^{k} a_i = 0$.

* Supported by NSF grant CCF-1423544 and a Simons Foundation Investigator grant.

A simple meet-in-the-middle algorithm can solve k-SUM in time $\tilde{O}(n^{\lceil k/2 \rceil})$,[1] and it is widely believed that this is the optimal time up to polylogarithmic factors. This is known as the k-SUM conjecture:

▶ **Conjecture 2** (k-SUM Conjecture). *For $k \geq 2$, k-SUM in $\mathbb{Z}$ requires randomized time $\Omega(n^{\lceil k/2 \rceil - o(1)})$.*

To support the k-SUM conjecture, Erickson [9] and Ailon and Chazelle [3] proved that k-linear decision trees cannot solve k-SUM with fewer than $n^{\lceil k/2 \rceil}$ queries. On the other hand, Freund [10], and Gold and Sharir [12] recently gave $O(n^2 \log\log n / \log n)$ algorithms for 3SUM. Hence, the standard 3SUM conjecture (Conjecture 7) is stated as an $\Omega(n^{2-o(1)})$ lower bound instead of $\Omega(n^2)$.

Intriguingly, in non-uniform models, substantially lower complexities are known: Grønlund and Pettie [14] showed that the decision tree complexity of 3SUM is $O(n^{3/2} \log n)$. Gold and Sharir [12] showed that the randomized $(2k-2)$-linear decision tree complexity of k-SUM is $O(n^{k/2})$ for any odd $k \geq 3$.

1.1 Multidimensional k-SUM in $\mathbb{F}_p^d$

One can also consider the k-SUM problem over domains other than $\mathbb{Z}$. The focus of this paper will be on the multidimensional case, where the domain is $\mathbb{F}_p^d$. The k-SUM problem in $\mathbb{F}_p^d$ is a problem of independent interest. For example, Jafargholi and Viola [15, 20] reduced listing triangles to 3SUM in $\mathbb{F}_2^d$. In coding theory, k-SUM in $\mathbb{F}_p^d$ is studied and known as WeightDistribution [8].

Bhattacharyya et. al. [6] recently gave a $\min(2^{\Omega(d)}, n^{\Omega(k)})$ lower bound for k-SUM in $\mathbb{F}_p^d$ under the Exponential Time Hypothesis. Pătraşcu and Williams [18] proved that the Exponential Time Hypothesis implies a weak version of the k-SUM conjecture - there is no $n^{o(k)}$ algorithm for k-SUM for all k. However, prior to this paper, no connection was known between integer k-SUM and k-SUM in $\mathbb{F}_p^d$. In this paper, we use the k-SUM conjecture to prove a more refined lower bound for k-SUM in $\mathbb{F}_p^d$:

▶ **Theorem 3.** *Under the k-SUM conjecture, for any $k \geq 2$, k-SUM in $\mathbb{F}_p^d$ requires time $\Omega(n^{k/2-o(1)})$ for even k, and time $\Omega(n^{\lceil k/2 \rceil - 2k\frac{\log k}{\log p} - o(1)})$ for odd k, when p is sufficiently large.*

Like the one-dimensional case, the fastest known algorithm for k-SUM in $\mathbb{F}_p^d$ is the meet-in-the-middle algorithm in time $\tilde{O}(n^{\lceil k/2 \rceil})$, which matches with the above conditional lower bound for even k.

Our conditional lower bound is meaningful for each $k \geq 2$, which is a stronger statement than the asymptotic result by Bhattacharyya et. al.

1.2 Monotone d-dimensional 3SUM

Chan and Lewenstein [7] first studied bounded monotone d-dimensional 3SUM, motivated by bounded monotone $(\min, +)$-convolution and histogram indexing. Chan and Lewenstein gave a remarkable $\tilde{O}(n^{2-\frac{2}{d+13}})$ algorithm with techniques from additive combinatorics. One of our main result is to show this algorithm is essentially optimal under the 3SUM conjecture.

[1] $\tilde{O}(f(n))$ is a notation for $O(f(n)\text{polylog}(n))$.

▶ **Definition 4** (Bounded Monotone d-dimensional 3SUM). A set $A \subset \mathbb{Z}^d$ is monotone increasing if it can be sorted as $A = \{a_1, \ldots, a_n\}$ such that the j-th coordinates of $a_1, \ldots, a_n$ form a monotone increasing sequence for each $j = 1, \ldots, d$. Given monotone sets $A, B, S \subset [n]^d$, bounded monotone d-dimensional 3SUM asks if there exist $a \in A, b \in B, s \in S$ such that $a + b = s$.[2]

Chan and Lewenstein's subquadratic $\tilde{O}(n^{2-\frac{2}{d+13}})$ algorithm shows that bounded monotone d-dimensional 3SUM is easier than integer 3SUM, but how much easier? Since monotonicity is a strong restriction on the set structure, one may wonder whether further improvements are possible. We show, under the 3SUM conjecture, the answer is no:

▶ **Theorem 5.** *Under the standard 3SUM conjecture, bounded d-dimensional monotone 3SUM requires time $\Omega(n^{2-\frac{4}{d}-o(1)})$.*

One can also define a strong version of the 3SUM conjecture (see Conjecture 8) and this yields a slightly stronger result:

▶ **Theorem 6.** *Under the strong 3SUM conjecture, bounded d-dimensional monotone 3SUM requires time $\Omega(n^{2-\frac{2}{d}-o(1)})$.*

In Chan and Lewenstein's $\tilde{O}(n^{2-\frac{2}{d+13}})$ upper bound, the exponent $2 - 2/(d + 13)$ comes from solving a quadratic equation capturing a fairly involved recurrence; it is surprising to see essentially the same exponent arise for completely different reasons in our lower bound.

1.3 Standard vs Strong 3SUM Conjecture

In this section we discuss the so-called "strong" 3SUM conjecture. For clarity, we refer to the well-known 3SUM conjecture (a special case of Conjecture 2) as the "standard" 3SUM conjecture:

▶ **Conjecture 7** (Standard 3SUM Conjecture). *Integer 3SUM requires time $\Omega(n^{2-o(1)})$.*

It is known that 3SUM on a set of n integers can be reduced to the bounded domain of $\{-n^3, \ldots, n^3\}$ via a randomized reduction [5, 17]. The strong 3SUM conjecture further restricts the domain to $\{-n^2, \ldots, n^2\}$. It was first proposed by Amir, Chan, Lewenstein, and Lewenstein to obtain better conditional lower bounds for jumbled indexing [4]. [3]

▶ **Conjecture 8** (Strong 3SUM Conjecture). *3SUM on a set of n integers in the domain of $\{-n^2, \ldots, n^2\}$ requires time $\Omega(n^{2-o(1)})$.*

As a context for the strong 3SUM conjecture, 3SUM in the domain of $\{-n^{2-\delta}, \ldots, n^{2-\delta}\}$ can be solved in time $\tilde{O}(n^{2-\delta})$ by Fast Fourier Transform (FFT). However, it is a long-standing open problem whether there is a truly subquadratic algorithm for 3SUM in the domain of $\{-n^2, \ldots, n^2\}$.

It is an open problem whether the strong 3SUM conjecture is equivalent to the standard 3SUM conjecture. In this paper, we prove a partial result along these lines in Theorem 9. This result may be a folklore in some communities, but it seems that it has not been written down, so we include a formal analysis for completeness.

[2] $[m]$ is a notation for $\{0, 1, \ldots, m-1\}$.

[3] Recently, Goldstein, Kopelowitz, Lewenstein, and Porat [13] showed that the strong 3SUM conjecture is not necessary for the hardness of jumbled indexing, and improved the result by basing on the standard 3SUM conjecture.

▶ **Theorem 9.** *Under the standard 3SUM conjecture, 3SUM$^+$ in the domain of $\{-n^{2+\delta}, ..., n^{2+\delta}\}$ requires time $\Omega(n^{2-o(1)})$ for any $\delta > 0$.*

Here, 3SUM$^+$ is the extension of 3SUM that reports all $a_3 \in A_3$ such that $a_1 + a_2 + a_3 = 0$ for some $a_1 \in A_1, a_2 \in A_2$, i.e. it outputs $A_3 \cap -(A_1 + A_2)$. As noted by Chan and Lewenstein [7], all the known 3SUM algorithms actually solve 3SUM$^+$, including Fast Fourier Transfrom and Baren et al.'s slightly subquadratic $O((n^2/\log^2 n)(\log\log n)^2)$ algorithm [5].

If Theorem 9 still holds with $\delta = 0$ and 3SUM in place of 3SUM$^+$, then the strong 3SUM conjecture would be equivalent to the standard 3SUM conjecture.

1.4 Organization

The next three sections contain the technical proofs of the main theorems. In Section 2, we prove Theorem 3, the lower bound for k-SUM in $\mathbb{F}_p^d$ under the k-SUM conjecture. Section 3 contains the proofs for Theorem 5 and Theorem 6, which are lower bounds for bounded monotone 3SUM under the standard and strong 3SUM conjectures. In Section 4, we prove Theorem 9.

2 Reductions Used for the Lower Bound for k-SUM in $\mathbb{F}_p^d$

Underlying the lower bound for k-SUM in $\mathbb{F}_p^d$ is a pair of reductions: a reduction from integer k-SUM in $\mathbb{F}_p^d$ to integer $(k+1)$-SUM, and a reduction from integer k-SUM to $(k+1)$-SUM in $\mathbb{F}_p^d$. From the reductions, we deduce conditional lower bounds for k-SUM in $\mathbb{F}_p^d$, for bounded monotone d-dimensional 3SUM, and for 3SUM$^+$ in bounded domain $\{-n^{2+\delta}, ..., n^{2+\delta}\}$.

A natural idea for reduction is to use the bijection between $\mathbb{F}_p^d$ and $[p^d] \subset \mathbb{Z}$, seeing $\mathbb{F}_p^d$ as a base-p representation of integers. However, the bijection is not an abelian group homomorphism, due to the "carries" in integer addition and the mod p effect in $\mathbb{F}_p^d$-addition. The main challenge is to simulate the carries while preserving the k-SUM structure.

Lemma 10 is the reduction from k-SUM in $\mathbb{F}_p^d$ to integer $(k+1)$-SUM. This is the easier direction among the two reductions. We map $a = (a_0, ..., a_{d-1}) \in \mathbb{F}_p^d \mapsto \sum_{i=0}^{d-1} a_i (kp)^i \in \mathbb{Z}$, treating $\mathbb{F}_p^d$ coordinates as digits in a base-kp number. Since the digits are blown up by powers of kp, there are no "carries" in integer addition.

▶ **Lemma 10.** *Given a k-SUM instance in $\mathbb{F}_p^d$ on k sets $A^{(1)}, \cdots, A^{(k)}$ of size n, it can be reduced to an integer $(k+1)$-SUM instance on k sets of size n and an additional set of size k^d.*

Proof. Let $g : \mathbb{F}_p^d \to \mathbb{Z}$ be the injective map $a = (a_0, ..., a_{d-1}) \mapsto \sum_{i=0}^{d-1} a_i (kp)^i$.

For any $a^{(1)}, \ldots, a^{(k)} \in \mathbb{F}_p^d$, the sum $a^{(1)} + \cdots + a^{(k)}$ is zero in $\mathbb{F}_p^d$ if and only if the i-th coordinate sum $\sum_{j=1}^{k} a_i^{(j)}$ is a multiple of p for all $0 \le i < d$. Since $a_i^{(j)} < p$, we know $\sum_{j=1}^{k} a_i^{(j)} < kp$. The above condition can be rewritten as $\sum_{j=1}^{k} a_i^{(j)} = \lambda_i p$, where $\lambda_i \in \{0, ..., k-1\}$. This is further equivalent to

$$g(a^{(1)}) + \cdots + g(a^{(k)}) = \sum_{i=0}^{d-1} \lambda_i p (kp)^i \text{ for some } (\lambda_1, \ldots, \lambda_d) \in [k]^d.$$

Therefore, the original k-SUM instance in $\mathbb{F}_p^d$ can be reduced to the integer $(k+1)$-SUM instance on $g(A^{(1)}), \ldots, g(A^{(k)})$, and an additional set $\{-\sum_{i=0}^{d-1} \lambda_i p (kp)^i : (\lambda_i) \in [k]^d\}$. ◀

▶ **Corollary 11.** *Given a k-SUM instance in $\mathbb{F}_p^d$ on k sets of size n, it can be reduced to k^d instances of integer k-SUM.*

Proof. Using the reduction in Lemma 10, we can solve the $(k+1)$-SUM instance by enumerating the additional set of size k^d. For each element a in the additional set of size k^d, subtract a from the first set, and solve the k-SUM instance on the first k sets. ◀

Since the proof only uses the additive structure of $\mathbb{F}_p^d$, Lemma 10 also holds more generally for k-SUM instance in $\mathbb{Z}_q^d$, where q is not necessarily a prime number. In fact, all proofs in this section can be adapted to $\mathbb{Z}_q^d$ with some modification.

In the reverse direction, it is more challenging to design a reduction from integer to $\mathbb{F}_p^d$, since the reduction needs to simulate integer addition carries with $\mathbb{F}_p^d$.

In the proof of Lemma 12, we start with the bijection between $\mathbb{F}_p^d$ and $[p^d] \subset \mathbb{Z}$, viewing $\mathbb{F}_p^d$ as a base-p representation of integers. This provides a way to map integers to $\mathbb{F}_p^d$, but unfortunately the map does not preserve the additive structure in k-SUM. To fix this problem, the key observation is that the map does preserve the additive structure with respect to k-SUM when all coordinates are between 0 and $\lfloor \frac{p}{k} \rfloor$. Points in $\{0, \ldots, \lfloor \frac{p}{k} \rfloor\}^d \subset \mathbb{F}_p^d$ behaves nicely for our purpose. Therefore, we divide $\mathbb{F}_p$ into k chunks: $\{0, \ldots, \lfloor \frac{p}{k} \rfloor\}, \{\lceil \frac{p}{k} \rceil, \ldots, 2\lfloor \frac{p}{k} \rfloor\}$, $\ldots, \{\lambda \lceil \frac{p}{k} \rceil, \ldots, (\lambda+1)\lfloor \frac{p}{k} \rfloor\}$, and so on. Accordingly, $\mathbb{F}_p^d$ is divided into k^d cubes $\{S_\lambda : \lambda = (\lambda_1, \ldots, \lambda_d) \in [k]^d\}$. For each cube, we shift it to align with the nice cube $S_0 = \{0, \ldots, \lfloor \frac{p}{k} \rfloor\}^d$ where the reduction preserves the additive structure, perform the reduction, and then shift it back.

A similar technique was first used by Abboud, Lewi, and Williams [2] to reduce integer k-SUM to k-VECTOR-SUM.

▶ **Lemma 12.** *Assume $p > k^2$. Given an integer k-SUM instance on k sets $X^{(1)}, \ldots, X^{(k)}$ of size n in the bounded universe $[m]$, it can be reduced to a $(k+1)$-SUM instance in $\mathbb{F}_p^{2d}$ on k sets of size n and a set of size k^{2d}, where $d = \log_p m$.*

Proof. Let $f : \mathbb{F}_p^d \to \{0, \ldots, m\}$ be the bijection $a = (a_0, \ldots, a_{d-1}) \mapsto \sum_{i=1}^{d-1} a_i p^i$. Note this is a bijection but does not preserve the additive structure.

Define $\mu : \mathbb{F}_p^d \to [k]^d$ as the following index function. For any $a = (a_0, \ldots, a_{d-1}) \in \mathbb{F}_p^d$, $\mu(a) = (\mu_0, \ldots, \mu_{d-1})$, where μ_i is defined to be the integer such that $\mu_i \lceil \frac{p}{k} \rceil \le a_i < (\mu_i+1)\lfloor \frac{p}{k} \rfloor$.

For any $\lambda \in [k]^d$, define $S_\lambda := \{a \in \mathbb{F}_p^d : \mu(a) = \lambda\}$. Define $\overline{a} = a - \lfloor \frac{p}{k} \rfloor \cdot \mu(a)$, then we can write

$$a = \overline{a} + \left\lfloor \frac{p}{k} \right\rfloor \cdot \mu(a),$$

where $\overline{a} \in S_0$.

$$f(a) = f(\overline{a}) + \sum_{i=1}^{d-1} \mu_i(a) \left\lfloor \frac{p}{k} \right\rfloor p^i.$$

Observe that for any $\overline{a^{(1)}}, \ldots, \overline{a^{(k)}} \in S_0$, it is true that $\sum_{j=1}^{k} f(\overline{a^{(j)}}) = f(\sum_{j=1}^{k} \overline{a^{(j)}})$. This equality does not hold in general for elements outside S_0.

Thus, for any $a^{(1)}, \ldots, a^{(k)} \in \mathbb{F}_p^d$,

$$\sum_{j=1}^{k} f(a^{(j)}) = \sum_{j=1}^{k} f(\overline{a^{(j)}}) + \sum_{i=1}^{d} \left(\sum_{j=1}^{k} \mu_i(a^{(j)}) \right) \left\lfloor \frac{p}{k} \right\rfloor p^i.$$

Fix $\lambda^{(1)}, \ldots, \lambda^{(k)} \in [k]^d$. For any $(x^1, \ldots, x^{(k)})$ where $x^{(j)} \in X^{(j)} \cap S_{\lambda^{(j)}}$, if we denote $a^{(j)} = f^{-1}(x^{(j)})$, then

$$\sum_{j=1}^{k} x^{(j)} = \sum_{j=1}^{k} f(a^{(j)}) = \sum_{j=1}^{k} f(\overline{a^{(j)}}) + \sum_{i=1}^{d} \sum_{j=1}^{k} \lambda_i^{(j)} \left\lfloor \frac{p}{k} \right\rfloor p^i = f(\sum_{j=1}^{k} \overline{a^{(j)}}) + \sum_{i=1}^{d} \sum_{j=1}^{k} \lambda_i^{(j)} \left\lfloor \frac{p}{k} \right\rfloor p^i.$$

Thus,

$$\sum_{j=1}^{k} x^{(j)} = 0 \iff f(\sum_{j=1}^{k} \overline{a^{(j)}}) + \sum_{i=1}^{d}\sum_{j=1}^{k} \lambda_i^{(j)} \left\lfloor \frac{p}{k} \right\rfloor p^i = 0,$$

and

$$\sum_{j=1}^{k} x^{(j)} = 0 \iff \sum_{j=1}^{k} \overline{a^{(j)}} = f^{-1}\left(-\sum_{i=1}^{d}\sum_{j=1}^{k} \lambda_i^{(j)} \left\lfloor \frac{p}{k} \right\rfloor p^i \right). \tag{*}$$

Note that the right hand side only depends on $\lambda^{(1)}, ..., \lambda^{(k)}$, or more specifically $\sum_{j=1}^{k} \lambda_i^{(j)}$.

This gives a reduction from the integer k-SUM instance to a $(k+1)$-SUM instance on k sets of size n in $\mathbb{F}_p^d \times \mathbb{Z}^d$ of the form

$$A^{(j)} = \{(\overline{a}, \mu(a)) : f(a) \in X^{(j)}\},$$

and a set of size k^{2d} consisting of elements in the form of

$$-\left(f^{-1}(-\sum_{i=1}^{d} \sigma_i \left\lfloor \frac{p}{k} \right\rfloor p^i), \quad \sigma \right),$$

for all $\sigma \in [k^2]^d \subset \mathbb{Z}^d$. The range $[k^2]^d$ is determined by the fact that since $\lambda^{(j)} \in [k]^d$ for each j, the sum σ is bounded above by k^2 in each coordinate. The $\mathbb{Z}^d$ component in $\mathbb{F}_p^d \times \mathbb{Z}^d$ is always bounded in $[k^2]^d$, so $\mathbb{F}_p^d \times \mathbb{Z}^d$ can be viewed as $\mathbb{F}_p^{2d}$ when $p > k^2$. ◀

The assumption $p > k^2$ in Lemma 12 is adopted only to simplify some details at the end when turning $\mathbb{F}_p^d \times [k^2]^d$ into $\mathbb{F}_p^{2d}$. When $p \le k^2$, the same techniques apply with more care in dealing with the $[k^2]^d$ component. All previous parts before (*) in the proof hold for $p > k$, so the following corollary only requires $p > k$.

▶ **Corollary 13.** *Assume $p > k$. Given an integer k-SUM instance on k sets of size n in the bounded universe $[m]$, it can be reduced to k^{2d} instances of k-SUM in $\mathbb{F}_p^d$ on k sets of size n, where $d = \log_p m$.*

Proof. This follows from the reduction up to (*) in the proof of the previous theorem. ◀

2.1 Lower Bound for k-SUM in $\mathbb{F}_p^d$

The following lower bound under the k-SUM conjecture follows from Lemma 12. For even k, our result matches with the $\tilde{O}(n^{k/2})$ upper bound. For odd k, our lower bound is $\Omega(n^{\lceil k/2 \rceil - 2k\frac{\log k}{\log p} - o(1)})$, while the best known upper bound is $\tilde{O}(n^{\lceil k/2 \rceil})$. The conditional lower bound converges to the upper bound as $p \to \infty$, which is expected since the group $\mathbb{F}_p$ behaves "more and more like $\mathbb{Z}$" as p increases.

The assumption that p is large enough such that $p \ge k^{2k}$ is still a meaningful assumption, because k^{2k} is a constant for fixed k.

▶ **Theorem 3.** (Restated) *For any $k \ge 2$, assume p is sufficiently large such that $p \ge k^{2k}$. Under the k-SUM conjecture, k-SUM in $\mathbb{F}_p^d$ requires time $\Omega(n^{k/2-o(1)})$ for even k, and $\Omega(n^{\lceil k/2 \rceil - 2k\frac{\log k}{\log p} - o(1)})$ for odd k, for $d \ge 2k \log_p n$.*

Proof. Using a randomized reduction [5, 17], we can assume without loss of generality that a given integer k-SUM instance is in the bounded domain $[n^k] = \{0, \ldots, n^k - 1\}$.

Suppose k is even. By Lemma 12, any integer $(k-1)$-SUM instance can be reduced to a k-SUM instance in $\mathbb{F}_p^d$ on $k-1$ sets of size n and a set of size $(k-1)^d$, where $d = 2\log_p n^{(k-1)}$. (The d used here is $2d$ in Lemma 12.) Assuming $p > k^{2k}$, then the size of the last set can be bounded above by $(k-1)^d \leq k^d = n^{2k\frac{\log k}{\log p}} \leq n$. Hence, integer $(k-1)$-SUM can be reduced to k-SUM in $\mathbb{F}_p^d$ on k sets of size n. If k-SUM in $\mathbb{F}_p^d$ can be solved in time $O(n^{k/2-\epsilon})$, then integer $(k-1)$-SUM could be solved in time $O(n^{k/2-\epsilon}) = O(n^{\lceil \frac{k-1}{2} \rceil - \epsilon})$, violating the k-SUM conjecture.

Suppose k is odd. By Corollary 13, integer k-SUM can be reduced to k^d instances of k-SUM in $\mathbb{F}_p^d$. If k-SUM in $\mathbb{F}_p^d$ can be solved in time $O(n^{\lceil k/2 \rceil - 2k\frac{\log k}{\log p} - \epsilon})$, then integer $(k-1)$-SUM could be solved in time $k^d \times O(n^{\lceil k/2 \rceil - 2k\frac{\log k}{\log p} - \epsilon})$. Since $d = 2\log_p n^{(k-1)} \leq 2k\frac{\log n}{\log p}$, we can bound k^d by $k^d = n^{d\frac{\log k}{\log n}} \leq n^{2k\frac{\log k}{\log p}}$. Therefore, integer $(k-1)$-SUM could be solved in time $k^d \times O(n^{\lceil k/2 \rceil - 2k\frac{\log k}{\log p} - \epsilon}) \leq O(n^{\lceil k/2 \rceil - \epsilon})$, violating the k-SUM conjecture. ◀

In particular, under the 3SUM conjecture, 3SUM in $\mathbb{F}_p^d$ requires time $\Omega(n^{2-\frac{6}{\log_3 p} - o(1)})$. This implies that there does not exist an $\epsilon > 0$ such that 3SUM in $\mathbb{F}_p^d$ can be solved in $O(n^{2-\epsilon})$ for all prime p, under the 3SUM conjecture. Combined with the reduction in Lemma 10, we obtain the following weak equivalence between integer k-SUM and k-SUM in $\mathbb{F}_p^d$.

▶ **Theorem 14.** *For any $k \geq 2$, integer k-SUM can be solved in $O(n^{\lceil k/2 \rceil - \delta})$ time for some $\delta > 0$ if and only if there exists an $\epsilon > 0$ such that k-SUM in $\mathbb{F}_p^d$ can be solved in $O(n^{\lceil k/2 \rceil - \epsilon})$ for all sufficiently large prime p.*

Proof. Suppose integer k-SUM can be solved in $O(n^{\lceil k/2 \rceil - \delta})$ time for some $0 < \delta < 1$. Take $\epsilon = \delta/2$, and take p large enough such that $k\frac{\log k}{\log p} < \delta/2$. Using a randomized reduction, we may assume without loss of generality that $d \leq k\log_p n$. By Corollary 11, k-SUM in $\mathbb{F}_p^d$ can be reduced to k^d instances of integer k-SUM, so k-SUM in $\mathbb{F}_p^d$ can be solved in time $k^d \cdot O(n^{\lceil k/2 \rceil - \delta}) \leq n^{k\frac{\log k}{\log p}} \cdot O(n^{\lceil k/2 \rceil - \delta}) = O(n^{\lceil k/2 \rceil - \delta/2})$.

Conversely, suppose there exists an $\epsilon > 0$ such that k-SUM in $\mathbb{F}_p^d$ can be solved in $O(n^{\lceil k/2 \rceil - \epsilon})$ for all sufficiently large prime p. Then, there exists some p large enough such that $\epsilon > 2k\frac{\log k}{\log p}$. By the proof of Theorem 3, integer k-SUM can be solved in time $O(n^{\lceil k/2 \rceil - \epsilon + 2k\frac{\log k}{\log p}})$. ◀

3 Proof of Lower Bound for Monotone 3SUM in $[n]^d$

In Definition 4, we defined bounded monotone d-dimensional 3SUM to be 3SUM on a monotone set in $[n]^d$, where n is the range of the coordinates. Alternatively, one could also define bounded monotone 3SUM to be on a monotone set of size n. Note that a monotone set in $[n]^d$ can have size at most dn, so the two definitions are equivalent up to a factor of d.

To prove the lower bound for bounded monotone 3SUM, we reduce from Convolution-3SUM to bounded monotone 3SUM. The Convolution-3SUM problem is a more restricted version of 3SUM:

▶ **Definition 15** (Convolution-3SUM)**.** Given an array $A[1 \ldots n]$, determine whether there exist $i \neq j$ with $A[i] + A[j] = A[i+j]$.

Pătraşcu [17] first proved that if 3SUM requires $\Omega(n^{2-o(1)})$ time, then so does Convolution-3SUM. Kopelowitz, Pettie, and Porat [16] showed that the randomized complexities of 3SUM and Convolution-3SUM differ by at most a logarithmic factor.

Note that the brute force algorithm for Convolution-3SUM runs in $O(n^2)$ time, as there are only $O(n^2)$ possible pairs to try. Amir, Chan, Lewenstein, and Lewenstein [4] pointed out a (randomized) reduction from Convolution-3SUM in any large domain to Convolution-3SUM in $\{-n^2, \ldots, n^2\}$.

The above results imply that the 3SUM conjecture is equivalent to the hardness of Convolution-3SUM in $\{-n^2, \ldots, n^2\}$, making Convolution-3SUM a useful tool for our purpose. Amir, Chan, Lewenstein, and Lewenstein [4] also restated the strong 3SUM conjecture as: Any algorithm for Convolution-3SUM in the domain of $\{-n, \ldots, n\}$ requires $\Omega(n^{2-o(1)})$ time.

Previously we defined k-SUM as determining whether there exists $a_1 + \cdots + a_k = 0$ for $a_i \in A_i$. In this section, for convenience we use an equivalent formulation of 3SUM: given sets A, B, S, determine whether there exist $a \in A, b \in B, s \in S$ such that $a + b = s$. The benefit is to avoid negative integer values. For 3SUM or Convolution-3SUM on A, B, S in $\{-u, \ldots, u\}$, add u to all values in A, B, and add $2u$ to all values in S. Therefore, it is sufficient to work with Convolution-3SUM in $[n^2]$ instead of $\{-n^2, \ldots, n^2\}$, and $[n]$ instead of $\{-n, \ldots, n\}$.

Convolution-3SUM in $[m]$ is a special case of 3SUM in $[n] \times [m]$, by mapping the array indices to the first coordinate, i.e. integer a_i in a Convolution-3SUM instance is mapped to the pair $(i, a_i) \in [n] \times [m]$. We will use this notation for simplicity. Since array indices are unique, no two values have the same first coordinate.

First, we reduce integer Convolution-3SUM to d-dimensional Convolution-3SUM, using the same techniques as in Lemma 12 (the reduction from integer 3SUM to d-dimensional 3SUM). Here we replace $\mathbb{F}_p^d$ with $[p]^d$, but the proof is nearly identical to Lemma 12, so we do not repeat it here.

▶ **Lemma 16.** *For any given dimension d, Convolution-3SUM in $[n^c]$ can be reduced to 4^d instances of Convolution-3SUM in $[n^{c/d}]^d$.*

Next we reduce from d-dimensional Convolution-3SUM to monotone d-dimensional Convolution-3SUM, by blowing up the bounded domain.

▶ **Lemma 17.** *Convolution-3SUM in $[m]^d$ can be reduced to Convolution-3SUM on monotone sets in $[nm]^d$.*

Proof. Let $f : [n] \times [m]^d \to [n] \times [nm]^d$ be the map $(a_0, a_1, ..., a_d) \mapsto (a_0, ma_0 + a_1, ..., ma_0 + a_d)$. Take $A' = f(A), B' = f(B), S' = f(S)$. Since f is linear in each coordinate, $a+b+s = 0$ implies $f(a) + f(b) + f(s) = 0$. Conversely, if $f(a) + f(b) + f(s) = 0$, the first coordinate guarantees $a_0 + b_0 + s_0 = 0$, and then it follows that $a_i + b_i + s_i = 0$ in each coordinate.

Since no two points in A (resp. B, S) share the same first coordinate, the new sets A' (resp. B', S') are monotone if we order them in increasing a_0 (resp. b_0, s_0), because the a_0 dominates in $ma_0 + a_i$. ◀

The following lemma is the main technical lemma from which Theorem 5 and Theorem 6 easily follow.

▶ **Lemma 18.** *Let c be a real constant between $1 \le c \le 2$, let $d > c$ be an integer, and let $\delta > 0$ be an arbitrarily small constant. Assume n is large enough such that $n > 2^{4d/\delta}$. If 3SUM for monotone sets in $[n]^d$ can be solved in time $O(n^{2-2c/d-\delta})$, then Convolution-3SUM in $[n^c]$ can be solved in time $O(n^{2-\delta/2})$.*

Proof. Let $\gamma = \frac{c}{d} < 1$. By Lemma 16 and Lemma 17, Convolution-3SUM in $[n^c]$ can be reduced to 4^d instances of Convolution-3SUM on monotone sets in $[n^{(1+\gamma)}]^d$. By writing the

array index as an additional dimension, i.e. mapping a_i in a Convolution-3SUM instance to (i, a_i), each of the 4^d Convolution-3SUM instances can be seen as $(d+1)$-dimensional monotone 3SUM in $[n^{(1+\gamma)}]^{(d+1)}$.

By assumption, 4^d instances of $(d+1)$-dimensional monotone 3SUM in $[n^{(1+\gamma)}]^{(d+1)}$ can be solved in time

$$4^d \cdot O(n^{(1+\gamma)(2-2c/d-\delta)}) = O(n^{\frac{d}{\log_4 n}+(1+\gamma)(2(1-c/d)-\delta)}).$$

The exponent can be simplified as follows. When $n > 2^{4d/\delta}$,

$$\frac{d}{\log_4 n} + (1+\gamma)(2(1-c/d)-\delta) = 2 - \delta - \gamma\delta - 2\gamma^2 + \frac{d}{\log_4 n} < 2 - \delta/2.$$

Thus, when n is sufficiently large, the 4^d instances of $(d+1)$-dimensional monotone 3SUM can be solved in time $O(n^{2-\delta/2})$. ◀

▶ **Theorem 5.** (Restated) *Under the standard 3SUM conjecture, bounded d-dimensional monotone 3SUM requires time $\Omega(n^{2-\frac{4}{d}-o(1)})$.*

Proof. This is a immediate corollary to Lemma 18 with $c = 2$. ◀

The lower bound can be improved to $\Omega(n^{2-\frac{4}{d+1}-o(1)})$ specifically for $c = 2$ with a little extra computation.

One can also define a strong version of the 3SUM conjecture (see Definition 8) and this yields a slightly stronger result:

▶ **Theorem 6.** (Restated) *Under the strong 3SUM conjecture, bounded d-dimensional monotone 3SUM requires time $\Omega(n^{2-\frac{2}{d}-o(1)})$.*

Proof. This is a immediate corollary to Lemma 18 with $c = 1$. ◀

4 Strong 3SUM Conjecture vs 3SUM Conjecture

The 3SUM conjecture states that integer 3SUM requires time $\Omega(n^{2-o(1)})$, while the strong 3SUM conjecture states that integer 3SUM in the bounded domain of $\{-n^2, \ldots, n^2\}$ requires time $\Omega(n^{2-o(1)})$. We would like to understand whether the 3SUM conjecture is equivalent to the strong 3SUM conjecture. To add evidence to this, we prove a partial result that 3SUM$^+$ in the domain of $\{-n^{2+\delta}, ..., n^{2+\delta}\}$ is as hard as unbounded integer 3SUM, using ideas about multidimensional 3SUM from previous sections.

▶ **Lemma 19** (Lemma 1, [5]). *Let A be a sorted list of n integers. For any fixed $c \in A$, we can decide whether c is a hit (i.e., whether there exist $a, b \in A$ such that $a + b = c$) in $O(n)$ time.*

The proof idea for Lemma 20 is similar to our reduction from multidimensional 3SUM to integer 3SUM in Lemma 10. Instead of $\mathbb{F}_p^3$, we consider 3SUM in the group $\mathbb{F}_{p_1} \times \mathbb{F}_{p_2} \times \mathbb{F}_{p_3}$ for three different primes.

▶ **Lemma 20.** *Let $0 < \epsilon < 1$. If 3SUM$^+$ in $[M]$ can be solved in $O(n^{2-\epsilon})$ time, then for any primes p_1, p_2, p_3 such that $p_1 p_2 p_3 \leq \frac{M}{32}$, 3SUM$^+$ in $\mathbb{F}_{p_1} \times \mathbb{F}_{p_2} \times \mathbb{F}_{p_3}$ can be solved in $O(n^{2-\epsilon})$ time.*

Proof. Let $S \subset \mathbb{F}_{p_1} \times \mathbb{F}_{p_2} \times \mathbb{F}_{p_3}$ be a subset of size n. Let $\varphi : \mathbb{F}_{p_1} \times \mathbb{F}_{p_2} \times \mathbb{F}_{p_3} \times \{0,1\}^3 \to [32p_1p_2p_3] \subset \mathbb{Z}$ be the injective map

$$\varphi : (a_1, a_2, a_3, t_1, t_2, t_3) \mapsto_\varphi (t_1p_1 + a_1) \cdot (16p_2p_3) + (t_2p_2 + a_2) \cdot (4p_3) + (t_3p_3 + a_3).$$

Let A be the image $A := \varphi(S \times \{0,1\}^3)$. The size of A is $8n$.

It is easy to check $\varphi(a_1, a_2, a_3, t_1, t_2, t_3) + \varphi(b_1, b_2, b_3, r_1, r_2, r_3) = \varphi(c_1, c_2, c_3, w_1, w_2, w_3)$ implies $(a_1, a_2, a_3) + (b_1, b_2, b_3) = (c_1, c_2, c_3)$ in $\mathbb{F}_{p_1} \times \mathbb{F}_{p_2} \times \mathbb{F}_{p_3}$.

Conversely, suppose $(a_1, a_2, a_3) + (b_1, b_2, b_3) = (c_1, c_2, c_3)$ in $\mathbb{F}_{p_1} \times \mathbb{F}_{p_2} \times \mathbb{F}_{p_3}$. For $i = 1, 2, 3$, since $a_i + b_i = c_i \pmod{p_i}$, either $a_i + b_i = c_i$ or $a_i + b_i = c_i + p_i$. Thus, there exists $w_1, w_2, w_3 \in \{0,1\}$ such that $\varphi(a_1, a_2, a_3, 0, 0, 0) + \varphi(b_1, b_2, b_3, 0, 0, 0) = \varphi(c_1, c_2, c_3, w_1, w_2, w_3)$.

To solve 3SUM$^+$ on S, first solve 3SUM$^+$ on A, which reports all $(A+A) \cap A$. The 3SUM$^+$ instance on S then outputs all $(c_1, c_2, c_3) \in S$ such that $\varphi(c_1, c_2, c_3, w_1, w_2, w_3) \in (A+A) \cap A$ for some $w_1, w_2, w_3 \in \{0,1\}$. The reduction takes $O(n)$ time. ◀

▶ **Lemma 21.** *Let $a_1, \ldots, a_k$ be k non-zero integers in the domain $\{-u, \ldots, u\}$. For any sufficiently large M,*

$$\Pr_{p_1,p_2,p_3 \in \mathcal{P}_M} \left[\#\{i : a_i \equiv 0 \pmod{p_1p_2p_3}\} < 24 \frac{\log^3(M) \log^3(u)}{M^3} k \right] \geq \frac{2}{3}.$$

Proof. Let $X_i = 1$ if $a_i \equiv 0 \pmod{p_1p_2p_3}$, and $X_i = 0$ otherwise. For each $i = 1, \ldots, k$, the number of distinct prime factors of a_i is at most $\log u$. By the Prime Number Theorem, $\pi(M) \sim \frac{M}{\log M}$, where $\pi(M)$ is the number of primes in $\{1, \ldots, M\}$. Hence, for M sufficiently large, $\pi(M) \geq \frac{1}{2} \frac{M}{\log M}$, and

$$\Pr_{p \in \mathcal{P}_M} [a_i \equiv 0 \pmod{p}] \leq \frac{\log u}{\frac{1}{2} \frac{M}{\log M}} = 2 \frac{\log M \log u}{M}.$$

Since p_1, p_2, p_3 are independently chosen primes,

$$E[X_i] = \Pr_{p_1,p_2,p_3 \in \mathcal{P}_M} [X_i = 1] = \prod_{j=1}^{3} \Pr_{p_j \in \mathcal{P}_M} [a_i \equiv 0 \pmod{p_j}] \leq \left(2 \frac{\log M \log u}{M} \right)^3.$$

Let $X = \sum_i X_i = \#\{i : a_i \equiv 0 \pmod{p_1p_2p_3}\}$. By linearity of expectation,

$$E[X] = \sum_i E[X_i] \leq k \cdot \left(2 \frac{\log M \log u}{M} \right)^3.$$

By Markov's Inequality, $\Pr[X \geq 3E[X]] \leq \frac{1}{3}$, as desired. ◀

▶ **Lemma 22.** *For any $0 < \delta < 1$, suppose 3SUM$^+$ in the bounded domain $[n^{2+\delta}]$ can be solved in $O(n^{2-\epsilon})$ time, then 3SUM can be solved in $O(n^{1+\delta} + n^{2-\epsilon} + n^{2-\delta} \log^6(n))$ time with probability $2/3$. (With probability $2/3$, the algorithm answers yes or no correctly, otherwise it outputs "Don't Know".)*

Proof. Using a randomized reduction [5, 17], assume without loss of generality the given 3SUM instance is in the domain of $[n^3]$. Also assume without loss of generality that the given 3SUM instance is on the same set A, asking whether there exist $a, b, c \in A$ such that $a + b = c$. Consider the following algorithm.

Step 1. Sort A. Choose $2n^\delta$ elements in A with uniform probability, and determine whether each of them is a hit. If a hit is found among the $2n^\delta$ elements, output yes and terminate. Otherwise, proceed to step 2. By Lemma 19, this step takes $O(n^{1+\delta})$ time.

Step 2. Choose three primes p_1, p_2, p_3 independently with uniform probability among all primes less than $n^{(2+\delta)/3}$. Map A to a set in A' in $\mathbb{F}_{p_1} \times \mathbb{F}_{p_2} \times \mathbb{F}_{p_3}$ by $a \mapsto (a \bmod p_1, a \bmod p_2, a \bmod p_3)$. By Lemma 20, under the given assumption, 3SUM$^+$ on A' can be solved in $O(n^{2-\epsilon})$ time.

Step 3. If the 3SUM$^+$ instance on A' reports more than $648\log^6(n)n^{1-\delta}$ hits, the algorithm fails and outputs "Don't Know".

Step 4. If the 3SUM$^+$ instance on A' reports no more than $648n^{1-\delta}\log^6(n)$ hits, for each pre-image of the hits, determine whether it is a hit in integer 3SUM. If a hit is found, output yes. Otherwise, output no. By Lemma 19, this step runs in time $O(n^{2-\delta}\log^6(n))$.

The total runtime of the algorithm is $O(n^{1+\delta} + n^{2-\epsilon} + n^{2-\delta}\log^6(n))$. To analyze the probability bound:

Suppose there are at least $n^{1-\delta}$ hits in A, then each randomly chosen element in Step 1 is a hit with probability at least $n^{-\delta}$. Let X be the total number of hits among the $2n^{\delta}$ randomly chosen elements in Step 1, and let $\mu = E[X]$. Since $\mu \geq 2$, by Multiplicative Chernoff Bound,

$$\Pr[X = 0] \leq \Pr[X < (1 - \frac{1}{2})\mu] < \left(\frac{e^{-\frac{1}{2}}}{(1-\frac{1}{2})^{1-\frac{1}{2}}}\right)^{\mu} = (2e)^{-\frac{\mu}{2}} \leq \frac{1}{2e}.$$

Thus, if the integer 3SUM instance on A has more than $n^{1-\delta}$ hits, the algorithm correctly outputs yes in Step 1 with probability at least $1 - \frac{1}{2e} \geq \frac{2}{3}$.

Suppose the 3SUM instance on A has fewer than $n^{1-\delta}$ hits. By applying Lemma 21 to all non-zero $a_i + b_j - c_k$ (at most n^3 triples), we know with probability at least $2/3$, the number of false positive triples is no more than

$$24n^3 \frac{\log^3(n^{(2+\delta)/3})\log^3(n^3)}{(n^{(2+\delta)/3})^3} \leq 24(2+\delta)^3\log^6(n)n^{1-\delta} \leq 648n^{1-\delta}\log^6(n).$$

Thus, the algorithm outputs yes/no correctly in Step 4 with probability at least $2/3$, and outputs "Don't Know" in Step 3 with probability at most $1/3$. ◀

▶ **Theorem 9.** (Restated) *Under the standard 3SUM conjecture, 3SUM$^+$ in the domain of $\{-n^{2+\delta}, ..., n^{2+\delta}\}$ requires time $\Omega(n^{2-o(1)})$ for any $\delta > 0$.*

Proof. This is an immediate corollary to Lemma 22. ◀

5 Conclusion and Open Problems

Under the standard k-SUM conjecture, we proved lower bounds for k-SUM in $\mathbb{F}_p^d$, for bounded monotone d-dimensional 3SUM, and for 3SUM$^+$ in the bounded domain of $\{-n^{2+\delta}, \ldots, n^{2+\delta}\}$ for arbitrarily small δ.

One open problem is whether there is a lower bound for 3XOR (3SUM in $\mathbb{F}_2^d$) under the 3SUM conjecture. Our result is only meaningful for p that are sufficiently large as a function of k. In particular, it does not assert anything for $p = 2$. 3XOR is related to other well known problems such as listing triangles [15]. Not much is known about the complexity of 3XOR. Even an $O(n^{1.99})$ algorithm or an $\Omega(n^{1.01})$ lower bound for 3XOR would be significant.

Another open problem is to obtain a faster k-SUM algorithm in $\mathbb{F}_p^d$ for odd k. Our conditional lower bound for k-SUM in $\mathbb{F}_p^d$ is tight for even k. However, for odd k, there is a gap between the best known $\tilde{O}(n^{\lceil k/2 \rceil})$ algorithm and our $\Omega(n^{\lceil k/2 \rceil - 2k\frac{\log k}{\log p} - o(1)})$ lower bound. From an optimistic perspective, for odd k, our results suggest that there may be

an $O(n^{\lceil k/2 \rceil - 2k\frac{\log k}{\log p}})$ algorithm for k-SUM in $\mathbb{F}_p^d$ without violating the k-SUM conjecture. In particular, there may be an $O(n^{2-\frac{c}{\log p}})$ algorithm for 3SUM in $\mathbb{F}_p^d$, even under the 3SUM conjecture.

References

1 Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-SUM conjecture. In *International Colloquium on Automata, Languages, and Programming*, pages 1–12. Springer, 2013.

2 Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In *European Symposium on Algorithms*, pages 1–12. Springer, 2014.

3 Nir Ailon and Bernard Chazelle. Lower bounds for linear degeneracy testing. *Journal of the ACM (JACM)*, 52(2):157–171, 2005.

4 Amihood Amir, Timothy M Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *Automata, Languages, and Programming*, pages 114–125. Springer, 2014.

5 Ilya Baran, Erik D Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.

6 Arnab Bhattacharyya, Piotr Indyk, David P Woodruff, and Ning Xie. The complexity of linear dependence problems in vector spaces. In *ICS*, pages 496–508, 2011.

7 Timothy M Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 31–40. ACM, 2015.

8 Rod G Downey, Michael R Fellows, Alexander Vardy, and Geoff Whittle. The parametrized complexity of some fundamental problems in coding theory. *SIAM Journal on Computing*, 29(2):545–570, 1999.

9 Jeff Erickson. Lower bounds for linear satisfiability problems. In *Chicago Journal of Theoretical Computer Science*, volume 8, 1999.

10 Ari Freund. Improved subquadratic 3sum. *Algorithmica*, pages 1–19, 2015.

11 Anka Gajentaan and Mark H Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry*, 5(3):165–185, 1995.

12 Omer Gold and Micha Sharir. Improved bounds for 3sum, k-sum, and linear degeneracy. *arXiv preprint arXiv:1512.05279*, 2015.

13 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. How hard is it to find (honest) witnesses? In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 57. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

14 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 621–630. IEEE, 2014.

15 Zahra Jafargholi and Emanuele Viola. 3SUM, 3XOR, triangles. *Algorithmica*, 74(1):326–343, 2016.

16 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1272–1287. Society for Industrial and Applied Mathematics, 2016.

17 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the Forty-Second ACM Symposium on Theory of computing*, pages 603–610. ACM, 2010.

18 Mihai Pătraşcu and Ryan Williams. On the possibility of faster SAT algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1065–1075. SIAM, 2010.

19 Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, pages 455–464. ACM, 2009.

20 Emanuele Viola. Reducing 3XOR to listing triangles, an exposition. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 18, page 113, 2011.

Parameterized Complexity of the List Coloring Reconfiguration Problem with Graph Parameters*†

Tatsuhiko Hatanaka[1], Takehiro Ito[2], and Xiao Zhou[3]

1 **Graduate School of Information Sciences, Tohoku University, Sendai, Japan**
hatanaka@ecei.tohoku.ac.jp
2 **Graduate School of Information Sciences, Tohoku University, Sendai, Japan**
takehiro@ecei.tohoku.ac.jp
3 **Graduate School of Information Sciences, Tohoku University, Sendai, Japan**
zhou@ecei.tohoku.ac.jp

Abstract

Let G be a graph such that each vertex has its list of available colors, and assume that each list is a subset of the common set consisting of k colors. For two given list colorings of G, we study the problem of transforming one into the other by changing only one vertex color assignment at a time, while at all times maintaining a list coloring. This problem is known to be PSPACE-complete even for bounded bandwidth graphs and a fixed constant k. In this paper, we study the fixed-parameter tractability of the problem when parameterized by several graph parameters. We first give a fixed-parameter algorithm for the problem when parameterized by k and the modular-width of an input graph. We next give a fixed-parameter algorithm for the shortest variant which computes the length of a shortest transformation when parameterized by k and the size of a minimum vertex cover of an input graph. As corollaries, we show that the problem for cographs and the shortest variant for split graphs are fixed-parameter tractable even when only k is taken as a parameter. On the other hand, we prove that the problem is W[1]-hard when parameterized only by the size of a minimum vertex cover of an input graph.

1998 ACM Subject Classification G.2.2 Graph Theory

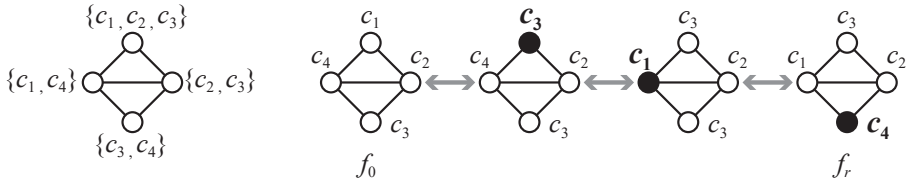
Keywords and phrases combinatorial reconfiguration, fixed-parameter tractability, graph algorithm, list coloring, W[1]-hardness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.51

1 Introduction

Recently, the framework of *reconfiguration* [14] has been extensively studied in the field of theoretical computer science. This framework models several situations where we wish to find a step-by-step transformation between two feasible solutions of a combinatorial (search) problem such that all intermediate solutions are also feasible and each step respects a fixed reconfiguration rule. This reconfiguration framework has been applied to several well-studied combinatorial problems. (See a survey [18].)

* This work is partially supported by JST CREST Grant Number JPMJCR1402, and by JSPS KAKENHI Grant Numbers JP16J02175, JP16K00003, and JP16K00004.
† A full version of the paper is available at https://arxiv.org/abs/1705.07551.



(a) G, L (b)

Figure 1 A reconfiguration sequence between two L-colorings f_0 and f_t of G.

1.1 Our problem

In this paper, we study a reconfiguration problem for list (vertex) colorings in a graph, which was introduced by Bonsma and Cereceda [3].

Let $C = \{c_1, c_2, \ldots, c_k\}$ be the set of k colors, called the *color set*. A (proper) *k-coloring* of a graph $G = (V, E)$ is a mapping $f\colon V \to C$ such that $f(v) \neq f(w)$ for every edge $vw \in E$. In *list coloring*, each vertex $v \in V$ has a set $L(v) \subseteq C$ of colors, called the *list of* v; sometimes, the list assignment $L\colon V \to 2^C$ itself is called a *list*. Then, a k-coloring f of G is called an *L-coloring* of G if $f(v) \in L(v)$ holds for every vertex $v \in V$. Therefore, a k-coloring of G is simply an L-coloring of G when $L(v) = C$ holds for every vertex v of G, and hence L-coloring is a generalization of k-coloring. Figure 1(b) illustrates four L-colorings of the same graph G in Figure 1(a); the color assigned to each vertex is attached to the vertex.

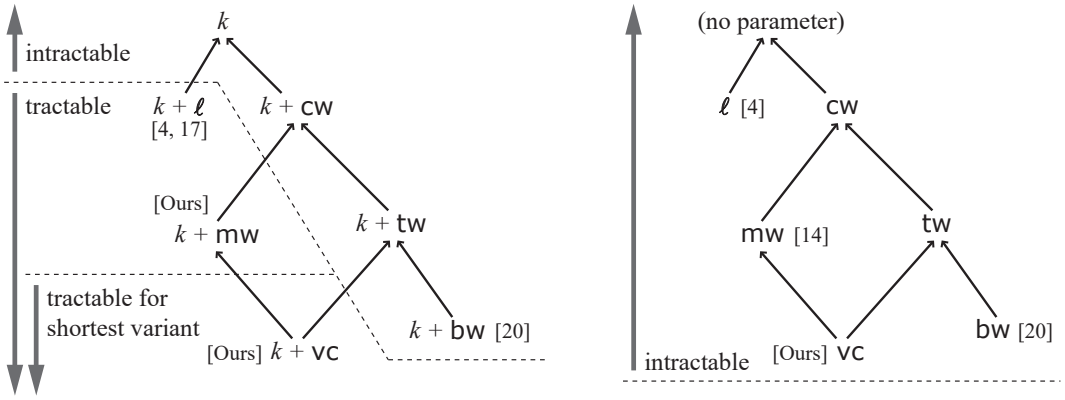
In the reconfiguration framework, two L-colorings f and f' of a graph $G = (V, E)$ are said to be *adjacent* if $|\{v \in V\colon f(v) \neq f'(v)\}| = 1$ holds, that is, f' can be obtained from f by recoloring exactly one vertex. A sequence $\langle f_0, f_1, \ldots, f_\ell \rangle$ of L-colorings of G is called a *reconfiguration sequence* between f_0 and f_ℓ (of *length* ℓ) if f_{i-1} and f_i are adjacent for each $i \in \{1, 2, \ldots, \ell\}$. Two L-colorings f and f' are *reconfigurable* if there exists a reconfiguration sequence between them. The LIST COLORING RECONFIGURATION problem is to determine whether two given L-colorings f_0 and f_t are reconfigurable, or not. Figure 1 shows an example of a yes-instance of LIST COLORING RECONFIGURATION, where the vertex whose color assignment was changed from the previous one is depicted by a black circle.

1.2 Known and related results

LIST COLORING RECONFIGURATION is one of the most well-studied reconfiguration problems, as well as COLORING RECONFIGURATION which is a special case of the problem such that $L(v) = \{c_1, c_2, \ldots, c_k\}$ holds for every vertex v. These problems have been studied intensively from various viewpoints [1, 2, 3, 4, 6, 7, 9, 13, 15, 19] including the generalizations [5, 20].

Bonsma and Cereceda [3] proved that COLORING RECONFIGURATION is PSPACE-complete even for bipartite graphs and any fixed constant $k \geq 4$. On the other hand, Cereceda et al. [7] gave a polynomial-time algorithm solving COLORING RECONFIGURATION for any graph and $k \leq 3$; the algorithm can be applied to LIST COLORING RECONFIGURATION, too. In particular, the former result implies that there is no fixed-parameter algorithm for COLORING RECONFIGURATION (and hence LIST COLORING RECONFIGURATION) when parameterized by only k under the assumption of P $\neq$ PSPACE.

Bonsma et al. [4] and Johnson et al. [15] independently developed a fixed-parameter algorithm to solve COLORING RECONFIGURATION when parameterized by $k + \ell$, where ℓ is the upper bound on the length of reconfiguration sequences, and again their algorithms can be applied to LIST COLORING RECONFIGURATION. In contrast, if COLORING RECONFIGURATION is parameterized only by ℓ, then it is W[1]-hard when k is an input [4] and does not admit a polynomial kernelization when k is fixed unless the polynomial hierarchy collapses [15].



(a) parameters with k (b) parameters without k

Figure 2 All results (including known ones) for LIST COLORING RECONFIGURATION from the viewpoint of parameterized complexity, where cw, tw, bw, mw, and vc are the upper bounds on the cliquewidth, treewidth, bandwidth, modular-width, and the size of a minimum vertex cover of an input graph, respectively.

Hatanaka et al. [13] proved that LIST COLORING RECONFIGURATION is PSPACE-complete even for complete split graphs, whose modular-width is zero. Wrochna [19] proved that LIST COLORING RECONFIGURATION is PSPACE-complete even when k and the bandwidth of an input graph are bounded by some constant; thus the treewidth and the cliquewidth of an input graph are also bounded.

1.3 Our contribution

To the best of our knowledge, known algorithmic results mostly employed the length ℓ of reconfiguration sequences as a parameter [4, 15], and no fixed-parameter algorithm is known when parameterized by graph parameters. Therefore, we study LIST COLORING RECONFIGURATION when parameterized by several graph parameters, and paint an interesting map of graph parameters which shows the boundary between fixed-parameter tractability and intractability. Our map is Figure 2 which shows both known and our results, where an arrow $\alpha \to \beta$ indicates that the parameter α is "stronger" than β, that is, β is bounded if α is bounded. (For relationships of parameters, see, e.g., [10, 16].)

More specifically, we first give a fixed-parameter algorithm solving LIST COLORING RECONFIGURATION when parameterized by k and the modular-width mw of an input graph. (The definition of modular-width will be given in Section 2.1.) Note that, according to the known results [3, 13], we cannot construct a fixed-parameter algorithm for general graphs when only one of k and mw is taken as a parameter under the assumption of P $\neq$ PSPACE. However, as later shown in Corollary 4, our algorithm implies that the problem is fixed-parameter tractable for cographs even when only k is taken as a parameter.

We then consider the shortest variant which computes the length of a shortest reconfiguration sequence (i.e., the minimum number of recoloring steps) for a yes-instance of LIST COLORING RECONFIGURATION, and show that it admits a fixed-parameter algorithm when parameterized by k and the size of a minimum vertex cover of an input graph. Moreover, as a corollary, we show that the shortest variant is fixed-parameter tractable for split graphs even when only k is taken as a parameter.

Finally, we prove that LIST COLORING RECONFIGURATION is $W[1]$-hard when parameterized only by the size of a minimum vertex cover of an input graph.

Due to the page limitation, several proofs are omitted from this extended abstract.

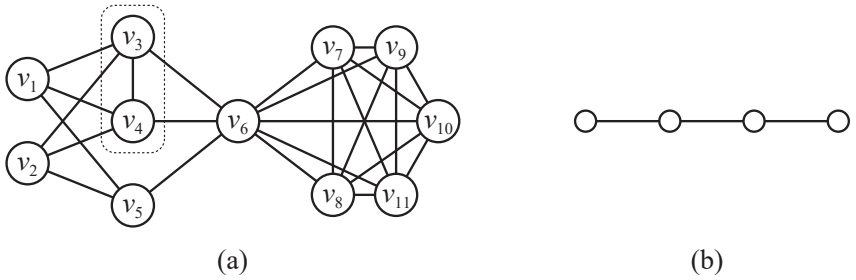


Figure 3 (a) An example of module and (b) a prime.

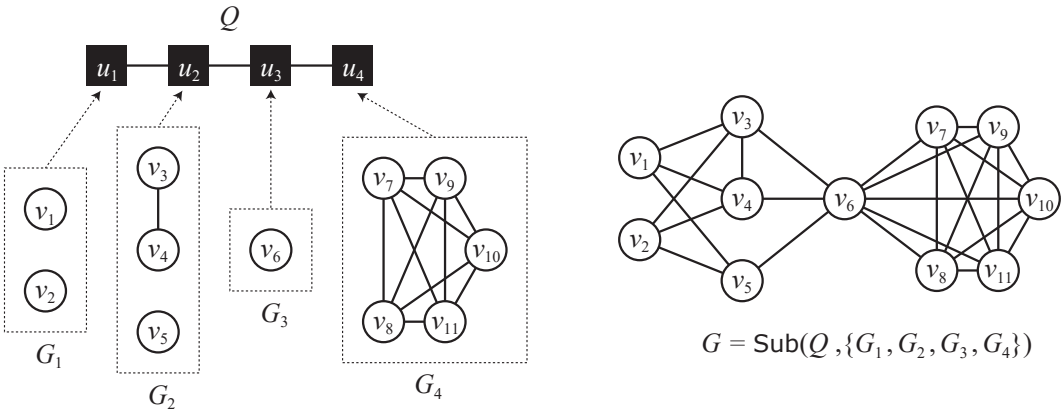


Figure 4 An example of substitution operation.

2 Preliminaries

We assume without loss of generality that graphs are simple and connected. Let $G = (V, E)$ be a graph with vertex set V and edge set E; we sometimes denote by $V(G)$ and $E(G)$ the vertex set and the edge set of G, respectively. For a vertex v in G, we denote by $N(G, v)$ the neighborhood $\{w \in V : vw \in E\}$ of v in G. For a vertex subset $V' \subseteq V$, we denote by $G[V']$ the subgraph of G induced by V', and denote $G \setminus V' = G[V(G) \setminus V']$. For a subgraph H of G, we denote $G \setminus H = G \setminus V(H)$. Let $\omega(G)$ be the size of a maximum clique of G. We have the following simple observation.

▶ **Observation 1.** *Let G be a graph with a list $L : V(G) \to 2^C$. If G has an L-coloring, then $\omega(G) \leq |C|$.*

A graph is *split* if its vertex set can be partitioned into a clique and an independent set. A graph is a *cograph* (or a P_4*-free* graph) if it contains no induced path with four vertices.

2.1 Modules and modular decomposition

A *module* of a graph $G = (V, E)$ is a vertex subset $M \subseteq V$ such that $N(G, v) \setminus M = N(G, w) \setminus M$ for every two vertices v and w in M. In other words, the module M is a set of vertices whose neighborhoods in $G \setminus M$ are the same. For example, the graph in Figure 3(a) has a module $M = \{v_3, v_4\}$ for which $N(G, v_3) \setminus M = N(G, v_4) \setminus M = \{v_1, v_2, v_6\}$ holds. Note that the vertex set V of G, the set consisting of only a single vertex, and the empty set $\emptyset$ are all modules of G; they are called *trivial*. A graph G is a *prime* if all of its modules are trivial; for an example, see Figure 3(b).

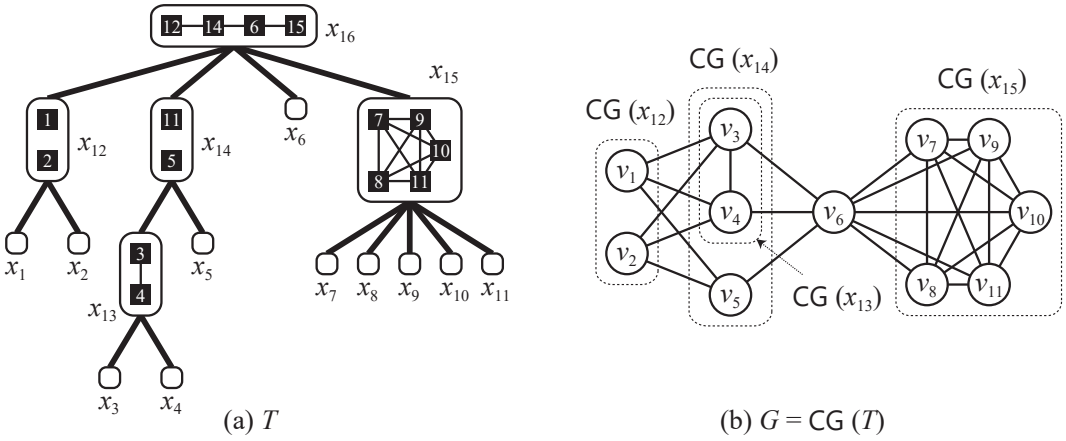


Figure 5 (a) A substitution tree T for (b) a graph G.

We now introduce the notion of modular decomposition, which was first presented by Gallai in 1967 as a graph decomposition technique [11]. For a survey, see, e.g., [12].

We first define the *substitution* operation, which constructs one graph from more than one graphs. Let Q be a graph, called a *quotient graph*, consisting of p (≥ 2) nodes $u_1, u_2, \ldots, u_p$, and let $\mathcal{F} = \{G_1, G_2, \ldots, G_p\}$ be a family of vertex-disjoint graphs such that G_i corresponds to u_i for every $i \in \{1, 2, \ldots, p\}$. The *$Q$-substitution* of $\mathcal{F}$, denoted by $\mathsf{Sub}(Q, \mathcal{F})$, is the graph which is obtained by taking a union of all graphs in $\mathcal{F}$ and then connecting every pair of vertices $v \in V(G_i)$ and $w \in V(G_j)$ by an edge if and only if u_i and u_j are adjacent in Q. That is, the vertex set of $\mathsf{Sub}(Q, \mathcal{F})$ is $\bigcup\{V(G_i) \colon G_i \in \mathcal{F}\}$, and the edge set of $\mathsf{Sub}(Q, \mathcal{F})$ is the union of $\bigcup\{E(G_i) \colon G_i \in \mathcal{F}\}$ and $\{vw \colon v \in V(G_i), w \in V(G_j), u_iu_j \in E(Q)\}$. (See Figure 4 as an example.)

A *substitution tree* is a rooted tree T such that each non-leaf node $x \in V(T)$ is associated with a quotient graph $\mathsf{Q}(x)$ and has $|V(\mathsf{Q}(x))|$ child nodes. For each node $x \in V(T)$, we can recursively define the *corresponding graph* $\mathsf{CG}(x)$ as follows: If x is a leaf, $\mathsf{CG}(x)$ consists of a single vertex. Otherwise, let $y_1, y_2, \ldots, y_p$ be $p = |V(\mathsf{Q}(x))|$ children of x, then $\mathsf{CG}(x) = \mathsf{Sub}(\mathsf{Q}(x), \{\mathsf{CG}(y_1), \mathsf{CG}(y_2), \ldots, \mathsf{CG}(y_p)\})$. For the root r of T, $\mathsf{CG}(r)$ is called the *corresponding graph of* T, and we denote $\mathsf{CG}(T) := \mathsf{CG}(r)$. We say that T is a substitution tree *for* a graph G if $\mathsf{CG}(T) = G$, and refer to a *node* in T in order to distinguish it from a vertex in G. Figure 5(a) illustrates a substitution tree for the graph G in Figure 5(b); each leaf x_i, $i \in \{1, 2, \ldots, 11\}$, corresponds to the subgraph of G consisting of a single vertex v_i. We note that the vertex set $V(\mathsf{CG}(x))$ of each corresponding graph $\mathsf{CG}(x)$, $x \in V(T)$, forms a module of $\mathsf{CG}(T)$.

A *modular decomposition tree* T (an *MD-tree* for short) for a graph G is a substitution tree for G which satisfies the following three conditions:

- Each node $x \in V(T)$ applies to one of the following three types:
  - a *series* node, whose quotient graph $\mathsf{Q}(x)$ is a complete graph;
  - a *parallel* node, whose quotient graph $\mathsf{Q}(x)$ is an edge-less graph; and
  - a *prime* node, whose quotient graph $\mathsf{Q}(x)$ is a prime with at least four vertices.
- No edge connects two series nodes.
- No edge connects two parallel nodes.

It is known that any graph G has a unique MD-tree with $O(|V(G)|)$ nodes, and it can be computed in time $O(|V(G)| + |E(G)|)$ [17]. We denote by $\mathsf{MD}(G)$ the unique MD-tree for a graph G. The *modular-width* $\mathsf{mw}(G)$ of a graph G is the maximum number of children of

a prime node in its MD-tree $\mathsf{MD}(G)$. The substitution tree T in Figure 5(a) is indeed the MD-tree for the graph G in Figure 5(b), and hence $\mathsf{mw}(G) = 4$; note that only x_{16} is a prime node in T.

We now define a variant of MD-trees, which will make our proofs and analyses simpler. A *pseudo modular decomposition tree* T (a *PMD-tree* for short) for a graph G is a substitution tree for G which satisfies the following two conditions:

- Each node $x \in V(T)$ applies to one of the following three types:
  - a 2-*join* node, whose quotient graph $\mathsf{Q}(x)$ is a complete graph with exactly two vertices;
  - a *parallel* node, whose quotient graph $\mathsf{Q}(x)$ is an edge-less graph; and
  - a *prime* node, whose quotient graph $\mathsf{Q}(x)$ is a prime with at least four vertices.
- No edge connects two parallel nodes.

▶ **Proposition 2.** *For any graph G, there exists a PMD-tree T with $O(|V(G)|)$ nodes such that each prime node $x \in V(T)$ has at most $\mathsf{mw}(G)$ children, and it can be constructed in polynomial time.*

We denote by $\mathsf{PMD}(G)$ a PMD-tree for G such that each prime node $x \in V(T)$ has at most $\mathsf{mw}(G)$ children. The *pseudo modular-width* $\mathsf{pmw}(G)$ of a graph G is the maximum number of children of a non-parallel node in its PMD-tree. Notice that $\mathsf{pmw}(G) = \max\{2, \mathsf{mw}(G)\}$ holds.

2.2 Other notation

Let $G = (V, E)$ be a graph, and let $L\colon V \to 2^C$ be a list. For two L-colorings f and f' of G, we define the *difference* $\mathsf{dif}(f, f')$ *between* f *and* f' as the set $\{v \in V\colon f(v) \neq f'(v)\}$. Notice that f and f' are adjacent if and only if $|\mathsf{dif}(f, f')| = 1$.

We express an instance $\mathcal{I}$ of LIST COLORING RECONFIGURATION by a 4-tuple (G, L, f_0, f_t) consisting of a graph G, a list L, and *initial* and *target* L-colorings f_0 and f_t of G.

Finally, we introduce a notion of "restriction" of mappings and instances. Consider an arbitrary mapping $\mu\colon V(G) \to S$, where G is a graph and S is any set. For a subgraph H of G, we denote by μ^H the *restriction* of μ on $V(H)$, that is, μ^H is a mapping from $V(H)$ to S such that $\mu^H(v) = \mu(v)$ for each vertex $v \in V(H)$. Let $\mathcal{I} = (G, L, f_0, f_t)$ be an instance of LIST COLORING RECONFIGURATION. For a subgraph H of G, we define the *restriction* $\mathcal{I}^H$ of $\mathcal{I}$ (on H) as the instance (H, L^H, f_0^H, f_t^H) of LIST COLORING RECONFIGURATION. Notice that f_0^H and f_t^H are proper L^H-colorings of H.

3 Fixed-Parameter Algorithm for Bounded Modular-Width Graphs

The following is our main theorem of this section.

▶ **Theorem 3.** LIST COLORING RECONFIGURATION *is fixed-parameter tractable when parameterized by $k + \mathsf{mw}$, where k and mw are the upper bounds on the size of the color set and the modular-width of an input graph, respectively.*

Because it is known that any cograph has modular-width zero, we have the following result as a corollary of Theorem 3.

▶ **Corollary 4.** LIST COLORING RECONFIGURATION *is fixed-parameter tractable for cographs when parameterized by the size k of the color set.*

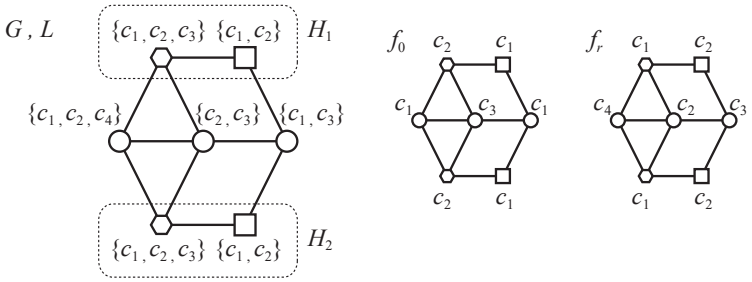


Figure 6 An instance $\mathcal{I} = (G, L, f_0, f_t)$ of LIST COLORING RECONFIGURATION, and two identical subgraphs H_1 and H_2.

Recall that $\mathsf{pmw}(G) = \max\{2, \mathsf{mw}(G)\}$, and hence $\mathsf{pmw}(G) \leq \mathsf{mw}(G) + 2$. Therefore, as a proof of Theorem 3, it suffices to give a fixed-parameter algorithm for LIST COLORING RECONFIGURATION with respect to $k + \mathsf{pmw}$, where pmw is an upper bound on $\mathsf{pmw}(G)$.

3.1 Reduction rule

In this subsection, we give a useful lemma, which compresses an input graph into a smaller graph with keeping the reconfigurability.

Let $\mathcal{I} = (G, L, f_0, f_t)$ be an instance of LIST COLORING RECONFIGURATION. For each vertex $v \in V(G)$, we define a *vertex assignment* $A(v)$ as a triple $(L(v), f_0(v), f_t(v))$ consisting of a list, and initial and target color assignments of v. Let H_1 and H_2 be two induced subgraphs of G such that $|V(H_1)| = |V(H_2)|$ and $V(H_1) \cap V(H_2) = \emptyset$. Then, H_1 and H_2 are *identical* (on $\mathcal{I}$) if there exists a bijective function $\phi\colon V(H_1) \to V(H_2)$ which satisfies the following two conditions:

1. H_1 and H_2 are isomorphic under ϕ, that is, $vw \in E(H_1)$ if and only if $\phi(v)\phi(w) \in E(H_2)$.
2. For all vertices $v \in V(H_1)$,
   a. $N(G, v) \setminus V(H_1) = N(G, \phi(v)) \setminus V(H_2)$; and
   b. $A(v) = A(\phi(v))$, that is, $L(v) = L(\phi(v))$, $f_0(v) = f_0(\phi(v))$ and $f_t(v) = f_t(\phi(v))$.

We note that the condition 2-a implies that there is no edge between H_1 and H_2. Figure 6 shows an example of identical subgraphs H_1 and H_2 on $\mathcal{I} = (G, L, f_0, f_t)$, where the bijective function maps each vertex in H_1 to a vertex in H_2 with the same shape.

We now prove the following key lemma, which holds for any graph.

▶ **Lemma 5.** (Reduction rule) *Let $\mathcal{I} = (G, L, f_0, f_t)$ be an instance of* LIST COLORING RECONFIGURATION*, and let H_1 and H_2 be two identical subgraphs of G. Then, $\mathcal{I}^{G \setminus H_2}$ is a yes-instance if and only if $\mathcal{I}$ is.*

3.2 Kernelization

Let $\mathcal{I} = (G, L, f_0, f_t)$ be an instance of LIST COLORING RECONFIGURATION. Suppose that the color set C has at most k colors, G is a connected graph with $\mathsf{pmw}(G) \leq \mathsf{pmw}$, and all vertices of G are totally ordered according to an arbitrary binary relation $\prec$.

3.2.1 Sufficient condition for identical subgraphs

We first give a sufficient condition for which two nodes in a PMD-tree $\mathsf{PMD}(G)$ for G correspond to identical subgraphs. Let $x \in V(\mathsf{PMD}(G))$ be a node, let $p := |V(\mathsf{CG}(x))|$, and assume that all vertices in $V(\mathsf{CG}(x))$ are labeled as $v_1, v_2, \ldots, v_p$ according to $\prec$; that is,

$v_i \prec v_j$ holds for each i, j with $1 \leq i < j \leq p$. Let $m \geq p$ be some integer which will be defined later. We now define an $(m+1) \times m$ matrix $\mathcal{M}_m(x)$ as follows:

$$(\mathcal{M}_m(x))_{i,j} = \begin{cases} 1 & \text{if } i,j \leq p \text{ and } v_iv_j \in E(\mathsf{CG}(x)); \\ 0 & \text{if } i,j \leq p \text{ and } v_iv_j \notin E(\mathsf{CG}(x)); \\ 0 & \text{if } p < i \leq m \text{ or } p < j \leq m; \\ A(v_j) & \text{if } i = m+1 \text{ and } j \leq p; \\ \emptyset & \text{otherwise,} \end{cases}$$

where $(\mathcal{M}_m(x))_{i,j}$ denotes an (i,j)-element of $\mathcal{M}_m(x)$. Notice that $\mathcal{M}_m(x)$ contains the adjacency matrix of $\mathsf{CG}(x)$ at its upper left $p \times p$ submatrix, and the bottommost row represents the vertex assignment of each vertex in $V(\mathsf{CG}(x))$. We call $\mathcal{M}_m(x)$ an *m-ID-matrix* of x. For example, consider the node x_{13} in Figure 5(a). Then, $p = 2$, and a 4-ID-matrix of x_{13} is as follows:

$$\mathcal{M}_4(x_{13}) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ A(x_3) & A(x_4) & \emptyset & \emptyset \end{bmatrix}$$

▶ **Lemma 6.** *Let y_1 and y_2 be two children of a parallel node x in $\mathsf{PMD}(G)$, and let m be an integer with $m \geq \max\{|V(\mathsf{CG}(y_1))|, |V(\mathsf{CG}(y_2))|\}$. If $\mathcal{M}_m(y_1) = \mathcal{M}_m(y_2)$ holds, then $\mathsf{CG}(y_1)$ and $\mathsf{CG}(y_2)$ are identical.*

3.2.2 Kernelization algorithm

We now describe how to kernelize an input instance. Our algorithm traverses a PMD-tree $\mathsf{PMD}(G)$ of G by a depth-first search in post-order starting from the root of $\mathsf{PMD}(G)$, that is, the algorithm processes a node of $\mathsf{PMD}(G)$ after its all children are processed.

Let $x \in V(\mathsf{PMD}(G))$ be a node which is currently visited. If x is a non-parallel node, we do nothing. Otherwise (i.e., if x is a parallel node,) let Y be the set of all children of x, and let $m := \max_{y \in Y} |V(\mathsf{CG}(y))|$. We first construct m-ID-matrices of all children of x. If there exist two nodes y_1 and y_2 such that $\mathcal{M}_m(y_1) = \mathcal{M}_m(y_2)$, then $\mathsf{CG}(y_1)$ and $\mathsf{CG}(y_2)$ are identical; and hence we remove $\mathsf{CG}(y_2)$ from G by Lemma 5. Then, we modify $\mathsf{PMD}(G)$ in order to keep it still being a PMD-tree for the resulting graph as follows. We remove a subtree rooted at y_2 from $\mathsf{PMD}(G)$, and delete a node corresponding to y_2 from a quotient graph $\mathsf{Q}(x)$ of x. If this removal makes x having only one child y in the PMD-tree, we contract the edge xy into a new node x' such that $\mathsf{Q}(x') = \mathsf{Q}(x)$.

The running time of this kernelization can be estimated as follows. For each node $x \in V(\mathsf{PMD}(G))$, the construction of m-ID-matrices can be done in time $O(|Y| \cdot m^2) = O(|V(G)|^3)$. We can check if $\mathcal{M}_m(y_1) = \mathcal{M}_m(y_2)$ for each pair of children y_1 and y_2 of x in time $O(m^2) = O(|V(G)|^2)$. Moreover, a modification of $\mathsf{PMD}(G)$, which follows an application of Lemma 5, can be done in polynomial time. Recall that the number of children of x and the size of a PMD-tree $\mathsf{PMD}(G)$ are both bounded linearly in $|V(G)|$, and hence our kernelization can be done in polynomial time.

3.2.3 Size of the kernelized instance

We finally prove that the size of the obtained instance $\mathcal{I}' = (G', L', f'_0, f'_t)$ depends only on $k + \mathsf{pmw}$; recall that pmw is the upper bound on $\mathsf{pmw}(G)$. By Observation 1, we can assume

that the maximum clique size $\omega(G')$ is at most k. In addition, G' is connected since G is connected and an application of Lemma 5 does not affect the connectivity of the graph. Therefore, it suffices to prove the following lemma.

▶ **Lemma 7.** *The graph G' has at most $h_{k,\mathsf{pmw}}(\omega(G'))$ vertices, where $h_{k,\mathsf{pmw}}(i)$ is recursively defined for an integer $i \geq 1$ as follows:*

$$h_{k,\mathsf{pmw}}(i) = \begin{cases} 1 & \text{if } i = 1; \\ \mathsf{pmw} \cdot h_{k,\mathsf{pmw}}(i-1) \cdot \sqrt{2}^{(h_{k,\mathsf{pmw}}(i-1))^2} \cdot (2^k \cdot k^2)^{h_{k,\mathsf{pmw}}(i-1)} & \text{otherwise.} \end{cases}$$

In particular, $h_{k,\mathsf{pmw}}(\omega(G'))$ depends only on $k + \mathsf{pmw}$.

Finally, we prove Theorem 3. By the above discussions, we can compute the kernelized instance $\mathcal{I}' = \mathcal{I}^{G'}$ of LIST COLORING RECONFIGURATION in polynomial time. Because the size of $\mathcal{I}'$ depends only on $k + \mathsf{pmw}$, we can solve $\mathcal{I}'$ by enumerating all $L^{G'}$-colorings. The running time for this enumeration depends only on $k + \mathsf{pmw}$, and hence we obtain a fixed-parameter algorithm for LIST COLORING RECONFIGURATION.

This completes the proof of Theorem 3.

4 Shortest Variant

In this section, we study the shortest variant, LIST COLORING SHORTEST RECONFIGURATION. We note that the shortest length can be expressed by a polynomial number of bits, because there are at most k^n colorings for a graph with n vertices and k colors. Therefore, the answer can be output in polynomial time. The following is our result.

▶ **Theorem 8.** LIST COLORING SHORTEST RECONFIGURATION *is fixed-parameter tractable when parameterized by $k + \mathsf{vc}$, where k and vc are the upper bounds on the sizes of the color set and a minimum vertex cover of an input graph, respectively.*

As a corollary, we have the following result.

▶ **Corollary 9.** LIST COLORING SHORTEST RECONFIGURATION *is fixed-parameter tractable for split graphs when parameterized by the size k of the color set.*

As a proof of Theorem 8, we give such a fixed-parameter algorithm. Our basic idea is the same as the fixed-parameter algorithm in Section 3. However, in order to compute the shortest length, we consider a more general "weighted" version of LIST COLORING SHORTEST RECONFIGURATION, which is defined as follows. Let $\mathcal{I} = (G, L, f_0, f_t)$ be an instance of LIST COLORING RECONFIGURATION, and assume that each vertex $v \in V(G)$ has a *weight* $w(v) \in \mathbb{N}$, where $\mathbb{N}$ is the set of all positive integers. For two adjacent L-colorings f and f' of a graph G, we define the *gap* $\mathsf{gap}_w(f, f')$ *between* f *and* f' as the weight $w(v)$ of v, where v is a unique vertex in $\mathsf{dif}(f, f')$. The *length* $\mathsf{len}_w(\mathcal{S})$ *of* a reconfiguration sequence $\mathcal{S} = \langle f_0, f_1, \ldots, f_\ell \rangle$ is defined as $\mathsf{len}_w(\mathcal{S}) = \sum_{i=1}^{\ell} \mathsf{gap}_w(f_{i-1}, f_i)$. We denote by $\mathsf{OPT}(\mathcal{I}, w)$ the minimum length of a reconfiguration sequence between f_0 and f_t; we define $\mathsf{OPT}(\mathcal{I}, w) = +\infty$ if $\mathcal{I}$ is a no-instance of LIST COLORING RECONFIGURATION. Then, LIST COLORING SHORTEST RECONFIGURATION can be seen as computing $\mathsf{OPT}(\mathcal{I}, w)$ for the case where every vertex has weight one. Thus, to prove Theorem 8, it suffices to construct a fixed-parameter algorithm for the weighted version when parameterized by $k + \mathsf{vc}$.

As with Section 3, we again use the concept of kernelization to prove Theorem 8. More precisely, for a given instance $(\mathcal{I}, w)$, we first construct an instance $(\mathcal{I}' = (G', L', f'_0, f'_t), w')$ in

polynomial time such that the size of $\mathcal{I}'$ depends only on $k+\mathsf{vc}$, and $\mathsf{OPT}(\mathcal{I}', w') = \mathsf{OPT}(\mathcal{I}, w)$ holds. Then, we can compute $\mathsf{OPT}(\mathcal{I}', w')$ by computing a (weighted) shortest path between f'_0 and f'_t in an edge-weighted graph defined as follows: the vertex set consists of all L'-colorings of G', and each pair of adjacent L'-colorings are connected by an edge with a weight corresponding to the gap between them.

4.1 Reduction rule for the weighted version

In this subsection, we give the counterpart of Lemma 5 for the weighted version.

Let $(\mathcal{I} = (G, L, f_0, f_t), w)$ be an instance of the weighted version, and assume that there exist two identical subgraphs H_1 and H_2 of G, both of which consist of single vertices, say, $V(H_1) = \{v_1\}$ and $V(H_2) = \{v_2\}$. We now define a new instance $(\mathcal{I}', w')$ as follows:

- $\mathcal{I}' = \mathcal{I}^{G \setminus H_2}$; and
- $w'(v_1) = w(v_1) + w(v_2)$ and $w'(v) = w(v)$ for any $v \in V(G) \setminus \{v_1, v_2\}$.

Intuitively, v_2 is merged into v_1 together with its weight. Then, we have the following lemma.

▶ **Lemma 10.** $\mathsf{OPT}(\mathcal{I}, w) = \mathsf{OPT}(\mathcal{I}', w')$.

4.2 Kernelization

Finally, we give a kernelization algorithm as follows.

Let $(\mathcal{I} = (G, L, f_0, f_t), w)$ be an instance of the weighted version such that G has a vertex cover of size at most vc. Because such a vertex cover can be computed in time $O(2^{\mathsf{vc}} \cdot |V(G)|)$ [8], we now assume that we are given a vertex cover V_C of size at most vc. Notice that $V_I := V \setminus V_C$ forms an independent set of G. Suppose that there exist two vertices $v_1, v_2 \in V_I$ such that $N(G, v_1) = N(G, v_2)$ and $A(v_1) = A(v_2)$ hold. Then, induced subgraphs $G[\{v_1\}]$ and $G[\{v_2\}]$ are identical. Therefore, we can apply Lemma 10 to remove v_2 from G, and modify a weight function without changing the optimality. As a kernelization, we repeatedly apply Lemma 10 for all such pairs of vertices in V_I, which can be done in polynomial time. Let G' be the resulting subgraph of G, and let $V'_I := V(G') \setminus V_C$. Since V_C is of size at most vc, it suffices to prove the following lemma.

▶ **Lemma 11.** $|V'_I| \leq 2^{\mathsf{vc}} \cdot 2^k \cdot k^2$.

This completes the proof of Theorem 8.

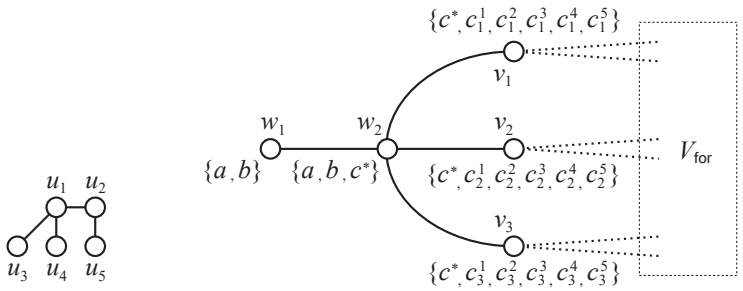
5 W[1]-Hardness

Because even the shortest variant is fixed-parameter tractable when parameterized by $k + \mathsf{vc}$, one may expect that vc is a strong parameter and the problem is fixed-parameter tractable with only vc. However, we prove the following theorem in this section.

▶ **Theorem 12.** List coloring reconfiguration *is W[1]-hard when parameterized by* vc*, where* vc *is the upper bound on the size of a minimum vertex cover of an input graph.*

Recall that list coloring reconfiguration is PSPACE-complete even for a fixed constant $k \geq 4$. Therefore, the problem is intractable if we take only one parameter, either k or vc.

In order to prove Theorem 12, we give an FPT-reduction from the independent set problem when parameterized by the solution size s, in which we are given a graph H and an integer $s \geq 0$, and asked whether H has an independent set of size at least s. This problem is known to be W[1]-hard [8].



(a) H , $s = 3$ (b) G , L

Figure 7 (a) An instance H of INDEPENDENT SET, and (b) the graph G and the list L. The set V_{for} contains vertices of $(i, j; p, q)$-forbidding gadgets for all $(i, j) \in \{(1, 2), (1, 3), (2, 3)\}$ and all $(p, q) \in \{(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (1, 2), (2, 1), (1, 3), (3, 1), (1, 4), (4, 1), (2, 5), (5, 2)\}$; thus $|V_{\mathsf{for}}| = 39$.

5.1 Construction

Let H be a graph with n vertices $u_1, u_2, \ldots, u_n$, and s be an integer as an input for INDEPENDENT SET. Then, we construct the corresponding instance (G, L, f_0, f_t) of LIST COLORING RECONFIGURATION as follows. (See also Figure 7.)

We first create s vertices $v_1, v_2, \ldots, v_s$, which are called *selection vertices*; let V_{sel} be the set of all selection vertices. For each $i \in \{1, 2, \ldots, s\}$, we set $L(v_i) = \{c^*, c_i^1, c_i^2, \ldots, c_i^n\}$. In our reduction, we will construct G and L so that assigning the color c_i^p, $p \in \{1, 2, \ldots, n\}$, to $v_i \in V_{\mathsf{sel}}$ corresponds to choosing the vertex $u_p \in V(H)$ as a vertex in an independent set of H. Then, in order to make a correspondence between a color assignment to V_{sel} and an independent set of size s in H, we need to construct the following properties:

- For each $p \in \{1, 2, \ldots, n\}$, we use at most one color from $\{c_1^p, c_2^p, \ldots, c_s^p\}$; this ensures that each vertex $u_p \in V(H)$ can be chosen at most once in an independent set.
- For each $p, q \in \{1, 2, \ldots, n\}$ with $u_p u_q \in E(H)$, we use at most one color from $\{c_1^p, c_2^p, \ldots, c_s^p, c_1^q, c_2^q, \ldots, c_s^q\}$; then, no two adjacent vertices in H are chosen in an independent set.

To do this, we define an $(i, j; p, q)$-forbidding gadget for $i, j \in \{1, 2, \ldots s\}$ and $p, q \in \{1, 2, \ldots, n\}$. The $(i, j; p, q)$-*forbidding gadget* is a vertex w which is adjacent to v_i and v_j and has a list $L(w) = \{c_i^q, c_j^p\}$. Observe that the vertex w forbids that v_i and v_j are simultaneously colored with c_i^p and c_j^q, respectively. In order to satisfy the desired properties above, we now add our gadgets as follows: for all $i, j \in \{1, 2, \ldots s\}$ with $i < j$,

- add an $(i, j; p, p)$-forbidding gadget for every vertex $u_p \in V(H)$; and
- add $(i, j; p, q)$- and $(i, j; q, p)$-forbidding gadgets for every edge $u_p u_q \in E(H)$.

We denote by V_{for} the set of all vertices in the forbidding gadgets. We finally create an edge consisting of two vertices w_1 and w_2 such that $L(w_1) = \{a, b\}$ and $L(w_2) = \{a, b, c^*\}$, and connect w_2 with all selection vertices in V_{sel}.

Finally, we construct two L-colorings f_0 and f_t of G as follows:

- for each $v_i \in V_{\mathsf{sel}}$, $f_0(v_i) = f_t(v_i) = c^*$;
- for each $w \in V_{\mathsf{for}}$, $f_0(w)$ and $f_t(w)$ are arbitrary chosen colors from $L(w)$; and
- $f_0(w_1) = f_t(w_2) = a$, and $f_t(w_1) = f_0(w_2) = b$.

Note that both f_0 and f_t are proper L-colorings of G. This completes the construction of (G, L, f_0, f_t).

5.2 Correctness of the reduction

In this subsection, we prove the following three statements:

- (G, L, f_0, f_t) can be constructed in time polynomial in the size of H.
- The upper bound vc on the size of a minimum vertex cover of G depends only on s.
- H is a yes-instance of INDEPENDENT SET if and only if (G, L, f_0, f_t) is a yes-instance of LIST COLORING RECONFIGURATION.

In order to prove the first statement, it suffices to show that the size of (G, L, f_0, f_t) is bounded polynomially in $n = |V(H)|$. From the construction, we have $|V(G)| = |V_{\mathsf{sel}}| + |V_{\mathsf{for}}| + |\{w_1, w_2\}| \leq s + s^2 \times (|V(H)| + 2|E(H)|) + 2 = O(n^4)$. In addition, each list contains $O(n)$ colors. Therefore, the construction can be done in time $O(n^{O(1)})$.

The second statement immediately follows from the fact that $\{w_2\} \cup V_{\mathsf{sel}}$ is a vertex cover in G of size $s + 1$; observe that $G \setminus (\{w_2\} \cup V_{\mathsf{sel}}) = G[\{w_1\} \cup V_{\mathsf{for}}]$ contains no edge.

Finally, we prove the third statement as follows.

▶ **Lemma 13.** *H is a yes-instance of* INDEPENDENT SET *if and only if (G, L, f_0, f_t) is a yes-instance of* LIST COLORING RECONFIGURATION.

This completes the proof of Theorem 12.

6 Conclusion

In this paper, we have studied LIST COLORING RECONFIGURATION from the viewpoint of parameterized complexity, in particular, with several graph parameters. We painted an interesting map of graph parameters in Figure 2 which shows the boundary between fixed-parameter tractability and intractability.

References

1. Marthe Bonamy and Nicolas Bousquet. Recoloring bounded treewidth graphs. *Electronic Notes in Discrete Mathematics*, 44:257–262, 2013. doi:10.1016/j.endm.2013.10.040.
2. Marthe Bonamy, Matthew Johnson, Ioannis Lignos, Viresh Patel, and Daniël Paulusma. Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *Journal of Combinatorial Optimization*, 27(1):132–143, 2014. doi:10.1007/s10878-012-9490-y.
3. Paul Bonsma and Luis Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009. doi:10.1016/j.tcs.2009.08.023.
4. Paul Bonsma, Amer E. Mouawad, Naomi Nishimura, and Venkatesh Raman. The complexity of bounded length graph recoloring and CSP reconfiguration. In *Parameterized and Exact Computation - 9th International Symposium, IPEC 2014, Wroclaw, Poland, September 10-12, 2014. Revised Selected Papers*, pages 110–121, 2014. doi:10.1007/978-3-319-13524-3_10.
5. Richard C. Brewster, Sean McGuinness, Benjamin Moore, and Jonathan A. Noel. A dichotomy theorem for circular colouring reconfiguration. *Theoretical Computer Science*, 639:1–13, 2016. doi:10.1016/j.tcs.2016.05.015.
6. Luis Cereceda. *Mixing Graph Colourings*. PhD thesis, The London School of Economics and Political Science, 2007.
7. Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011. doi:10.1002/jgt.20514.
8. Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

9 Martin Dyer, Abraham D. Flaxman, Alan M. Frieze, and Eric Vigoda. Randomly coloring sparse random graphs with fewer colors than the maximum degree. *Random Structures & Algorithms*, 29(4):450–465, 2006. doi:10.1002/rsa.20129.

10 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, pages 163–176, 2013. doi:10.1007/978-3-319-03898-8_15.

11 Tibor Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(1):25–66, 1967. doi:10.1007/BF02020961.

12 Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010. doi:10.1016/j.cosrev.2010.01.001.

13 Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. The list coloring reconfiguration problem for bounded pathwidth graphs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E98.A(6):1168–1178, 2015. doi:10.1587/transfun.E98.A.1168.

14 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):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

15 Matthew Johnson, Dieter Kratsch, Stefan Kratsch, Viresh Patel, and Daniël Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321, 2016. doi:10.1007/s00453-015-0009-7.

16 Sampath Kannan, Moni Naor, and Steven Rudich. Implicat representation of graphs. *SIAM Journal on Discrete Mathematics*, 5(4):596–603, 1992. doi:10.1137/0405049.

17 Ross M. McConnell and Fabien de Montgolfier. Linear-time modular decomposition of directed graphs. *Discrete Applied Mathematics*, 145(2):198–209, 2005. doi:10.1016/j.dam.2004.02.017.

18 Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, pages 127–160. 2013. doi:10.1017/CBO9781139506748.005.

19 Marcin Wrochna. Reconfiguration in bounded bandwidth and treedepth. *CoRR*, abs/1405.0847, 2014.

20 Marcin Wrochna. Homomorphism reconfiguration via homotopy. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 730–742, 2015. doi:10.4230/LIPIcs.STACS.2015.730.

Automata in the Category of Glued Vector Spaces*†

Thomas Colcombet[1] and Daniela Petrişan[2]

1 **CNRS, IRIF, Univ. Paris-Diderot, Paris 7, France**
`thomas.colcombet@irif.fr`
2 **CNRS, IRIF, Univ. Paris-Diderot, Paris 7, France**
`petrisan@irif.fr`

Abstract

In this paper we adopt a category-theoretic approach to the conception of automata classes enjoying minimization by design. The main instantiation of our construction is a new class of automata that are hybrid between deterministic automata and automata weighted over a field.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases hybrid set-vector automata, automata minimization in a category

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.52

1 Introduction

In this paper we introduce a new automata model, *hybrid set-vector automata*, designed to accept weighted languages over a field in a more efficient way than Schützenberger's weighted automata [13]. The space of states for these automata is not a vector space, but rather a union of vector spaces "glued" together along subspaces. We call them hybrid automata, since they naturally embed both deterministic finite state automata and finite automata weighted over a field. In Section 2 we present at an informal level a motivating example and the intuitions behind this construction, avoiding as much as possible category-theoretical technicalities. We use this example to guide us throughout the rest of the paper.

A key property that the new automata model should satisfy is minimization. Since the morphisms of "glued" vector spaces are rather complicated to describe, proving the existence of minimal automata "by hand" is rather complicated. Therefore we opted for a more systematic approach and adopted a category-theoretic perspective for designing *new forms of automata* that enjoy *minimization by design*. In particular, we introduce the category of "glued" vector spaces in which these automata should live and we analyse its properties that render minimization possible.

Starting with the seminal papers of Arbib and Manes, see for example [3] and the references therein, and of Goguen [10], it became well established that category theory offers a neat understanding of several phenomena in automata theory. In particular, the key property of minimization in different contexts, such as for deterministic automata (over finite words)

* This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624), and by the DeLTA ANR project (ANR-16-CE40-0007). The authors also thank the Simons Institute for the Theory of Computing where this work has been partly developed.

† A version using the knowledge package is available at `https://arxiv.org/abs/1711.06065`.

and Schützenberger's automata weighted over fields [13], arises from the same categorical reasons (existence of some limits/colimits and an (epi,mono)-factorization system [3]).

There is a long tradition of seeing automata either as algebras or coalgebras for a functor. However, in the case of deterministic automata, the algebraic view does not capture the accepting states, while the coalgebraic view does not capture the initial state. In the coalgebraic setting one needs to consider the so-called pointed coalgebras, see for example [1], where minimal automata are modelled as well-pointed coalgebras. The dual perspective of automata seen as both algebras or coalgebras, as well as the duality between reachability and observability, has been explored more recently in papers such as [4, 5, 6].

Here we take yet another approach to defining automata in a category. The reader acquainted with category theory will recognise that we see automata as functors from an input category (that specifies the type of the machines under consideration, which in this paper is restricted to word automata) to a category of output values. We show that the next ingredients are sufficient to ensure minimization: the existence of an initial and of a final automaton for a language, and a factorization system on the category in which we interpret our automata.

For example, deterministic and weighted automata over a field are obtained by considering as output categories the categories Set of sets and functions and Vec of vector spaces and linear maps, respectively. Since Set and Vec have all limits and colimits, it is very easy to prove the existence of initial and final automata accepting a given language. In both cases, the minimal automaton for a language is obtained by taking an epi-mono factorization of the unique arrow from the initial to the final automaton.

Notice that the initial and the final automata have infinite (-dimensional) state sets (spaces). If the language at issue is regular, that is, if the unique map from the initial to the final automaton factors through a finite (-dimensional) automaton then, automatically, the minimal automaton will also be finite (-dimensional). However, this relies on very specific properties of the categories of sets and vector spaces, namely on the fact that the full subcategories $\mathsf{Set}_{\mathsf{fin}}$ of finite sets and $\mathsf{Vec}_{\mathsf{fin}}$ of finite dimensional vector spaces are closed in Set, respectively in Vec, under both quotients and subobjects.

Coming back to hybrid set-vector automata, we define them as word automata interpreted in an output category $\mathtt{Glue}(\mathsf{Vec})$ which we obtain as the completion of Vec under certain colimits, and can be described at an informal level as "glueings" of arbitrary vector spaces. The definition of this form of cocompletion $\mathtt{Glue}(\mathcal{C})$ of a category $\mathcal{C}$ is the subject of Section 4.

We are interested in those hybrid automata for which the state object admits a finitary description, which intuitively can be described as *finite* glueings of *finite dimensional* spaces. For this reason we will consider the subcategory $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$ of $\mathtt{Glue}(\mathsf{Vec})$. It turns out that $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$ is closed under quotients in $\mathtt{Glue}(\mathsf{Vec})$ but, crucially, *it is not closed under subobjects*. For example, a glueing of infinitely many one-dimensional spaces is a subobject of a two-dimensional space, but only the latter is an object of $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$.

This is the motivation for introducing a notion of *$(E_{\mathcal{S}},M_{\mathcal{S}})$-factorization of a category $\mathcal{C}$ through a subcategory $\mathcal{S}$*. This is a refinement of the classical notion of factorization system on $\mathcal{C}$ and is used for isolating the semantical computations (in $\mathcal{C}$) from the automata themselves (with an object from $\mathcal{S}$ as "set of configurations").[1] We show how it provides a minimization of "$\mathcal{S}$-automata for representing $\mathcal{C}$-languages". A concrete instance of this is a factorization system on $\mathtt{Glue}(\mathsf{Vec})$ through $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$, which plays a crucial role in proving the

[1] This distinction is usually not necessary, and we are not aware of its existence in the literature. It is crucial for us, thus we cannot use already existing results from the coalgebraic literature, e.g. [1].

existence of minimal $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$-automata for recognizing weighted languages.

The rest of the paper is organised as follows. We first develop a motivating example of a hybrid set-vector space automaton in Section 2. We then identify in Section 3 the category-theoretic ingredients that are sufficient for a class of automata to enjoy minimization. We then turn to our main contribution, namely the description and the study of the properties of (finite-)mono-diagrams in a category, in Section 4. We conclude in Section 5 with a discussion of some of the design choices we made in this paper.

2 The hybridisation of deterministic finite state and vector automata

In this section, we (rather informally) describe the motivating example of this paper: the construction of a family of automata that naturally extends both deterministic finite state automata and finite automata weighted over a field in the sense of Schützenberger (i.e., automata in the category of finite vector spaces). The intuition should then support the categorical constructions that we develop in the subsequent sections.

Set automata (deterministic automata). Let us fix ourselves an alphabet $\mathtt{A}$. A *deterministic automaton* (or *set automaton*) is a tuple

$$\mathcal{A} = (Q, i, f, (\delta_a)_{a\in\mathtt{A}})\,,$$

in which Q is a *set of states*, i is a map from a one element set $1 = \{0\}$ to Q (i.e. an *initial state*), f is a map from Q to a two elements set $2 = \{0, 1\}$ (i.e. a *set of accepting states*), and δ_a is a map from Q to itself for all letters $a \in \mathtt{A}$. Given a word $u = a_1 \dots a_n$, the automaton *accepts* the map $[\![\mathcal{A}]\!](u)\colon 1 \to 2$ defined as:

$$[\![\mathcal{A}]\!](u) = f \circ \delta_u \circ i \qquad \text{where} \quad \delta_{a_1\dots a_n} = \delta_{a_n} \circ \dots \circ \delta_{a_1}\,.$$

We recognize here the standard definition of a deterministic automaton, in which a word u is accepted if the map $[\![\mathcal{A}]\!](u)$ is the constant 1, and rejected if it is the constant 0.

Vector space automata (automata weighted over a field). Now, we can use the same definition of an automaton, this time with Q a vector space (over, say, the field $\mathbb{R}$), i a linear map from $\mathbb{R}$ to Q, f a linear map from Q to $\mathbb{R}$ (seen as a $\mathbb{R}$-vector space as usual), and δ_a a linear map from Q to itself. In other words, we have used the same definition, but this time in the category of vector spaces. Given a word u, a vector space automaton $\mathcal{A}$ computes $[\![\mathcal{A}]\!](u)\colon \mathbb{R} \to \mathbb{R}$ as the composite described above. Since a linear map from $\mathbb{R}$ to $\mathbb{R}$ is only determined by the image of 1, this automaton can be understood as associating to each input word u the real number $[\![\mathcal{A}]\!](u)(1)$. We will informally refer to such automata in this section as *vector space automata*. Let us provide an example.

Leading example. For a word $u \in \{a, b, c\}^*$ let $|u|_a$ denote the number of occurrences of the letter a in u. Let us compute the map F which, given a word $u \in \{a, b, c\}^*$, outputs $2^{|u|_a}$ if it contains an even number of b's and no c's, and 0 in all other cases. This is achieved with the vector space automaton $\mathcal{A}^{\mathsf{vec}} = (Q^{\mathsf{vec}} = \mathbb{R}^2, i^{\mathsf{vec}}, f^{\mathsf{vec}}, \delta^{\mathsf{vec}})$ where for all $x, y \in \mathbb{R}$,

$$i^{\mathsf{vec}}(x) = (x, 0)\,, \qquad \delta_a^{\mathsf{vec}}(x, y) = (2x, 2y)\,, \qquad \delta_b^{\mathsf{vec}}(x, y) = (y, x)\,,$$
$$f^{\mathsf{vec}}(x, y) = x\,, \qquad \delta_c^{\mathsf{vec}}(x, y) = (0, 0)\,.$$

One easily checks that indeed $[\![\mathcal{A}^{\mathsf{vec}}]\!](u)(1) = F(u)$ for all words $u \in \mathtt{A}^*$.

Can we do better? It is well known from Schützenberger's seminal work that the vector space automaton $\mathcal{A}^{\texttt{vec}}$ is minimal, both in an algebraic sense (to be described later) as well as at an intuitive level in the sense that no vector space automaton could recognize F with a dimension one vector space as configuration space: $\mathcal{A}^{\texttt{vec}}$ is "dimension minimal."

However, let us think for one moment on how one would "implement" the function F as an online device that would get letters as input, and would modify its internal state accordingly. Would we implement concretely $\mathcal{A}^{\texttt{vec}}$ directly? Probably not, since there is a more economic[2] way to obtain the same result: we can maintain 2^m where m is the number of a's seen so far, together with one bit for remembering whether the number of b's is even or odd. Such an automaton would start with 1 in its unique real valued register. Each time an a is met, the register is doubled, each time b is met, the bit is reversed, and when c is met, the register is set to 0. At the end of the input word, the automaton would output 0 or the value of the register depending on the current value of the bit. If we consider the configuration space that we use in this encoding, we use $\mathbb{R} \uplus \mathbb{R}$ instead of $\mathbb{R} \times \mathbb{R}$. Can we define an automata model that would faithfully implement this example?

A first generalization: disjoint unions of vector spaces. A way to achieve this is to interpret the generic notion of automata in the category of finite disjoint unions of vector spaces (*duvs*). One way to define such a *finite disjoint unions of vector spaces* is to use a finite set N of '*indices*' $p, q, r \ldots$, and to each index p associate a vector space V_p. The 'space' represented is then $\{(p, \vec{v}) \mid p \in N,\ \vec{v} \in V_p\}$. A 'map' between duvs represented by (N, V) and (N', V') is then a pair $h : N \to N'$ together with a linear map f_p from V_p to $V'_{h(p)}$ for all $p \in N$. It can be seen as mapping each $(p, \vec{v}) \in N \times V_p$ to $(h(p), f_p(\vec{v}))$. Call this a *duvs map*. Such duvs maps are composed in a natural way. This defines a category, and hence we can consider *duvs automata* which are automata with a duvs for its state space, and transitions implemented by duvs maps.

For instance, we can pursue with the computation of F and provide a duvs automaton $\mathcal{A}^{\texttt{duvs}} = (Q^{\texttt{duvs}}, i^{\texttt{duvs}}, f^{\texttt{duvs}}, \delta^{\texttt{duvs}})$ where $Q^{\texttt{duvs}} = \{(s, x) \mid s \in \{\texttt{even}, \texttt{odd}\},\ x \in \mathbb{R}\}$ (considered as a disjoint union of vector spaces with indices `even` and `odd` and all associated vector spaces $V_{\texttt{even}} = V_{\texttt{odd}} = \mathbb{R}$). The maps can be conveniently defined as follows:

$$
\begin{array}{lll}
i^{\texttt{duvs}}(x) = (\texttt{even}, x) & \delta_a^{\texttt{duvs}}(\texttt{even}, x) = (\texttt{even}, 2x) & \delta_a^{\texttt{duvs}}(\texttt{odd}, x) = (\texttt{odd}, 2x) \\
f^{\texttt{duvs}}(\texttt{even}, x) = x & \delta_b^{\texttt{duvs}}(\texttt{even}, x) = (\texttt{odd}, x) & \delta_b^{\texttt{duvs}}(\texttt{odd}, x) = (\texttt{even}, x) \\
f^{\texttt{duvs}}(\texttt{odd}, x) = 0 & \delta_c^{\texttt{duvs}}(\texttt{even}, x) = (\texttt{even}, 0) & \delta_c^{\texttt{duvs}}(\texttt{odd}, x) = (\texttt{odd}, 0)
\end{array}
$$

This automaton computes the expected F. It is also obvious that such automata over finite disjoint unions of vector spaces generalize both deterministic finite state automata (using only 0-dimensional vector spaces), and vector space automata (using only one index). However, is it the joint generalization that we hoped for? The answer is no...

Minimization of duvs automata. We could think that the above automaton $\mathcal{A}^{\texttt{duvs}}$ is minimal. However, it involved some arbitrary decisions when defining it. This can be seen in the fact that when $\delta_c^{\texttt{duvs}}$ is applied, we chose to not change the index (and set to null the real value): this is arbitrary, and we could have exchanged `even` and `odd`, or fixed it abitrarily to `even`, or to `odd`. All these *variants* would be equally valid for computing F.

[2] Under the reasonable assumption that maintaining a real is more costly than maintaining a bit.

It is a bit difficult at this stage to explain the non-minimality of these automata since we did not introduce the proper notions yet. Let us try at a high level, invoking some standard automata-theoretic concepts. The first remark is that every configuration in $Q^{\mathtt{duvs}}$ is 'reachable' in this automaton: indeed $(\mathtt{even}, x) = i^{\mathtt{duvs}}(x)$ and $(\mathtt{odd}, x) = \delta_b^{\mathtt{duvs}} \circ i^{\mathtt{duvs}}(x)$ for all $x \in \mathbb{R}$. Hence there is no hope to improve the automaton $\mathcal{A}^{\mathtt{duvs}}$ or one of its variants by some form of 'restriction to its reachable configurations'. Only 'quotienting of configurations' remains. However, one can show that none among $\mathcal{A}^{\mathtt{duvs}}$ and the variants mentioned above is the quotient of another. Keeping in mind the Myhill-Nerode equivalence, we should instead merge the configurations $(\mathtt{even}, 0)$ and $(\mathtt{odd}, 0)$ since these are observationally equivalent:

$$f^{\mathtt{duvs}} \circ \delta_u^{\mathtt{duvs}}(\mathtt{even}, 0) = 0 = f^{\mathtt{duvs}} \circ \delta_u^{\mathtt{duvs}}(\mathtt{odd}, 0) \qquad \text{for all words } u \in \mathtt{A}^*.$$

However, the quotient duvs obtained by merging $(\mathtt{even}, 0)$ and $(\mathtt{odd}, 0)$, albeit not very intuitive, consists of one index associated to a two dimensional vector space, which is essentially an indexed version of the vector space automaton $\mathcal{A}^{\mathtt{vec}}$ computed before. At this stage, we understand that minimising in the category of duvs is not very helpful, as we do not obtain the desired optimisation.

How to proceed from here. The only reasonable thing to do is indeed to merge $(\mathtt{even}, 0)$ and $(\mathtt{odd}, 0)$, but we have to be more careful about the precise meaning of 'quotient'. A possibility is to add explicitly equivalence classes in the definition of the automaton. However, category theory provides useful concepts and terminology for defining these objects: colimits, and more precisely the free co-completion of a category. In the previous paragraph, we have shown that the category of duvs – which is itself the free completion of Vec with respect to finite coproducts – is not a good ambient category for our purposes. We need more colimits, so that the notion of 'quotient' is further refined. At the other extreme, we could consider the free completion with respect to all colimits, which, informally, consists of objects obtained from the category using copying and gluing. We will explain later in Section 5 why we choose to not use this completion. Intuitively, by adding all colimits we glue the vector spaces "too much", and not only we loose a geometric intuition of the objects we are dealing with, but we may run into actual technical problems when it comes the existence of minimal automata.

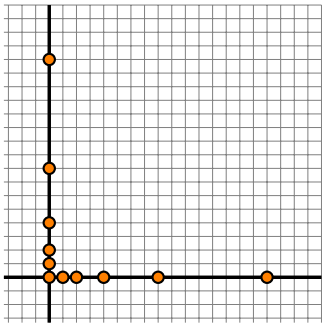
Instead, we restrict our attention to a class of colimits (which strictly contains coproducts) for which different spaces in the colimit can be "glued" together along subspaces, but which do not contain implicit self folding (i.e., such that an element of a vector space is not glued to a distinct element of the same vector space, directly or indirectly). E.g., we can describe 'two one-dimensional spaces, the 0-dimensional subspaces of which are identified through a linear bijection'. In this way we obtain the new category of *glued vector spaces* and *hybrid set-vector space automata*, corresponding to $\mathtt{Glue}(\mathsf{Vec})$-automata in the rest of the paper.

Generic arguments of colimits provide the language for describing these objects, but do not solve the question of minimality. In particular, we are interested in automata whose space of configurations is a *finite* colimit belonging to the class described above. The categorical development in this work addresses the minimization problem for hybrid automata.

An intuition in the case of gluing of vector spaces. In the case of gluing of vector spaces, it is possible to isolate a combinatorial statement that plays a crucial role in the existence of minimal hybrid set-vector automata:

> *(a)* Any subset of a finite-dimensional vector space admits a minimal cover as a finite union of subspaces. *(b)* Furthermore, there is a unique such cover which is a union of subspaces which are incomparable with respect to inclusion.

For instance, in the original vector space automaton $\mathcal{A}^{\mathsf{vec}}$, the states that are reachable in fact all belong to $\mathbb{R} \times \{0\} \cup \{0\} \times \mathbb{R}$, and this is the minimal cover as in (a) of these reachable configurations. This subset of $\mathbb{R}^2$ has the structure of two $\mathbb{R}$-spaces. These happen to intersect at $(0, 0)$, hence it is necessary to glue them at 0 to faithfully represent this set of reachable configurations. Thanks to (b) this decomposition is canonical, and hence can be used for describing the automaton.



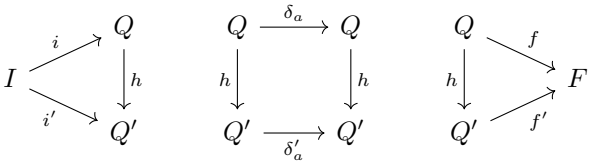
3 Automata in a category

In this section, we provide the general definition for a (finite word) automaton in a category. We also isolate properties guaranteeing the existence of minimal automata. Though presented differently, the material in the first subsection is essentially a slight variation around the work of Arbib and Manes [3], which introduced a notion of automaton in a category and, moreover, highlighted the connection between factorization systems of the ambient category, duality and minimization. In the remaining subsections we develop a refinement of this approach to minimization, and introduce a notion of factorization system through a subcategory.

3.1 Automata in a category, initial automaton, final automaton

▶ **Definition 3.1** (automata)**.** Let $\mathcal{C}$ be a locally small category, I and F be objects of $\mathcal{C}$, and $\mathtt{A}$ be some alphabet. An *automaton $\mathcal{A}$ in the category $\mathcal{C}$* (over the alphabet $\mathtt{A}$), for short a *$\mathcal{C}, I, F$-automaton* (or simply *$\mathcal{C}$-automaton* when I and F are obvious in the context), is a tuple (Q, i, f, δ), where Q is an object in $\mathcal{C}$ (called the *state object*), $i \colon I \to Q$ and $f \colon Q \to F$ are morphisms in $\mathcal{C}$ (called *initial* and *final morphisms*), and $\delta \colon \mathtt{A} \to \mathcal{C}(Q, Q)$ is a function associating to each letter $a \in \mathtt{A}$ a morphism $\delta_a \colon Q \to Q$ in $\mathcal{C}$. We extend the function δ to $\mathtt{A}^*$ as with δ_ϵ being the identity morphism on Q and $\delta_{wa} = \delta_a \circ \delta_w$ for all $a \in \mathtt{A}$ and $w \in \mathtt{A}^*$.

A *morphism of $\mathcal{C}, I, F$-automata* $h \colon \mathcal{A} \to \mathcal{A}'$ is a morphism $h \colon Q \to Q'$ in $\mathcal{C}$ between the state objects which commutes with the initial, final and transition morphisms:



(1)

▶ **Example 3.2.** The two guiding instantiation of this definition are as follows. When the category $\mathcal{C}$ is Set, $I = 1$ and $F = 2$, we recover the standard notion of a deterministic and complete automaton (over the alphabet $\mathtt{A}^*$). In the second case, when $\mathcal{C}$ is Vec over a base field $\mathbb{K}$, $I = \mathbb{K}$ and $F = \mathbb{K}$, we obtain $\mathbb{K}$-weighted automata. Indeed, if Q is isomorphic to $\mathbb{K}^n$ for some natural number n, then linear maps $i \colon \mathbb{K} \to Q$ are in one-to-one correspondence

with vectors $\mathbb{K}^n$. The same holds for linear maps $f\colon Q \to \mathbb{K}$, hence i and f are simply selecting an initial, respectively, a final vector.

▶ **Definition 3.3** (languages and language accepted). A $\mathcal{C}, I, F$-*language* (or $\mathcal{C}$-*language* when I and F are clear from the context) is a function $L\colon \mathtt{A}^* \to \mathcal{C}(I, F)$. We say that $\mathcal{A}$ *accepts the language* L if $L(w) = [\![\mathcal{A}]\!](w) := f \circ \delta_w \circ i$ for all $w \in \mathtt{A}^*$. Let $\mathsf{Auto}_{\mathcal{C}}(L)$ denote the *category of* $\mathcal{C}, I, F$*-automata for* L, that is, the category whose objects are $\mathcal{C}, I, F$-automata that accept the language L and whose arrows are morphisms of $\mathcal{C}, I, F$-automata[3].

▶ **Lemma 3.4.** *If the coproduct* $\coprod_{w\in\mathtt{A}^*} I$ *exists in* $\mathcal{C}$*, then* $\mathsf{Auto}_{\mathcal{C}}(L)$ *has an initial object* $\mathtt{init}_{\mathcal{C}}(L)$*. If the product* $\prod_{w\in\mathtt{A}^*} F$ *exists in* $\mathcal{C}$*, then* $\mathsf{Auto}_{\mathcal{C}}(L)$ *has a final object* $\mathtt{final}_{\mathcal{C}}(L)$*.*

In the case of Set, these automata are well known. The first one has as states $\mathtt{A}^*$, as initial state ε, and when it reads a letter a, its maps w to wa. Its final map sends the state w to $L(w)(0)$. There exists one and exactly one morphism from this automaton to each automata for the same language. The generalisation of this construction is that the state space is the coproduct of A^*-many copies of I. The final automaton is known as the automaton of 'residuals'. Its set of states are the maps from $\mathtt{A}^*$ to 2. The initial state is L itself, and when reading a letter a, the state S is mapped to $w \mapsto S(aw)$. The final map sends S to $S(\varepsilon)$. The generalisation of this construction is that the state space is the product of A^*-many copies of F.

3.2 Factorizations through a subcategory

It is important in the development of this paper to distinguish the category $\mathsf{Auto}_{\mathcal{C}}(L)$ in which the initial and final automata for a language L exist (recall Lemma 3.4) and which contains 'infinite automata', from the subcategory, named $\mathsf{Auto}_{\mathcal{S}}(L)$ that is used for the concrete automata (with state object in $\mathcal{S}$) which are intended to be algorithmically manageable. In this section, we provide the concept of factorizing through a subcategory, which articulates the relation between these two categories.

▶ **Definition 3.5** (factorization through a subcategory). Assume $\mathcal{S}$ is a subcategory of $\mathcal{C}$. An arrow $f\colon X \to Y$ in $\mathcal{C}$ is called $\mathcal{S}$-*small* if it factors through some object S of $\mathcal{S}$, that is, f is the composite $X \xrightarrow{u} S \xrightarrow{v} Y$ for some $u\colon X \to S$ and $v\colon S \to Y$.

A *factorization system through* $\mathcal{S}$ *on* $\mathcal{C}$ (or simply a *factorization system on* $\mathcal{C}$ if $\mathcal{C} = \mathcal{S}$) is a pair $(E_{\mathcal{S}}, M_{\mathcal{S}})$ where $E_{\mathcal{S}}$ and $M_{\mathcal{S}}$ are classes of arrows in $\mathcal{C}$ so that the codomains of all arrows in $E_{\mathcal{S}}$, the domains of all arrows in $M_{\mathcal{S}}$ are in $\mathcal{S}$, and the following conditions hold:

1. $E_{\mathcal{S}}$ and $M_{\mathcal{S}}$ are closed under composition with isomorphisms in $\mathcal{S}$, on the right, respectively left side.
2. All $\mathcal{S}$-small arrows in $\mathcal{C}$ have an $(E_{\mathcal{S}}, M_{\mathcal{S}})$-*factorization*, that is, if $f\colon X \to Y$ factors through an object of $\mathcal{S}$, then there exists $e \in E_{\mathcal{S}}$ and $m \in M_{\mathcal{S}}$, such that $f = m \circ e$.
3. The *unique* $(E_{\mathcal{S}}, M_{\mathcal{S}})$-*diagonalization property* holds: for each commutative diagram

$$\begin{array}{ccc} X & \xrightarrow{e} & T \\ {\scriptstyle f}\downarrow & \overset{u}{\swarrow} & \downarrow{\scriptstyle g} \\ S & \xrightarrow[m]{} & Y \end{array} \tag{2}$$

[3] If $\mathcal{A}$ accepts the language L and $h\colon \mathcal{A} \to \mathcal{A}'$ is a morphism of $\mathcal{C}, I, F$-automata, then $\mathcal{A}'$ also accepts the language L. Hence, $\mathsf{Auto}_{\mathcal{C}}(L)$ is a 'connected component' in the category of all $\mathcal{C}, I, F$-automata.

with $e \in E_{\mathcal{S}}$ and $m \in M_{\mathcal{S}}$, there exists a unique *diagonal*, that is, a unique morphism $u\colon T \to S$ such that $u \circ e = f$ and $m \circ u = g$.

Using standard techniques, we can prove that whenever $(E_{\mathcal{S}}, M_{\mathcal{S}})$ is a factorization system through $\mathcal{S}$ on $\mathcal{C}$, both classes $E_{\mathcal{S}}$ and $M_{\mathcal{S}}$ are closed under composition, their intersection consists of precisely the isomorphisms in $\mathcal{S}$, and, as expected, that $(E_{\mathcal{S}}, M_{\mathcal{S}})$-factorizations of $\mathcal{S}$-small morphisms are unique up to isomorphism.

▶ **Example 3.6.** Instantiating $(\mathcal{C}, \mathcal{S})$ to be $(\mathsf{Set}, \mathsf{Set}_{\mathsf{fin}})$ yields a natural factorization system through $\mathsf{Set}_{\mathsf{fin}}$ on Set (as the restriction of the standard (epi,mono)-factorization system on Set to $\mathsf{Set}_{\mathsf{fin}}$-small morphisms, i.e., the maps of finite image). Over these categories $\mathsf{Set}_{\mathsf{fin}}$-automata are deterministic finite state automata inside the more general category of Set-automata which are deterministic (potentially infinite) automata. The example $(\mathsf{Vec}, \mathsf{Vec}_{\mathsf{fin}})$ was already mentioned. In this case, being $\mathsf{Vec}_{\mathsf{fin}}$-small is equivalent to having finite rank.

Notice that for $(\mathcal{C}, \mathcal{S}) = (\mathsf{Set}, \mathsf{Set}_{\mathsf{fin}})$ or $(\mathsf{Vec}, \mathsf{Vec}_{\mathsf{fin}})$, the factorization systems through the subcategories are obtained simply by restricting the factorization systems on Set, respectively Vec. This is because, in these cases $\mathcal{S}$ is closed under quotients and subobjects in $\mathcal{C}$. The category $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$ used in this paper is closed under quotients, but in general not under subobjects (and this is the important reason for this extension of the standard notion of factorization). This is also a case in which there is a factorization system in the category $\mathcal{C}$, that coincide over $\mathcal{S}$ with factorizing through $\mathcal{S}$, but for which factorizing in $\mathcal{C}$ of an $\mathcal{S}$-small morphism and factorizing it through $\mathcal{S}$ yield different results.

A factorization system on $\mathcal{C}$ lifts naturally to categories of $\mathcal{C}$-valued functors. Automata being very close in definition to such a functor category, factorization systems also lift to them. Lemma 3.7 shows that this is also the case for factorization systems through $\mathcal{S}$, assuming of course that the input and output objects I and F belong to $\mathcal{S}$.

▶ **Lemma 3.7.** *Whenever $(E_{\mathcal{S}}, M_{\mathcal{S}})$ is a factorization system through a category $\mathcal{S}$ then $(E_{\mathsf{Auto}_{\mathcal{S}}(L)}, M_{\mathsf{Auto}_{\mathcal{S}}(L)})$ is a factorization system through $\mathsf{Auto}_{\mathcal{S}}(L)$ for the category $\mathsf{Auto}_{\mathcal{C}}(L)$, where $E_{\mathsf{Auto}_{\mathcal{S}}(L)}$ (resp. $M_{\mathsf{Auto}_{\mathcal{S}}(L)}$) contains these $\mathsf{Auto}_{\mathcal{C}}(L)$-morphisms that belong to $E_{\mathcal{S}}$ (resp. to $M_{\mathcal{S}}$) as $\mathcal{C}$-morphisms.*

3.3 Minimization through a subcategory

In this section, we show how the joint combination of having initial and final automata for a language, as given by Lemma 3.4, together with a factorization system through a subcategory $\mathcal{S}$ yields the existence of a minimal $\mathcal{S}$-automaton for small $\mathcal{C}$-languages.

We make the *following assumptions*: $(E_{\mathcal{S}}, M_{\mathcal{S}})$ is a factorization system through $\mathcal{S}$ on $\mathcal{C}$, and L is a $\mathcal{C}$-language accepted by some $\mathcal{S}$-automaton such that there exist an initial $\mathtt{init}_{\mathcal{C}}(L)$ $\mathcal{C}$-automaton and a final $\mathcal{C}$-automaton $\mathtt{final}_{\mathcal{C}}(L)$ for L.

▶ **Definition 3.8** (minimal automaton)**.** The *minimal $\mathcal{C}$-automaton for L*, denoted $\mathtt{min}_{\mathcal{S}}(L)$, is the[4] $\mathcal{S}$-automaton for L obtained by $(E_{\mathsf{Auto}_{\mathcal{S}}(L)}, M_{\mathsf{Auto}_{\mathcal{S}}(L)})$-factorization of the unique $\mathsf{Auto}_{\mathcal{S}}(L)$-small morphism from $\mathtt{init}_{\mathcal{C}}(L)$ to $\mathtt{final}_{\mathcal{C}}(L)$.

▶ **Theorem 3.9.** *For all $\mathcal{S}$-automata $\mathcal{A}$ for L satisfying the above assumptions, we have*

$$\mathtt{min}_{\mathcal{S}}(L) \cong \mathtt{obs}_{\mathcal{S}}(\mathtt{reach}_{\mathcal{S}}(\mathcal{A})) \cong \mathtt{reach}_{\mathcal{S}}(\mathtt{obs}_{\mathcal{S}}(\mathcal{A}))\,,$$

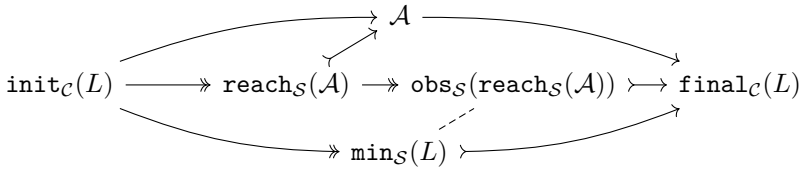
in which

[4] It is unique up to isomorphism according to the diagonal property.

- $\mathtt{reach}_{\mathcal{S}}(\mathcal{A})$ *is the result of applying an* $(E_{\mathsf{Auto}_{\mathcal{S}}(L)}, M_{\mathsf{Auto}_{\mathcal{S}}(L)})$*-factorization to the unique* $\mathsf{Auto}_{\mathcal{S}}(L)$*-morphism from* $\mathtt{init}_{\mathcal{C}}(L)$ *to* $\mathcal{A}$*, and*
- $\mathtt{obs}_{\mathcal{S}}(\mathcal{A})$ *is the result of applying an* $(E_{\mathsf{Auto}_{\mathcal{S}}(L)}, M_{\mathsf{Auto}_{\mathcal{S}}(L)})$*-factorization to the unique* $\mathsf{Auto}_{\mathcal{S}}(L)$*-morphism from* $\mathcal{A}$ *to* $\mathtt{final}_{\mathcal{C}}(L)$*.*

This theorem does not only state the existence of a minimal automaton, it also makes transparent how to make effective its construction: if one possesses both an implementation of $\mathtt{reach}_{\mathcal{S}}$ and $\mathtt{obs}_{\mathcal{S}}$, then their sequencing minimises an input automaton. From the above theorem it immediately follows that $\mathtt{min}_{\mathcal{S}}(L)$ is both an $E_{\mathsf{Auto}_{\mathcal{S}}(L)}$-quotient of a $M_{\mathsf{Auto}_{\mathcal{S}}(L)}$-subobject of $\mathcal{A}$ and a $M_{\mathsf{Auto}_{\mathcal{S}}(L)}$-subobject of an $E_{\mathsf{Auto}_{\mathcal{S}}(L)}$-quotient of $\mathcal{A}$: the minimal automaton divides every other automaton for the language.

Proof idea. The proof is contained in the following commutative diagram, in which $\twoheadrightarrow$ denotes $\mathsf{Auto}_{\mathcal{C}}(L)$-morphisms in $E_{\mathsf{Auto}_{\mathcal{S}}(L)}$, and $\rightarrowtail$ $\mathsf{Auto}_{\mathcal{C}}(L)$-morphisms in $M_{\mathsf{Auto}_{\mathcal{S}}(L)}$:



That $\mathtt{obs}_{\mathcal{S}}(\mathtt{reach}_{\mathcal{S}}(\mathcal{A}))$ is an $(E_{\mathsf{Auto}_{\mathcal{S}}(L)}, M_{\mathsf{Auto}_{\mathcal{S}}(L)})$-factorization of the unique $\mathsf{Auto}_{\mathcal{C}}(L)$-morphism from $\mathtt{init}_{\mathcal{C}}(L)$ to $\mathtt{final}_{\mathcal{C}}(L)$ follows since $E_{\mathsf{Auto}_{\mathcal{S}}(L)}$ is closed under composition. By the unique diagonal property, it is isomorphic to $\mathtt{min}_{\mathcal{S}}(L)$. The case $\mathtt{reach}_{\mathcal{S}}(\mathtt{obs}_{\mathcal{S}}(\mathcal{A}))$ is symmetric. ◀

3.4 A special case of factorization through

So far, the description of factorization and minimization of automata is very generic. Hereafter, the classes $E_{\mathcal{S}}$ and $M_{\mathcal{S}}$ are constructed along a particular principle which we describe now. In the next sections we will instantiate this construction when $\mathcal{S}$ is the subcategory $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$ of glued vector spaces.

In this section we fix an (E, M)-factorization system on $\mathcal{C}$ and a subcategory $\mathcal{S} \hookrightarrow \mathcal{C}$.

▶ **Definition 3.10.** An $\mathcal{S}$*-extremal epimorphism*[5] in $\mathcal{C}$ is an arrow $e\colon X \to S$ in $\mathcal{C}$, with S an object in $\mathcal{S}$, such that if $e = m \circ g$ where m is in M with domain in $\mathcal{S}$, then m is an isomorphism. We set $M_{\mathcal{S}}$ to be the class of arrows in M with domain in $\mathcal{S}$, and $E_{\mathcal{S}}$ to be the class of $\mathcal{S}$-extremal epimorphisms.

▶ **Definition 3.11.** An $M_{\mathcal{S}}$*-subobject in* $\mathcal{S}$ of an object X of $\mathcal{C}$ is an equivalence class up to isomorphism of a morphism $S \to X$ belonging to M, where S is an object of $\mathcal{S}$. The $M_{\mathcal{S}}$-subobject $S \to X$ is called *proper* if it is not an isomorphism.

▶ **Lemma 3.12.** *Assume the following conditions hold:*

1. *all arrows in* M *are monomorphisms in* $\mathcal{C}$*,*
2. $\mathcal{S}$ *is closed under* E*-quotients, i.e., if* $e\colon S \to T$ *is in* E *with* S *in* $\mathcal{S}$*, then* T *is isomorphic to an object of* $\mathcal{S}$*,*
3. *the intersection of a nonempty set of* $M_{\mathcal{S}}$*-subobjects of an object* X *of* $\mathcal{C}$ *exists and is an* $M_{\mathcal{S}}$*-subobject of* X*, and,*

[5] Note that $\mathcal{S}$-extremal epimorphisms need not be epimorphisms in $\mathcal{C}$.

4. *the pullback of an $M_{\mathcal{S}}$-subobject $m\colon S \to T$ of T along a morphism $T' \to T$ in $\mathcal{S}$ is an $M_{\mathcal{S}}$-subobject of T'.*

Then $(E_{\mathcal{S}}, M_{\mathcal{S}})$ is a factorization system through $\mathcal{S}$ on $\mathcal{C}$.

The next lemma ensures that condition 3 of Lemma 3.12 can be replaced with the weaker version involving only binary intersections of $M_{\mathcal{S}}$-subobjects, provided that any infinite descending chain of $M_{\mathcal{S}}$-subobjects eventually stabilises (of course, up to isomorphism).

▶ **Lemma 3.13.** *Assume that there are no infinite descending chains of proper $M_{\mathcal{S}}$-subobjects*

$$X \longleftarrow\!\!\prec S_1 \longleftarrow\!\!\prec S_2 \longleftarrow\!\!\prec \dots$$

and furthermore that the intersection of any two $M_{\mathcal{S}}$-subobjects of an object X of $\mathcal{C}$ exists and is an $M_{\mathcal{S}}$-subobject of X, then condition 3 in the hypothesis of Lemma 3.12 holds.

The proof simply uses finite partial intersections in order to create a strictly descending chain of $M_{\mathcal{S}}$-subobjects. By assumption, this construction has to stop, and the last element of the sequence happens to be the intersection of the entire family.[6]

4 Gluing of categories

We turn now to the central construction of this paper: given a category $\mathcal{C}$ and a subcategory $\mathcal{S}$, we construct a category $\mathtt{Glue}(\mathcal{C})$ of "gluings of objects in $\mathcal{C}$" that has both $\mathcal{C}$ and $\mathtt{Glue}_{\mathsf{fin}}(\mathcal{S})$ – the category of "finite gluings of objects in $\mathcal{S}$" – as subcategories. Under proper assumptions on $\mathcal{C}$ and $\mathcal{S}$, the resulting pair $(\mathtt{Glue}(\mathcal{C}), \mathtt{Glue}_{\mathsf{fin}}(\mathcal{S}))$ satisfies the assumption required for constructing minimisable automata for $\mathtt{Glue}(\mathcal{C})$-languages. Taking $\mathcal{C} = \mathsf{Vec}$ and $\mathcal{S} = \mathsf{Vec}_{\mathsf{fin}}$ we obtain the construction informally described in Section 2.

Throughout this section we assume that $\mathcal{C}$ is equipped with a (E, M)-factorization system consisting of strong epimorphisms and monomorphisms.

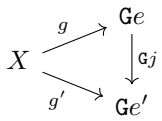
4.1 The free gluing of a category

When $\mathcal{C}$ is small, it is well known that the Yoneda embedding of $\mathcal{C}$ into the category of presheaves over $\mathcal{C}$ is a free completion of $\mathcal{C}$ under colimits of small diagrams. For a possibly large category, one has to consider instead the category of small presheaves, i.e. small colimits of representable ones, see for example [9]. For our purposes, we found more illuminating and direct to use a syntactic way of describing the colimit completion of a category.

The category of diagrams. Assume $\mathcal{C}$ is a locally small category. The *free colimit completion of* $\mathcal{C}$ is the category $\mathsf{Diag}(\mathcal{C})$ whose objects are *diagrams* $\mathtt{F}\colon \mathcal{D} \to \mathcal{C}$ and morphisms between two diagrams $\mathtt{F}\colon \mathcal{D} \to \mathcal{C}$ and $\mathtt{G}\colon \mathcal{E} \to \mathcal{C}$ will be given in Definition 4.1.

To this end we define an equivalence relation on arrows from an arbitrary object X of $\mathcal{C}$ to the objects in the image of $\mathtt{G}$. Assume e, e' are objects in $\mathcal{E}$. We consider the least equivalence relation $\sim_{\mathtt{G}}$ which contains all pairs (g, g'), where $g\colon X \to \mathtt{G}e$, $g'\colon X \to \mathtt{G}e'$ are such that there exists $j\colon e \to e'$ a map in $\mathcal{E}$ with $\mathtt{G}j \circ g = g'$, i.e., the diagram below commutes.

[6] The attentive reader will have recognised in this argument part of the reason why every subset of a finite-dimensional vector space admits a minimal cover as a finite union of subspaces.



We denote by $\widehat{\mathtt{G}}(X)$ the equivalence classes of the relation $\sim_{\mathtt{G}}$.

▶ **Definition 4.1.** A *morphism between diagrams* $\mathtt{F}\colon \mathcal{D} \to \mathcal{C}$ and $\mathtt{G}\colon \mathcal{E} \to \mathcal{C}$ is a map f which associates to each object d in $\mathcal{D}$ an equivalence class $f(d) \in \widehat{\mathtt{G}}(\mathtt{F}d)$, such that whenever $u\colon d \to d'$ is a morphism in $\mathcal{D}$ and $g\colon \mathtt{F}d' \to \mathtt{G}e$ is in the equivalence class $f(d')$, then $g \circ \mathtt{F}u$ is in the equivalence class $f(d)$.

The subcategory of gluings. We are now ready to define the category $\mathtt{Glue}(\mathcal{C})$, which is a restriction of $\mathsf{Diag}(\mathcal{C})$ to M-diagrams (see below), that is, to diagrams that intuitively 'do not quotient'. Recall that $\mathcal{C}$ has a factorization system (E, M) in which E are the strong epimorphisms and M are the monomorphisms.

▶ **Definition 4.2** (*glued category*)**.** An *M-cocone* over a diagram $\mathtt{F}\colon \mathcal{D} \to \mathcal{C}$ is a cocone $(u_d\colon \mathtt{F}d \to X)_{d\in\mathcal{D}}$ such that all the structural components of the cocone u_d are in M. An *M-diagram* is a diagram that has an M-cocone.

The *glued category* $\mathtt{Glue}(\mathcal{C})$ is the subcategory of $\mathsf{Diag}(\mathcal{C})$ over the M-diagrams $\mathtt{F}\colon \mathcal{D} \to \mathcal{C}$. Let $\mathtt{Glue}_{\mathsf{fin}}(\mathcal{C})$ the subcategory of $\mathtt{Glue}(\mathcal{C})$ that has as objects the finite diagrams of $\mathtt{Glue}(\mathcal{C})$.

Notice that, if $\mathtt{F}$ is such a diagram, then we can show that for each morphism $v\colon d \to d'$ in $\mathcal{D}$, we have that $\mathtt{F}u\colon \mathtt{F}d \to \mathtt{F}d'$ is in M (however this is not a characterisation). Also, if there exists a universal cocone for an M-diagram, then this cocone is in particular an M-cocone.

▶ **Lemma 4.3.** *If $\mathcal{C}$ is cocomplete, then $\mathtt{Glue}(\mathcal{C})$ is a full reflective subcategory of $\mathsf{Diag}(\mathcal{C})$, and hence $\mathtt{Glue}(\mathcal{C})$ is a cocomplete category. If $\mathcal{C}$ is furthermore complete, then so is $\mathtt{Glue}(\mathcal{C})$.*

In the automata theoretic application we have in mind, we use this category in order to construct the initial and final automata for a language.

4.2 A factorization system through finite gluings

The category of most interest for us is the full subcategory $\mathtt{Glue}_{\mathsf{fin}}(\mathcal{S})$ of $\mathtt{Glue}(\mathcal{C})$ which consists of the finite M-diagrams over $\mathcal{S}$. In this section we construct in particular, under suitable assumptions, a factorization system through $\mathtt{Glue}_{\mathsf{fin}}(\mathcal{S})$ on $\mathtt{Glue}(\mathcal{C})$, making use of Lemma 3.12. For $\mathcal{S} = \mathsf{Vec}_{\mathsf{fin}}$, this is the category of 'finite gluings of finite vector spaces' that we longly introduced in Section 2.

We define the following classes of morphisms in $\mathtt{Glue}(\mathcal{C})$.

- $\mathit{Epi}_{\mathtt{Glue}(\mathcal{C})}$ consists of the morphisms $f\colon \mathtt{F} \to \mathtt{G}$, where $\mathtt{F}\colon \mathcal{D} \to \mathcal{C}$ and $\mathtt{F}\colon \mathcal{E} \to \mathcal{C}$, such that for all e in $\mathcal{E}$ there exists a representative $f_d\colon \mathtt{F}d \to \mathtt{G}e'$ in the equivalence class $f(d)$ and a morphism $u\colon \mathtt{G}e \to \mathtt{G}e'$, so that $u \sim_{\mathtt{G}} id_{\mathtt{G}e}$ and u factors through the image of f_d.
- $\mathit{Mono}_{\mathtt{Glue}(\mathcal{C})}$ consists of morphisms $f\colon \mathtt{F} \to \mathtt{G}$ such that for all morphisms $u\colon X \to \mathtt{F}d$ and $v\colon X \to \mathtt{F}d'$ such that $f_d \circ u \sim_{\mathtt{G}} f_{d'} \circ v$ (for f_d and $f_{d'}$ in the equivalence classes $f(d)$, respectively $f(d')$), we have that $u \sim_{\mathtt{F}} v$.

One can easily verify that the arrows in $\mathit{Mono}_{\mathtt{Glue}(\mathcal{C})}$ are exactly the monomorphisms in $\mathtt{Glue}(\mathcal{C})$.[7] The next lemma establishes that under mild conditions on $\mathcal{C}$ we have a (strong epi, mono) factorization system on $\mathtt{Glue}(\mathcal{C})$.

[7] As a side remark, we should mention that these classes of arrows correspond precisely to the natural transformations between the induced presheaves that are pointwise injective.

▶ **Lemma 4.4.** *Assume $\mathcal{C}$ has intersections. Then $(Epi_{\mathtt{Glue}(\mathcal{C})}, Mono_{\mathtt{Glue}(\mathcal{C})})$ is a (strong epi, mono) factorization system on $\mathtt{Glue}(\mathcal{C})$.*

In what follows we say that a subcategory $\mathcal{S}$ of $\mathcal{C}$ is *well-behaved* if it satisfies the hypothesis of Lemmas 3.12 and 3.13 with respect to the (strong epi, mono) factorization system on $\mathcal{C}$. (In fact condition 3 of Lemma 3.12 can be replaced by its binary version.)

▶ **Theorem 4.5.** *Assume $\mathcal{C}$ has intersections and pullbacks. If the subcategory $\mathcal{S}$ of $\mathcal{C}$ is well-behaved, then $\mathtt{Glue}_{\mathsf{fin}}(\mathcal{S})$ is a well-behaved subcategory of $\mathtt{Glue}(\mathcal{C})$.*

Some ideas about the proof. This result is an application of Lemmas 3.12 and 3.13. The central combinatorial aspect of this statement is that there exists no infinite strictly descending chains of $Mono_{\mathtt{Glue}(\mathcal{C})}$-subobjects in $\mathtt{Glue}_{\mathsf{fin}}(\mathcal{S})$. For the sake of contradiction, let us consider a descending sequence of diagrams from $\mathtt{Glue}_{\mathsf{fin}}(\mathcal{S})$:

$$\mathtt{F}_0 \xleftarrow{f_1} \mathtt{F}_1 \xleftarrow{f_2} \mathtt{F}_2 \xleftarrow{f_3} \dots$$

We have to prove that it is ultimately constant (up to isomorphism). Let the diagrams be $\mathtt{F}_i : \mathcal{D}_i \to \mathcal{S}$ for all i. The first step is to consider an *aggregation of mono-diagrams*, that is, we construct a big diagram that aggregates all the $\mathtt{F}_i$'s. At the level of objects this diagram contains the disjoint unions of the $\mathcal{D}_i$'s. We call the objects originating from $\mathcal{D}_i$ of *rank* i. Secondly, we prove a *global homogeneity* property of $\mathtt{F}$: given two arrows $g\colon X \to \mathtt{F}d$, $g'\colon X \to \mathtt{F}d'$ with d, d' at the same rank i, then $g \sim_{\mathtt{F}} g'$ if and only if $g \sim_{\mathtt{F}_i} g'$. Finally we prove the existence of an isomorphism g_j from $\mathtt{F}_{j-1}$ to $\mathtt{F}_j$ for some j by analysing the structure of $\mathtt{F}$ and using König's lemma. ◀

We come back to the leading example of $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$-automata. Applying Theorem 4.5 for $(\mathcal{C}, \mathcal{S}) = (\mathsf{Vec}, \mathsf{Vec}_{\mathsf{fin}})$ we obtain a factorization system through $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$ on $\mathtt{Glue}(\mathsf{Vec})$. Using Lemma 4.3 and Theorem 3.9 we derive that hybrid set-vector automata are minimisable.

▶ **Corollary 4.6.** *For any $\mathtt{Glue}(\mathsf{Vec})$-language accepted by some $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$-automaton there exists a minimal $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$-automaton. In particular, any Vec-language accepted by a $\mathsf{Vec}_{\mathsf{fin}}$-automaton has a minimal $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$-automaton.*

For the language described in Section 2, and for which the minimal vector automaton has a two-dimensional state space, we obtain a minimal $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$-automaton obtained by glueing two one-dimensional spaces at 0. Formally, this is an M-diagram $\mathtt{F}\colon \mathcal{D} \to \mathsf{Vec}_{\mathsf{fin}}$ where $\mathcal{D}$ is a three object poset $\{\bot, 0, 1\}$ with $\bot \leq 0$ and $\bot \leq 1$. The functor $\mathtt{F}$ maps $0, 1$ to one-dimensional spaces, $\bot$ to the zero-dimensional space and the morphisms $\bot \leq 0$ and $\bot \leq 1$ to its inclusions in the respective one-dimensional spaces.

For another example, consider the language which to a word $u \in a^*$ associates the value $\cos(\alpha|u|)$ for some α which is not a rational multiple of π, and whose minimal vector space automaton has a two-dimensional state space. If we used the factorization in $\mathtt{Glue}(\mathsf{Vec})$ we would obtain a glueing of infinitely many one-dimensional subspaces (obtained by rotations with angle α). Thus, it is crucial for our setting to use the *factorization system through* $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$. In this case, the minimal $\mathtt{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$-automaton also has just a two-dimensional vector space of states.

5 Conclusion

We have introduced a new way to construct automata which, thanks to category-theoretic insights, admits minimal automata 'by design'. The introductory example of hybrid set-vector

automata is a convincing instance of this approach, which has both algorithmic merits (in succinctness of the encoding of the state space) and theoretical merits (in that there exists minimal automata). The closest work to our knowledge is the work of Lombardy and Sakarovitch [12] which studies the sequentialisability of weighted automata; in the framework of this paper, this is answering the question whether a vector space automaton is equivalent to a hybrid set-vector automaton for which the state space consists of dimension 1 vector spaces only, glued at 0 (the problem remains open).

At the categorical level, we should say a few words regarding our design choices. First why not use more familiar co-completions such as the Ind-completion of the free co-completion? The answer is that if we did so, we would not obtain the desired behaviour when we restrict our attention to the 'finite' automata. For example finite filtered colimits are not very interesting, while the freely added finite colimits of vector spaces are not closed under quotients in the free co-completion, thus the work in the previous sections cannot be applied.

Another question one may ask is why we haven't used coalgebras, as in [2] or in the work of [1] on well-pointed coalgebras. First, the factorization through a subcategory, which plays a crucial role in our work, is not developed in that setting. Secondly, we believe that the functorial approach to automata, which neatly combines the dual narrative of automata seen as both algebras and coalgebras is worth saying. As we show in [8], we can employ this framework for minimizing subsequential transducers à la Choffrut [7] (by interpreting them as automata in a Kleisli category). This is also an example in which the conditions in Lemma 3.4 are not necessary. We believe, that at least in that situation the functorial approach works slightly smoother than the coalgebraic one [11]. Also, by changing the input category, we can further extend this work to capture tree automata or algebras (for instance monoids).

In the particular model of hybrid set-vector automata the problem of effectiveness remains: we have proved the existence of a minimal automaton for a language, but obtaining the reachable configurations in an effective way is the subject of ongoing work.

References

1 Jirí Adámek, Stefan Milius, Lawrence S. Moss, and Lurdes Sousa. Well-pointed coalgebras. *Logical Methods in Computer Science*, 9(3), 2013.

2 Jiří Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius, and Alexandra Silva. A coalgebraic perspective on minimization and determinization. In *Proceedings of the 15th International Conference on Foundations of Software Science and Computational Structures*, FOSSACS'12, pages 58–73, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-28729-9_4.

3 Michael A. Arbib and Ernest G. Manes. Adjoint machines, state-behavior machines, and duality. *Journal of Pure and Applied Algebra*, 6(3):313 – 344, 1975. doi:10.1016/0022-4049(75)90028-6.

4 Nick Bezhanishvili, Clemens Kupke, and Prakash Panangaden. *Minimization via Duality*, pages 191–205. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-32621-9_14.

5 Filippo Bonchi, Marcello Bonsangue, Michele Boreale, Jan Rutten, and Alexandra Silva. A coalgebraic perspective on linear weighted automata. *Inf. Comput.*, 211:77–105, February 2012. doi:10.1016/j.ic.2011.12.002.

6 Filippo Bonchi, Marcello M. Bonsangue, Helle Hvid Hansen, Prakash Panangaden, Jan J. M. M. Rutten, and Alexandra Silva. Algebra-coalgebra duality in Brzozowski's minimization algorithm. *ACM Trans. Comput. Log.*, 15(1):3:1–3:29, 2014.

7 Christian Choffrut. A generalization of Ginsburg and Rose's characterization of G-S-M mappings. In *ICALP*, volume 71 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 1979.

8 Thomas Colcombet and Daniela Petrişan. Automata minimization: a functorial approach. *CALCO*, 72:8:1–8:15, 2017.

9 Brian J. Day and Stephen Lack. Limits of small functors. *Journal of Pure and Applied Algebra*, 210(3):651 – 663, 2007. doi:10.1016/j.jpaa.2006.10.019.

10 J. A. Goguen. Minimal realization of machines in closed categories. *Bull. Amer. Math. Soc.*, 78(5):777–783, 09 1972. URL: http://projecteuclid.org/euclid.bams/1183533991.

11 Helle Hvid Hansen. Subsequential transducers: a coalgebraic perspective. *Inf. Comput.*, 208(12):1368–1397, 2010.

12 Sylvain Lombardy and Jacques Sakarovitch. Sequential? *Theor. Comput. Sci.*, 356(1-2):224–244, 2006. doi:10.1016/j.tcs.2006.01.028.

13 M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2):245 – 270, 1961. doi:10.1016/S0019-9958(61)80020-X.

The Equivalence, Unambiguity and Sequentiality Problems of Finitely Ambiguous Max-Plus Tree Automata are Decidable*

Erik Paul

Institute of Computer Science, Leipzig University, Leipzig, Germany
epaul@informatik.uni-leipzig.de

Abstract

We show that the equivalence, unambiguity and sequentiality problems are decidable for finitely ambiguous max-plus tree automata.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Tree Automata, Max-Plus Automata, Equivalence, Unambiguity, Sequentiality, Decidability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.53

1 Introduction

A max-plus automaton is a finite automaton with transition weights in the real numbers. To each word, it assigns the maximum weight of all accepting paths on the word, where the weight of a path is the sum of the path's transition weights. Max-plus automata and their min-plus counterparts are weighted automata [19, 18, 13, 2, 4] over the max-plus or min-plus semiring. Under varying names, max-plus and min-plus automata have been studied and employed many times in the literature. They can be used to determine the star height of a language [7], to decide the finite power property [20, 21] and to model certain timed discrete event systems [5, 6]. Additionally, they appear in the context of natural language processing [14].

For practical applications, the decidable properties of an automaton model are usually of great interest. Typical problems considered include the emptiness, universality, inclusion, equivalence, sequentiality and unambiguity problems. We consider the last three of these problems for *finitely ambiguous* automata, which are automata in which the number of accepting paths for every word is bounded by a global constant. If there is at most one accepting path for every word, the automaton is called *unambiguous*. It is called *deterministic* or *sequential* if for each pair of a state and an input symbol, there is at most one valid transition into a next state. It is known [11] that finitely ambiguous max-plus automata are strictly more expressive than unambiguous max-plus automata, which in turn are strictly more expressive than deterministic max-plus automata.

Let us quickly recall the considered problems and the related results. The equivalence problem asks whether two automata are *equivalent*, which is the case if the weights assigned by them coincide on all words. In general, the equivalence problem is undecidable [12] for max-plus automata, but for finitely ambiguous max-plus automata it becomes decidable

* This work was supported by Deutsche Forschungsgemeinschaft (DFG), Graduiertenkolleg 1763 (QuantLA).

[22, 9]. The sequentiality problem asks whether for a given automaton, there exists an equivalent deterministic automaton. This was shown to be decidable by Mohri [14] for unambiguous max-plus automata. Finally, the unambiguity problem asks whether for a given automaton, there exists an equivalent unambiguous automaton. This problem is known to be decidable for finitely ambiguous and even *polynomially ambiguous* max-plus automata [11, 10]. In conjunction with Mohri's results, it follows that the sequentiality problem is decidable for these classes of automata as well.

In this paper, we show that these three problems are decidable for finitely ambiguous max-plus tree automata, which are max-plus automata that operate on trees instead of words. In the form of *probabilistic context-free grammars*, max-plus tree automata are commonly employed in natural language processing [17]. Our approach to the decidability of the equivalence problem uses ideas from [9]. We use a similar induction argument and also reduce the equivalence problem to the same decidable problem, namely the decidability of the existence of an integer solution for a system of linear inequalities [15]. On words, the proof relies on the decomposition of words into subwords of bounded length, of which one is removed in the induction step. This argument cannot be applied to trees as easily. A tree can be decomposed into *contexts* of bounded height, but this requires contexts with multiple variables. Removing such a context does usually not yield a tree. Consequently, our induction is much more involved. We also point out and correct an important oversight in the main theorem of [9].

The decidability of the unambiguity problem employs ideas from [11]. Here, we show how the *dominance property* can be generalized to max-plus tree automata. To show the decidability of the sequentiality problem, we first combine results from [3] and [14] to show the decidability of this problem for unambiguous max-plus tree automata, and then combine this result with the decidability of the unambiguity problem.

Our solution of the equivalence problem can be applied to weighted logics. In [16], a fragment of a weighted logic is shown to have the same expressive power as finitely ambiguous weighted tree automata. Over the max-plus semiring, equivalence is decidable for formulas of this fragment due to our results.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$. By $\mathbb{N}^*$ we denote the set of all finite words over $\mathbb{N}$. The empty word is denoted by ε, and the length of a word $w \in \mathbb{N}^*$ by $|w|$. The set $\mathbb{N}^*$ is partially ordered by the prefix relation $\leq_p$ and totally ordered with respect to the lexicographic ordering $\leq_l$. A *ranked alphabet* is a pair $(\Gamma, \mathrm{rk}_\Gamma)$, often abbreviated by Γ, where Γ is a finite set and $\mathrm{rk}_\Gamma \colon \Gamma \to \mathbb{N}$. For every $m \geq 0$ we define $\Gamma^{(m)} = \mathrm{rk}_\Gamma^{-1}(m)$ as the set of all symbols of rank m. The rank $\mathrm{rk}(\Gamma)$ of Γ is defined as $\max\{\mathrm{rk}_\Gamma(a) \mid a \in \Gamma\}$.

The set of (*finite, labeled and ordered*) Γ-*trees*, denoted by T_Γ, is the set of all pairs $t = (\mathrm{pos}(t), \mathrm{label}_t)$, where $\mathrm{pos}(t) \subset \mathbb{N}^*$ is a finite non-empty prefix-closed set, $\mathrm{label}_t \colon \mathrm{pos}(t) \to \Gamma$ is a mapping and for every $w \in \mathrm{pos}(t)$ we have $wi \in \mathrm{pos}(t)$ iff $1 \leq i \leq \mathrm{rk}_\Gamma(\mathrm{label}_t(w))$. We write $t(w)$ for $\mathrm{label}_t(w)$. We also refer to the elements of $\mathrm{pos}(t)$ as *nodes*, to ε as the *root* of t and to prefix-maximal nodes as *leaves*.

Now let $s, t \in T_\Gamma$ and $w \in \mathrm{pos}(t)$. The *subtree of* t *at* w, denoted by $t|_w$, is a Γ-tree defined as follows. We let $\mathrm{pos}(t|_w) = \{v \in \mathbb{N}^* \mid wv \in \mathrm{pos}(t)\}$ and for $v \in \mathrm{pos}(t|_w)$, $\mathrm{label}_{t|_w}(v) = t(wv)$. The *substitution of* s *into* w *of* t, denoted by $t\langle s \to w\rangle$, is a Γ-tree defined as follows. We let $\mathrm{pos}(t\langle s \to w\rangle) = \{v \in \mathrm{pos}(t) \mid w \not\leq_p v\} \cup \{wv \mid v \in \mathrm{pos}(s)\}$. For $u \in \mathrm{pos}(t\langle s \to w\rangle)$, we let $\mathrm{label}_{t\langle s\to w\rangle}(u) = s(v)$ if $u = wv$, and otherwise $\mathrm{label}_{t\langle s\to w\rangle}(u) = t(u)$.

For $a \in \Gamma^{(m)}$ and trees $t_1, \dots, t_m \in T_\Gamma$, we also write $a(t_1, \dots, t_m)$ to denote the tree t with $\mathrm{pos}(t) = \{\varepsilon\} \cup \{iw \mid i \in \{1, \dots, m\}, w \in \mathrm{pos}(t_i)\}$, $\mathrm{label}_t(\varepsilon) = a$ and $\mathrm{label}_t(iw) = t_i(w)$.

A *commutative semiring* is a tuple $(K, \oplus, \odot, \mathbb{0}, \mathbb{1})$, abbreviated by K, with operations sum $\oplus$ and product $\odot$ and constants $\mathbb{0}$ and $\mathbb{1}$ such that $(K, \oplus, \mathbb{0})$ and $(K, \odot, \mathbb{1})$ are commutative monoids, multiplication distributes over addition, and $k \odot \mathbb{0} = \mathbb{0} \odot k = \mathbb{0}$ for every $k \in K$. In this paper, we only consider the following two semirings.

- The *boolean semiring* $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$ with disjunction $\vee$ and conjunction $\wedge$.
- The *max-plus semiring* $\mathbb{R}_{\max} = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$ where the sum and the product operations are max and $+$, respectively, extended to $\mathbb{R} \cup \{-\infty\}$ in the usual way.

A *(formal) tree series* is a mapping $S \colon T_\Gamma \to K$. The set of all tree series (over Γ and K) is denoted by $K\langle\langle T_\Gamma \rangle\rangle$. For two tree series $S, T \in K\langle\langle T_\Gamma \rangle\rangle$, the sum $S \oplus T$ and the *Hadamard product* $S \odot T$ are defined pointwise.

Let $(K, \oplus, \odot, \mathbb{0}, \mathbb{1})$ be a commutative semiring. A *weighted bottom-up finite state tree automaton (short: WTA) over K and Γ* is a tuple $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ where Q is a finite set (of states), Γ is a ranked alphabet (of input symbols), $\mu \colon \bigcup_{m=0}^{\mathrm{rk}(\Gamma)} Q^m \times \Gamma^{(m)} \times Q \to K$ (the weight function) and $\nu \colon Q \to K$ (the function of final weights). We set $\Delta_\mathcal{A} = \bigcup_{m=0}^{\mathrm{rk}(\Gamma)} Q^m \times \Gamma^{(m)} \times Q$. A tuple $(\vec{p}, a, q) \in \Delta_\mathcal{A}$ is called a *transition* and $(\vec{p}, a, q)$ is called *valid* if $\mu(\vec{p}, a, q) \neq \mathbb{0}$. A state $q \in Q$ is called *final* if $\nu(q) \neq \mathbb{0}$.

We call a WTA over the max-plus semiring a max-plus-WTA and a WTA over the boolean semiring a *finite tree automaton* (FTA). A WTA $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ over $\mathbb{B}$ is also written as a tuple $\mathcal{A}' = (Q, \Gamma, \delta, F)$ where $\delta = \{d \in \Delta_\mathcal{A} \mid \mu(d) = 1\}$ and $F = \{q \in Q \mid \nu(q) = 1\}$.

For $t \in T_\Gamma$, a mapping $r \colon \mathrm{pos}(t) \to Q$ is called a *quasi-run of $\mathcal{A}$ on t*. For a quasi-run r on t and $w \in \mathrm{pos}(t)$ with $t(w) = a \in \Gamma^{(m)}$, the tuple $\mathbb{t}(t, r, w) = (r(w1), \dots, r(wm), a, r(w))$ is called the *transition at w*. The quasi-run r is called a *(valid) run* if for every $w \in \mathrm{pos}(t)$ the transition $\mathbb{t}(t, r, w)$ is valid with respect to $\mathcal{A}$. We call a run r *accepting* if $r(\varepsilon)$ is final. By $\mathrm{Run}_\mathcal{A}(t)$ and $\mathrm{Acc}_\mathcal{A}(t)$ we denote the sets of all runs and all accepting runs of $\mathcal{A}$ on t, respectively. For $r \in \mathrm{Run}_\mathcal{A}(t)$ the *weight of r* is defined by $\mathrm{wt}_\mathcal{A}(t, r) = \bigodot_{w \in \mathrm{pos}(t)} \mu(\mathbb{t}(t, r, w))$. The *tree series accepted by $\mathcal{A}$*, denoted by $[\![\mathcal{A}]\!] \in K\langle\langle T_\Gamma \rangle\rangle$, is the tree series defined for every $t \in T_\Gamma$ by $[\![\mathcal{A}]\!](t) = \bigoplus_{r \in \mathrm{Acc}_\mathcal{A}(t)} \mathrm{wt}_\mathcal{A}(t, r) \odot \nu(r(\varepsilon))$ where the sum over the empty set is $\mathbb{0}$ by convention. The *support* of $\mathcal{A}$ is the set $\mathrm{supp}(\mathcal{A}) = \{t \in T_\Gamma \mid [\![\mathcal{A}]\!](t) \neq \mathbb{0}\}$.

The support of an FTA $\mathcal{A}$ is also called the *language accepted by $\mathcal{A}$* and denoted by $\mathcal{L}(\mathcal{A})$. A subset $L \subseteq T_\Gamma$ is called *recognizable* if there exists an FTA $\mathcal{A}$ with $L = \mathcal{L}(\mathcal{A})$.

A WTA $\mathcal{A}$ is called *deterministic* if for every $m \geq 0$, $a \in \Gamma^{(m)}$ and $\vec{p} \in Q^m$ there exists at most one $q \in Q$ with $\mu(\vec{p}, a, q) \neq \mathbb{0}$. We call $\mathcal{A}$ *finitely ambiguous* or *M-ambiguous* if $|\mathrm{Acc}_\mathcal{A}(t)| \leq M$ for some $M \geq 1$ and every $t \in T_\Gamma$. A 1-ambiguous WTA is also called *unambiguous*. We recall that for every recognizable language $L \subseteq T_\Gamma$, there exists a deterministic FTA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = L$.

An automaton $\mathcal{A}$ is called *trim* if (i) for every $q \in Q$ there exist $t \in T_\Gamma$, $r \in \mathrm{Acc}_\mathcal{A}(t)$ and $w \in \mathrm{pos}(t)$ such that $q = r(w)$ and (ii) for every valid $d \in \Delta_\mathcal{A}$ there exist $t \in T_\Gamma$, $r \in \mathrm{Acc}_\mathcal{A}(t)$ and $w \in \mathrm{pos}(t)$ such that $d = \mathbb{t}(t, r, w)$. The *trim part of $\mathcal{A}$* is the automaton obtained by removing all states $q \in Q$ which do not satisfy (i) and setting $\mu(d) = \mathbb{0}$ for all valid $d \in \Delta_\mathcal{A}$ which do not satisfy (ii). This process obviously has no influence on $[\![\mathcal{A}]\!]$.

3 The Equivalence Problem

For two max-plus-WTA $\mathcal{A}_1$ and $\mathcal{A}_2$ over an alphabet Γ, we say that $\mathcal{A}_1$ *dominates* $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \geq \mathcal{A}_2$, if for all trees $t \in T_\Gamma$ we have $[\![\mathcal{A}_1]\!](t) \geq [\![\mathcal{A}_2]\!](t)$. We say that $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent, denoted by $\mathcal{A}_1 = \mathcal{A}_2$, if for all $t \in T_\Gamma$ we have $[\![\mathcal{A}_1]\!](t) = [\![\mathcal{A}_2]\!](t)$.

The equivalence problem for max-plus (tree) automata asks whether for two given max-plus (tree) automata $\mathcal{A}_1$ and $\mathcal{A}_2$, it holds that $\mathcal{A}_1 = \mathcal{A}_2$. For words, this problem was shown to be undecidable in general [12], but it is decidable if both automata are finitely ambiguous [9]. In this section, we prove that the equivalence problem is decidable for finitely ambiguous max-plus-WTA. This section is based on ideas from [9].

▶ **Theorem 1.** *The equivalence problem for finitely ambiguous weighted tree automata over the max-plus semiring is decidable.*

In fact, we will show that if $\mathcal{A}_1$ is a finitely ambiguous max-plus-WTA and $\mathcal{A}_2$ any max-plus-WTA, then it is decidable whether $\mathcal{A}_1$ dominates $\mathcal{A}_2$.

▶ **Theorem 2.** *Let $\mathcal{A}_1$ be a finitely ambiguous max-plus-WTA and $\mathcal{A}_2$ any max-plus-WTA. It is decidable whether or not $\mathcal{A}_1 \geq \mathcal{A}_2$.*

If both automata in Theorem 2 are finitely ambiguous, we can reverse their roles. Consequently, Theorem 1 is a corollary of Theorem 2. The remainder of this section is dedicated to the proof of Theorem 2.

As a first step, we show in the following lemma that every finitely ambiguous max-plus-WTA $\mathcal{A}$ can be "normalized" such that all trees, which have an accepting run in $\mathcal{A}$, have the same number of accepting runs.

▶ **Lemma 3.** *Let $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ be an M-ambiguous max-plus-WTA. Then there exists a finitely ambiguous max-plus-WTA $\mathcal{A}'$ with $\mathcal{A} = \mathcal{A}'$ and $|\mathrm{Acc}_{\mathcal{A}'}(t)| \in \{0, M\}$ for all $t \in T_\Gamma$.*

For the rest of this section, fix an M-ambiguous max-plus-WTA $\mathcal{A}_1$ and a max-plus-WTA $\mathcal{A}_2$. By Lemma 3, we can assume that for all $t \in T_\Gamma$ we have $|\mathrm{Acc}_{\mathcal{A}_1}(t)| \in \{0, M\}$. Note that $\mathcal{A}_1 \geq \mathcal{A}_2$ can only hold if $\mathrm{supp}(\mathcal{A}_2) \subseteq \mathrm{supp}(\mathcal{A}_1)$, which is decidable since the supports of $\mathcal{A}_1$ and $\mathcal{A}_2$ are recognizable languages. Therefore, in the forthcoming considerations we will always assume that $\mathrm{supp}(\mathcal{A}_2) \subseteq \mathrm{supp}(\mathcal{A}_1)$ holds. We write $\mathcal{A}_i = (Q_i, \Gamma, \mu_i, \nu_i)$ for $i = 1, 2$.

For any tree in $\mathrm{supp}(\mathcal{A}_2)$, there are exactly M accepting runs of $\mathcal{A}_1$ on this tree. We want to apply pumping type arguments to all of these runs and a given accepting run of $\mathcal{A}_2$ simultaneously. For this, we encode the runs of $\mathcal{A}_1$ and the given run of $\mathcal{A}_2$ directly into the tree. Moreover, we want to decompose these trees, with all runs encoded, into smaller parts. Formally, such a decomposition will be a tree of trees. To mark where these smaller trees connect to each other, we use the new label $\diamond$.

▶ **Definition 4.** For a set X and an alphabet Σ, we define for $(a, x) \in \Sigma \times X$ the rank $\mathrm{rk}_{\Sigma \times X}(a, x) = \mathrm{rk}_\Sigma(a)$. We let $\Gamma_\diamond = (\Gamma \cup \{\diamond\}, \mathrm{rk}_\Gamma \cup \{\diamond \mapsto 0\})$, where $\diamond$ is a new symbol. Let $\mathfrak{Q} = Q_1^M \times Q_2$ and let $\pi_\mathfrak{Q}$, π_Γ and $\pi_{\Gamma_\diamond}$ be the projections of $\Gamma \times \mathfrak{Q}$ and $\Gamma_\diamond \times \mathfrak{Q}$ onto $\mathfrak{Q}$, Γ and $\Gamma_\diamond$, respectively.

For a tree $t \in T_{\Gamma_\diamond \times \mathfrak{Q}}$, we define $\mathrm{label}_t^{\Gamma_\diamond} = \pi_{\Gamma_\diamond} \circ \mathrm{label}_t$ and $\mathrm{label}_t^i = \pi_i \circ \pi_\mathfrak{Q} \circ \mathrm{label}_t$ where π_i is the i-th projection on $\mathfrak{Q}$. For $i \in \{1, \ldots, M+1\}$, we define

$$\mathrm{wt}_i(t) = \begin{cases} \sum\limits_{\substack{w \in \mathrm{pos}(t) \\ \mathrm{label}_t^{\Gamma_\diamond}(w) \neq \diamond}} \mu_1(\mathbb{t}((\mathrm{pos}(t), \mathrm{label}_t^{\Gamma_\diamond}), \mathrm{label}_t^i, w)) & \text{if } 1 \leq i \leq M \\ \sum\limits_{\substack{w \in \mathrm{pos}(t) \\ \mathrm{label}_t^{\Gamma_\diamond}(w) \neq \diamond}} \mu_2(\mathbb{t}((\mathrm{pos}(t), \mathrm{label}_t^{\Gamma_\diamond}), \mathrm{label}_t^i, w)) & \text{if } i = M+1. \end{cases}$$

A tree $t \in T_{\Gamma \times \mathfrak{Q}}$ is called *accepting* if $\mathrm{label}_t^1, \ldots, \mathrm{label}_t^M$ are pairwise distinct accepting runs of $\mathcal{A}_1$ on $(\mathrm{pos}(t), \mathrm{label}_t^{\Gamma_\diamond})$ and label_t^{M+1} is an accepting run of $\mathcal{A}_2$ on $(\mathrm{pos}(t), \mathrm{label}_t^{\Gamma_\diamond})$.

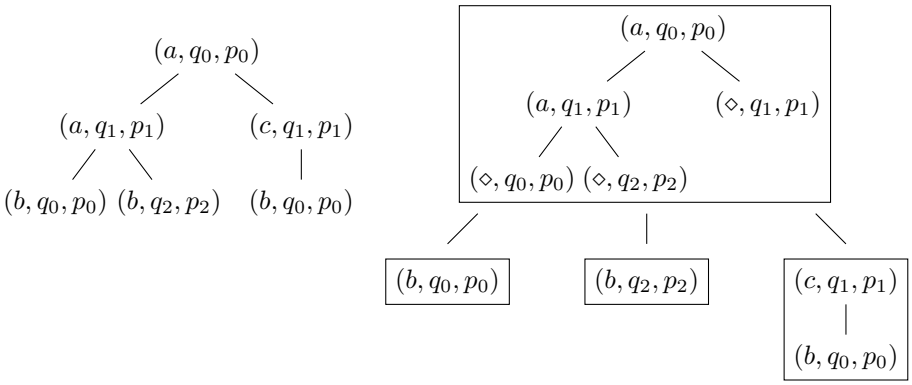


Figure 1 A tree in $T_{\Gamma\times\mathfrak{Q}}$ $(M = 1)$ and a possible cycle decomposition with f defined as $f(\varepsilon) = 1$ and $f(1) = f(2) = f(3) = 0$.

Let $t \in T_{\Gamma_\diamond\times\mathfrak{Q}}$, $n = |\{w \in \mathrm{pos}(t) \mid \mathrm{label}_t^{\Gamma_\diamond}(w) = \diamond\}|$ be the number of $\diamond$-leaves in t and $\{w_1, \dots, w_n\}$ a lexicographically ordered enumeration of these leaves, i.e. $w_1 \leq_l \dots \leq_l w_n$. For $1 \leq i \leq n$, we define $W_i(t) = w_i$. The set

$$\Xi = \{t \in T_{\Gamma_\diamond\times\mathfrak{Q}} \mid \forall w \in \mathrm{pos}(t) : |w| \leq |\mathfrak{Q}| \text{ and } \mathrm{label}_t^{\Gamma_\diamond}(\varepsilon) \neq \diamond\}$$

forms a ranked alphabet where for t as above we define $\mathrm{rk}_\Xi(t) = n$. We recall that $|w|$ denotes the length of w and $|\mathfrak{Q}|$ the cardinality of $\mathfrak{Q}$.

Every tree over Ξ corresponds to a unique tree over $\Gamma \times \mathfrak{Q}$. This inclusion $\mathcal{J} \colon T_\Xi \hookrightarrow T_{\Gamma\times\mathfrak{Q}}$ is formally given as follows. For $\mathfrak{t} \in T_\Xi$ with $\mathrm{pos}(\mathfrak{t}) = \{\varepsilon\}$ we let $\mathcal{J}(\mathfrak{t}) = \mathfrak{t}(\varepsilon)$. Otherwise let $m = \mathrm{rk}_\Xi(\mathfrak{t}(\varepsilon))$ and $\mathcal{J}(\mathfrak{t}) = \mathfrak{t}(\varepsilon)\langle\mathcal{J}(\mathfrak{t}|_1) \to W_1(\mathfrak{t}(\varepsilon))\rangle \dots \langle\mathcal{J}(\mathfrak{t}|_m) \to W_m(\mathfrak{t}(\varepsilon))\rangle$.

A tree $\mathfrak{t} \in T_\Xi$ is called *matching* if for all $w \in \mathrm{pos}(\mathfrak{t})$ and $i \in \{1, \dots, \mathrm{rk}_\Xi(\mathfrak{t}(w))\}$ we have $\pi_\mathfrak{Q}(\mathfrak{t}(w)(W_i(\mathfrak{t}(w)))) = \pi_\mathfrak{Q}(\mathfrak{t}(wi)(\varepsilon))$, i.e. the "state labels" of the $\diamond$-letters match with the state labels of the root at the corresponding child.

▶ **Definition 5.** Let $t \in T_{\Gamma\times\mathfrak{Q}}$. A *cycle decomposition of* t is a pair $\mathfrak{D} = (\mathfrak{t}, f)$, where $\mathfrak{t} \in T_\Xi$ and $f \colon \mathrm{pos}(\mathfrak{t}) \to \mathbb{N}$ is a mapping, satisfying the following.

1. We have $\mathcal{J}(\mathfrak{t}) = t$ and $\mathfrak{t}$ is matching.
2. For all $w \in \mathrm{pos}(\mathfrak{t})$ we have $f(w) \leq \mathrm{rk}_\Xi(\mathfrak{t}(w))$.
3. If $w \in \mathrm{pos}(\mathfrak{t})$ with $f(w) > 0$, then with $v = W_{f(w)}(\mathfrak{t}(w))$ we have $\pi_\mathfrak{Q}(\mathfrak{t}(w)(\varepsilon)) = \pi_\mathfrak{Q}(\mathfrak{t}(w)(v))$. In other words, the state labels of the root of $\mathfrak{t}(w)$ coincide with the state labels of the $f(w)$-th $\diamond$-leaf of $\mathfrak{t}(w)$.
4. If $f(w) = 0$ for some $w \in \mathrm{pos}(\mathfrak{t})$, then for all $i > 0$ with $wi \in \mathrm{pos}(\mathfrak{t})$, we have $f(wi) > 0$.

We call $w \in \mathrm{pos}(\mathfrak{t})$ a *cycle* if $f(w) > 0$ and otherwise a *link*. The set of all cycles of $\mathfrak{D}$ is denoted by $\mathrm{Cyc}(\mathfrak{D})$. By $\mathrm{Decomp}(t)$ we denote the set of all cycle decompositions of t. For $w \in \mathrm{pos}(\mathfrak{t})$ we call the set $\mathrm{Anc}_\mathfrak{D}(w) = \{v \in \mathrm{Cyc}(\mathfrak{D}) \mid vi \leq_p w \text{ for some } i \geq 1 \text{ with } i \neq f(v)\}$ the *ancestors* of w, see also Figure 2. We have the following lemma.

▶ **Lemma 6.** *For every $t \in T_{\Gamma\times\mathfrak{Q}}$, there exists a cycle decomposition $(\mathfrak{t}, f) \in \mathrm{Decomp}(t)$.*

For $w \in \mathrm{Cyc}(\mathfrak{D})$ we now define a new tree $\mathfrak{t}_\mathfrak{D}(w)$ over the alphabet $\Xi \cup \{\diamond\}$, where $\diamond$ has rank 0. Intuitively, $\mathfrak{t}_\mathfrak{D}(w)$ is the subtree of $\mathfrak{t}$ at w, with all cycles apart from w itself removed and the subtree at $wf(w)$ replaced by $\diamond$, see also Figure 2.

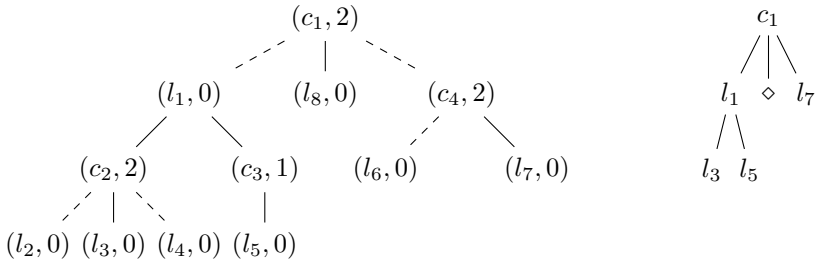


Figure 2 A cycle decomposition $\mathfrak{D}$ with f included in the labels and the tree $\mathfrak{t}_{\mathfrak{D}}(\varepsilon)$. A c denotes a cycle and an l a link. The position of l_8 has no ancestors, l_5 has only the ancestor c_1, and l_4 has ancestors c_1 and c_2.

Formally, we construct the tree as follows. We write f into the labels of $\mathfrak{t}$, i.e. we define $\mathfrak{t}' = (\mathrm{pos}(\mathfrak{t}), \mathrm{label}_{\mathfrak{t}} \times f)$. By π_1 and π_2 we denote the projections of $\Xi \times \mathbb{N}$ to the respective entries. We let $\mathfrak{s} = \mathfrak{t}'|_w$. Now, as long as there is $v \in \mathrm{pos}(\mathfrak{s})$ with $\pi_2(\mathfrak{s}(v)) > 0$ and $v \neq \varepsilon$, we redefine $\mathfrak{s} = \mathfrak{s}\langle \mathfrak{s}|_{v\pi_2(\mathfrak{s}(v))} \to v\rangle$. Finally, we let $\mathfrak{s}' = (\mathrm{pos}(\mathfrak{s}), \pi_1 \circ \mathrm{label}_{\mathfrak{s}})$ and $\mathfrak{t}_{\mathfrak{D}}(w) = \mathfrak{s}'\langle \diamond \to f(w)\rangle$. Note that $\mathfrak{t}_{\mathfrak{D}}(w)$ is matching.

For $i \in \{1, \ldots, M+1\}$ and $w \in \mathrm{Cyc}(\mathfrak{D})$ we define

$$\mathrm{wt}_i^{\mathfrak{D}}(w) = \sum_{\substack{v \in \mathrm{pos}(\mathfrak{t}_{\mathfrak{D}}(w)) \\ \mathrm{label}_{\mathfrak{t}_{\mathfrak{D}}(w)}(v) \neq \diamond}} \mathrm{wt}_i(\mathrm{label}_{\mathfrak{t}_{\mathfrak{D}}(w)}(v))$$

$$b_i^{\mathfrak{D}} = \sum_{\substack{v \in \mathrm{pos}(\mathfrak{t}) \\ v \notin \mathrm{Cyc}(\mathfrak{D}) \\ \mathrm{Anc}_{\mathfrak{D}}(v) = \emptyset}} \mathrm{wt}_i(\mathfrak{t}(v)) + \begin{cases} \nu_1(\mathrm{label}^i_{\mathfrak{t}(\varepsilon)}(\varepsilon)) & \text{if } 1 \leq i \leq M \\ \nu_2(\mathrm{label}^{M+1}_{\mathfrak{t}(\varepsilon)}(\varepsilon)) & \text{if } i = M+1. \end{cases}$$

We have the following lemma.

▶ **Lemma 7.** *Let $t \in T_{\Gamma \times \mathfrak{Q}}$, $s = (\mathrm{pos}(t), \mathrm{label}_t^{\Gamma_\diamond})$, $r_i = \mathrm{label}_t^i$ and $\mathfrak{D} = (\mathfrak{t}, f) \in \mathrm{Decomp}(t)$. Let $\{w_1, \ldots, w_n\}$ be a lexicographically ordered enumeration of $\mathrm{Cyc}(\mathfrak{D})$. Then*

$$\begin{aligned} \nu_1(r_i(\varepsilon)) \quad + \mathrm{wt}_{\mathcal{A}_1}(s, r_i) \quad &= b_i^{\mathfrak{D}} \quad + \mathrm{wt}_i^{\mathfrak{D}}(w_1) \quad + \ldots + \mathrm{wt}_i^{\mathfrak{D}}(w_n) \\ \nu_2(r_{M+1}(\varepsilon)) + \mathrm{wt}_{\mathcal{A}_2}(s, r_{M+1}) &= b_{M+1}^{\mathfrak{D}} + \mathrm{wt}_{M+1}^{\mathfrak{D}}(w_1) + \ldots + \mathrm{wt}_{M+1}^{\mathfrak{D}}(w_n) \end{aligned}$$

for $i \in \{1, \ldots, M\}$.

Let $\{w_1, \ldots, w_n\}$ be a lexicographically ordered enumeration of $\mathrm{Cyc}(\mathfrak{D})$. We consider the system of linear inequalities

$$b_i^{\mathfrak{D}} + \mathrm{wt}_i^{\mathfrak{D}}(w_1)X_1 + \ldots + \mathrm{wt}_i^{\mathfrak{D}}(w_n)X_n < b_{M+1}^{\mathfrak{D}} + \mathrm{wt}_{M+1}^{\mathfrak{D}}(w_1)X_1 + \ldots + \mathrm{wt}_{M+1}^{\mathfrak{D}}(w_n)X_n$$
$$0 < X_j$$

where i ranges over $1, \ldots, M$ and j over $1, \ldots n$. For a cycle decomposition $\mathfrak{D} \in \mathrm{Decomp}(t)$, the system above is denoted by $\mathrm{LIS}(\mathfrak{D})$.

▶ **Lemma 8.** *Let $t \in T_{\Gamma \times \mathfrak{Q}}$ be accepting and $\mathfrak{D} = (\mathfrak{t}, f) \in \mathrm{Decomp}(t)$. For every choice of $X_1, \ldots, X_n \in \mathbb{N}$, $X_i \geq 1$, there is an accepting tree $s \in T_{\Gamma \times \mathfrak{Q}}$ with*

$$\begin{aligned} \nu_1(\mathrm{label}_s^i(\varepsilon)) + \quad \mathrm{wt}_i(s) = \quad b_i^{\mathfrak{D}} + \quad \mathrm{wt}_i^{\mathfrak{D}}(w_1)X_1 + \ldots + \quad \mathrm{wt}_i^{\mathfrak{D}}(w_n)X_n \\ \nu_2(\mathrm{label}_s^{M+1}(\varepsilon)) + \mathrm{wt}_{M+1}(s) = b_{M+1}^{\mathfrak{D}} + \mathrm{wt}_{M+1}^{\mathfrak{D}}(w_1)X_1 + \ldots + \mathrm{wt}_{M+1}^{\mathfrak{D}}(w_n)X_n \end{aligned}$$

for every $i \in \{1, \ldots, M\}$.

Proof. For $X_1 = \ldots = X_n = 1$ we know by Lemma 7 that this is true for $s = t$. Otherwise let $w \in \mathrm{Cyc}(\mathfrak{D})$. We can "insert" the tree $\mathfrak{t}_{\mathfrak{D}}(w)$ into $\mathfrak{t}$ at w as follows. Let $\mathfrak{s} = \mathfrak{t}\langle \mathfrak{t}_{\mathfrak{D}}(w) \to wf(w)\rangle\langle \mathfrak{t}|_{wf(w)} \to wf(w)f(w)\rangle$. As w is a cycle, we know that $\mathfrak{s}$ is matching and with $s = \mathcal{J}(\mathfrak{s})$ we have

$$\begin{aligned}\nu_i(\mathrm{label}_s^i(\varepsilon)) + \mathrm{wt}_i(s) &= \nu_i(\mathrm{label}_t^i(\varepsilon)) + \mathrm{wt}_i(t) + \mathrm{wt}_i^{\mathfrak{D}}(w)\\ &= b_i^{\mathfrak{D}} + \mathrm{wt}_i^{\mathfrak{D}}(w_1) + \ldots + \mathrm{wt}_i^{\mathfrak{D}}(w_n) + \mathrm{wt}_i^{\mathfrak{D}}(w)\end{aligned}$$

where the last equality follows from Lemma 7. For every $j \in \{1, \ldots, n\}$ we apply this procedure $X_j - 1$ times to w_j to obtain s as needed. To see that s is indeed accepting, note that $\mathrm{label}_t^1, \ldots, \mathrm{label}_t^M$ are pairwise distinct. Since s is obtained from t by inserting subtrees, $\mathrm{label}_s^1, \ldots, \mathrm{label}_s^M$ must also be pairwise distinct. ◀

We are now ready to prove Theorem 2.

▶ **Lemma 9.** *Let* $N = \sum_{k=0}^{|\Omega|} \mathrm{rk}(\Gamma)^k$, $\Upsilon = \sum_{k=1}^{\mathrm{rk}(\Xi)+1}(|\Xi|+1)^k$, $\Omega = \sum_{k=1}^{\mathrm{rk}(\Xi)+2}(|\Xi|+1)^k$ *and* $\Theta = |\Xi|\Omega(2^\Omega + 2)$. *Then the following statements are equivalent.*

(i) $\mathcal{A}_1 \geq \mathcal{A}_2$.

(ii) *For all accepting* $t \in T_{\Gamma\times\mathfrak{Q}}$ *with* $|\mathrm{pos}(t)| \leq N\Upsilon^2\Theta(2+\mathrm{rk}(\Xi))$ *and all cycle decompositions* $\mathfrak{D} \in \mathrm{Decomp}(t)$, *the system of linear inequalities* $\mathrm{LIS}(\mathfrak{D})$ *does not possess an integer solution.*

(iii) *For all accepting* $t \in T_{\Gamma\times\mathfrak{Q}}$ *and all cycle decompositions* $\mathfrak{D} \in \mathrm{Decomp}(t)$, *the system of linear inequalities* $\mathrm{LIS}(\mathfrak{D})$ *does not possess an integer solution.*

Property (ii) is clearly decidable. There are only finitely many trees to check, each tree has only finitely many cycle decompositions, and the satisfiability of the corresponding linear inequality systems with integers is decidable due to [15]. In particular, Theorem 2 holds.

Proof (sketch).

(i) ⇒ (iii). We prove this by contradiction and assume that (iii) does not hold. Then there is an accepting $t \in T_{\Gamma\times\mathfrak{Q}}$ and a cycle decomposition $\mathfrak{D} \in \mathrm{Decomp}(t)$ such that the system of inequalities $\mathrm{LIS}(\mathfrak{D})$ has an integer solution. By Lemma 8 we can find an accepting tree $s \in T_{\Gamma\times\mathfrak{Q}}$ with

$$\begin{aligned}\nu_1(\mathrm{label}_s^i(\varepsilon)) + \quad \mathrm{wt}_i(s) = \quad b_i^{\mathfrak{D}} + \quad \mathrm{wt}_i^{\mathfrak{D}}(w_1)X_1 + \ldots + \quad \mathrm{wt}_i^{\mathfrak{D}}(w_n)X_n\\ \nu_2(\mathrm{label}_s^{M+1}(\varepsilon)) + \mathrm{wt}_{M+1}(s) = b_{M+1}^{\mathfrak{D}} + \mathrm{wt}_{M+1}^{\mathfrak{D}}(w_1)X_1 + \ldots + \mathrm{wt}_{M+1}^{\mathfrak{D}}(w_n)X_n\end{aligned}$$

for every $i \in \{1, \ldots, M\}$. Thus by Lemma 7 with $s' = (\mathrm{pos}(t), \mathrm{label}_s^{\Gamma_\diamond})$ and $r_i = \mathrm{label}_s^i$ for $i \in \{1, \ldots, M+1\}$ we have $\nu_1(r_i(\varepsilon)) + \mathrm{wt}_{\mathcal{A}_1}(s', r_i) < \nu_2(r_{M+1}(\varepsilon)) + \mathrm{wt}_{\mathcal{A}_2}(s', r_{M+1})$ for all $i \in \{1, \ldots, M\}$. Since $\mathcal{A}_1$ is M-ambiguous and $r_1, \ldots, r_M$ are pairwise distinct, this means $[\![\mathcal{A}_1]\!](s') < [\![\mathcal{A}_2]\!](s')$, i.e. (i) does not hold.

(iii) ⇒ (i). We show this by contradiction and assume that (i) does not hold. Then there is some tree $s \in \mathrm{supp}(\mathcal{A}_2)$ with $[\![\mathcal{A}_1]\!](s) < [\![\mathcal{A}_2]\!](s)$. Let $\mathrm{Acc}_{\mathcal{A}_1}(s) = \{r_1, \ldots, r_M\}$. Since $[\![\mathcal{A}_1]\!](s) < [\![\mathcal{A}_2]\!](s)$, there must be $r_{M+1} \in \mathrm{Acc}_{\mathcal{A}_2}(s)$ with $\nu_1(r_i(\varepsilon)) + \mathrm{wt}_{\mathcal{A}_1}(s, r_i) < \nu_2(r_{M+1}(\varepsilon)) + \mathrm{wt}_{\mathcal{A}_2}(s, r_{M+1})$ for all $i \in \{1, \ldots, M\}$. Consider the accepting tree $t = (\mathrm{pos}(s), (\mathrm{label}_s, r_1, \ldots, r_{M+1})) \in T_{\Gamma\times\mathfrak{Q}}$ and let $\mathfrak{D} = (\mathfrak{t}, f) \in \mathrm{Decomp}(t)$. Then according to Lemma 7, the system $\mathrm{LIS}(\mathfrak{D})$ clearly has the integer solution $X_1 = \ldots = X_n = 1$, i.e. (iii) does not hold.

(ii) ⇔ (iii). The direction (iii) ⇒ (ii) is clear. We prove (ii) ⇒ (iii) by induction on the size of the trees t. For "small" trees, it follows by assuming (ii) as true. For "large" trees t, we

show that if for a cycle decomposition $\mathfrak{D} = (\mathfrak{t}, f)$ the system $\mathrm{LIS}(\mathfrak{D})$ has an integer solution, then we can find a smaller tree and a cycle decomposition $\mathfrak{D}'$ of that tree for which $\mathrm{LIS}(\mathfrak{D}')$ also has an integer solution. This constitutes a contradiction to our induction hypothesis.

The main issue is how to construct this smaller tree. For words, it is easy. If a word is sufficiently long, there are two cycles with the same label. We remove one of these cycles from the word, thereby making the word shorter, and in $\mathrm{LIS}(\mathfrak{D})$ add the coefficient for this cycle to that of the other, identical cycle. It is clear that this is not possible for trees. By removing any cycle w, we also remove all other cycles v with $w \in \mathrm{Anc}_{\mathfrak{D}}(v)$.

Our solution for this is as follows. Using the concept of ancestors, we construct a tree hierarchy on the cycles of $\mathfrak{D}$. The "child cycles" of a given cycle w are all cycles for which w is the prefix-largest ancestor. We call this tree $\mathfrak{T}$ and consider two different cases. If $\mathfrak{T}$ has sufficiently many leaves, there are two different leaves pointing to the "same cycle". More precisely, we find $w_1 \neq w_2$ in $\mathrm{Cyc}(\mathfrak{D})$ with $\mathfrak{t}_{\mathfrak{D}}(w_1) = \mathfrak{t}_{\mathfrak{D}}(w_2)$ and for all $v \in \mathrm{Cyc}(\mathfrak{D})$ we have $w_1 \notin \mathrm{Anc}_{\mathfrak{D}}(v)$ and $w_2 \notin \mathrm{Anc}_{\mathfrak{D}}(v)$. The leaves of $\mathfrak{T}$ correspond to cycles which are save to remove as they are not ancestors of any other cycles. We can therefore remove w_2 and add this cycle's coefficient to that of w_1.

However, $\mathfrak{T}$ might not have sufficiently many leaves for this argumentation. But assuming that the number of leaves stays below the bound Υ, sufficiently large trees $\mathfrak{T}$ have arbitrarily long successions of nodes $w, w1, w1^2, \ldots, w1^n$ each having only one child. Now consider the cycles $v_0, \ldots, v_n \in \mathrm{Cyc}(\mathfrak{D})$ which correspond to such a succession $w, w1, \ldots, w1^n$. We can show that there is only a finite number of possible trees $\mathfrak{t}_{\mathfrak{D}}(v_i)$ for these cycles. If n is large enough, we can find $i_1 < i_2 < i_3 < i_4$ such that $\mathfrak{t}_{\mathfrak{D}}(v_{i_1}) = \mathfrak{t}_{\mathfrak{D}}(v_{i_2}) = \mathfrak{t}_{\mathfrak{D}}(v_{i_3}) = \mathfrak{t}_{\mathfrak{D}}(v_{i_4})$ and in addition $\{\mathfrak{t}_{\mathfrak{D}}(v_i) \mid i_1 \leq i \leq i_2\} = \{\mathfrak{t}_{\mathfrak{D}}(v_i) \mid i_3 \leq i \leq i_4\}$. We can then "remove" all cycles $v_{i_3}, \ldots, v_{i_4-1}$ by inserting the subtree of $\mathfrak{t}$ at v_{i_4} into the node v_{i_3}. The coefficients for the cycles removed in this way can then be added to the coefficients of the corresponding cycles in $v_{i_1}, \ldots, v_{i_2}$. ◀

On the Proof for the Word Case

For words, Theorem 1 was shown by Hashiguchi et al. There are two different versions of the paper, namely [8, 9]. In both papers, it is first shown that for deterministic max-plus word automata $\mathcal{A}_1, \ldots, \mathcal{A}_{M+1}$, it is decidable whether $\max_{i=1}^{M} [\![\mathcal{A}_i]\!] \geq [\![\mathcal{A}_{M+1}]\!]$. The approach for the generalization to finitely ambiguous automata is then different in both papers.

In [8], it is claimed that every finitely ambiguous max-plus word automaton can be written as a pointwise maximum of finitely many deterministic max-plus automata. This argumentation was withdrawn in [9] and the claim posed as an open problem. It does in fact not hold as shown in [1].

In [9], the argumentation is done directly on the runs of the finitely ambiguous max-plus automata. However, this causes problems when not all words have the same number of accepting runs. The two automata in Figure 3 over the one-letter alphabet $\{a\}$ constitute a counter example to Theorem 5.6 in [9], which is similar to our Lemma 9.

One easily checks that $\mathcal{A}_1 \geq \mathcal{A}_2$. There are two accepting runs of $\mathcal{A}_1$ on a^3, namely $q_1 a q_2 a q_3 a q_3$ and $q_4 a q_5 a q_6 a q_6$, and one of $\mathcal{A}_2$ on a^3, namely $p_1 a p_2 a p_3 a p_3$. The last a thus induces a cycle in the sense of [9]. From this, the linear inequality system

$$\begin{aligned} 2+ (-1) \cdot X &< 1+ 0 \cdot X \\ -2+ \quad 1 \cdot X &< 1+ 0 \cdot X \\ 0 &\leq X \end{aligned}$$

is derived. It clearly has the solution $X = 2$. However, it is stated that from the satisfiability

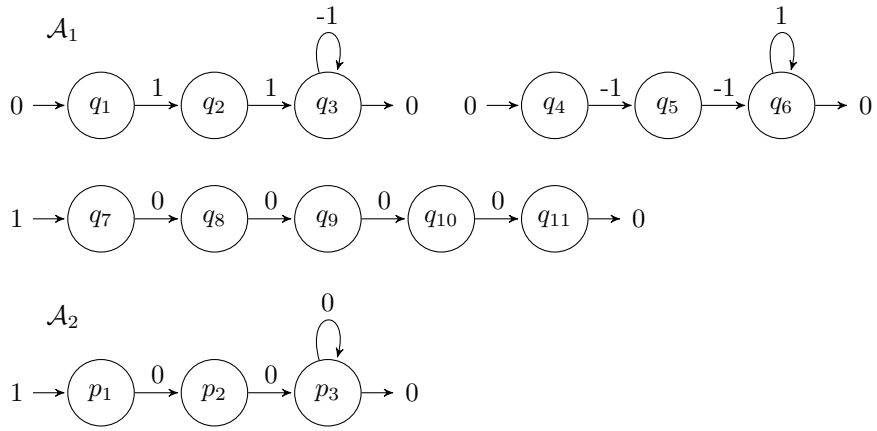


Figure 3 The automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over the alphabet $\{a\}$ constitute a counter example to [9, Theorem 5.6]. The transition letters are omitted.

of this inequality system with an integer value it follows that $\mathcal{A}_1 \geq \mathcal{A}_2$ does not hold. The problem here is that the word a^4, which is supposed to "realize" the solution of the inequality system, in fact possesses a third accepting run $q_7aq_8aq_9aq_{10}aq_{11}$ which compensates the other two runs.

The proof of Theorem 5.6 in [9] can easily be fixed by normalizing the automaton $\mathcal{A}_1$ as we did in Lemma 3. If for some $M \geq 1$ we have $|\mathrm{Acc}_{\mathcal{A}_1}(w)| \in \{0, M\}$ for every word w, all arguments of the proof work as intended.

4 The Unambiguity Problem

The unambiguity problem asks whether for a given max-plus-WTA $\mathcal{A}$ there exists an unambiguous max-plus-WTA $\mathcal{A}'$ such that $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$. In this section, we show that the unambiguity problem is decidable for finitely ambiguous max-plus-WTA. We follow ideas from [11, Section 5], where the decidability of this problem was shown for finitely ambiguous max-plus word automata. The unambiguity problem is, in fact, even known to be decidable for polynomially ambiguous max-plus word automata [10]. We leave the question open as to whether the same holds true for polynomially ambiguous max-plus-WTA.

▶ **Theorem 10.** *For a finitely ambiguous max-plus-WTA $\mathcal{A}$ it is decidable whether there exists an unambiguous max-plus-WTA $\mathcal{A}'$ with $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$. If $\mathcal{A}'$ exists, it can be effectively constructed.*

The rest of this section is dedicated to the proof of Theorem 10.

For an alphabet Γ, a tree over the alphabet $\Gamma_\diamond = (\Gamma \cup \{\diamond\}, \mathrm{rk}_\Gamma \cup \{\diamond \mapsto 0\})$ is called a Γ-*context*. For a max-plus-WTA $\mathcal{A} = (Q, \Gamma, \mu, \nu)$, a run of $\mathcal{A}$ on a Γ-context t is a run of the max-plus-WTA $\mathcal{A}' = (Q, \Gamma_\diamond, \mu', \nu)$ on t, where $\mu'(\diamond, q) = 0$ for all $q \in Q$ and $\mu'(d) = \mu(d)$ for $d \in \Delta_\mathcal{A}$. We denote $\mathrm{Run}^\diamond_\mathcal{A}(t) = \mathrm{Run}_{\mathcal{A}'}(t)$ and for $r \in \mathrm{Run}^\diamond_\mathcal{A}(t)$ write $\mathrm{wt}^\diamond_\mathcal{A}(t, r) = \mathrm{wt}_{\mathcal{A}'}(t, r)$.

For $s \in T_\Gamma$ with $|\{w \in \mathrm{pos}(s) \mid s(w) = \diamond\}| = 1$ and $r \in \mathrm{Run}^\diamond_\mathcal{A}(s)$ such that for $w_0 \in \mathrm{pos}(s)$ with $s(w_0) = \diamond$ we have $r(\varepsilon) = r(w_0)$ the pair (s, r) is called an $\mathcal{A}$-*circuit*. We call (s, r) *small* if $|w| \leq |Q|$ for all $w \in \mathrm{pos}(s)$.

Now let $\mathcal{A}$ be a finitely ambiguous max-plus-WTA. We decompose $\mathcal{A}$ into unambiguous max-plus-WTA as follows.

▶ **Lemma 11** ([16]). *Let $\mathcal{A}$ be a finitely ambiguous max-plus-WTA over Γ, then there exist finitely many unambiguous max-plus-WTA $\mathcal{A}_1, \dots, \mathcal{A}_M$ over Γ with $[\![\mathcal{A}]\!] = \max_{i=1}^{M} [\![\mathcal{A}_i]\!]$ and $\operatorname{supp}(\mathcal{A}_1) = \dots = \operatorname{supp}(\mathcal{A}_M)$.*

Let $\mathcal{A}_1, \dots, \mathcal{A}_M$ be unambiguous max-plus-WTA with $\operatorname{supp}(\mathcal{A}_1) = \dots = \operatorname{supp}(\mathcal{A}_M)$ and $[\![\mathcal{A}]\!] = \max_{i=1}^{M} [\![\mathcal{A}_i]\!]$. We write $\mathcal{A}_i = (Q_i, \Gamma, \mu_i, \nu_i)$ for $i \in \{1, \dots, M\}$. The *product automaton of $\mathcal{A}_1, \dots, \mathcal{A}_M$* is the trimmed automaton $\mathcal{B} = (Q, \Gamma, \mu, \nu)$ over the product semiring $(\mathbb{R}_{\max})^M$ defined as follows. We let $Q = Q_1 \times \dots \times Q_M$ and for $a \in \Gamma$ with $\operatorname{rk}_\Gamma(a) = m$ and $\mathbf{p}_0, \dots, \mathbf{p}_m \in Q$ with $\mathbf{p}_i = (p_{i1}, \dots, p_{iM})$ we define with $x_j = \mu_j(p_{1j}, \dots, p_{mj}, a, p_{0j})$ and $y_j = \nu_j(p_{0j})$

$$\mu(\mathbf{p}_1, \dots, \mathbf{p}_m, a, \mathbf{p}_0) = \begin{cases} (x_1, \dots, x_M) & \text{if } (x_1, \dots, x_M) \in \mathbb{R}^M \\ (-\infty, \dots, -\infty) & \text{otherwise} \end{cases}$$

$$\nu(\mathbf{p}_0) = \begin{cases} (y_1, \dots, y_M) & \text{if } (y_1, \dots, y_M) \in \mathbb{R}^M \\ (-\infty, \dots, -\infty) & \text{otherwise.} \end{cases}$$

Then $\mathcal{B}$ is unambiguous and for $t \in T_\Gamma$ we have $[\![\mathcal{B}]\!](t) = ([\![\mathcal{A}_1]\!](t), \dots, [\![\mathcal{A}_M]\!](t))$.

For $\mathbf{q}, \mathbf{p} \in Q$, we write $\mathbf{q} \preceq \mathbf{p}$ if there exists $t \in T_\Gamma$, $r \in \operatorname{Acc}_\mathcal{B}(t)$ and $w_1, w_2 \in \operatorname{pos}(t)$ with $w_1 \leq_p w_2$ such that $r(w_1) = \mathbf{q}$ and $r(w_2) = \mathbf{p}$. We write $\mathbf{q} \approx \mathbf{p}$ if $\mathbf{q} \preceq \mathbf{p}$ and $\mathbf{p} \preceq \mathbf{q}$. By $[\mathbf{q}]$ we denote the set of all $\mathbf{p} \in Q$ with $\mathbf{q} \approx \mathbf{p}$.

▶ **Definition 12.** Let $s \in T_{\Gamma_\diamond}$ be a Γ-context, $r \in \operatorname{Run}^\diamond_\mathcal{B}(s)$ and write $\operatorname{wt}^\diamond_\mathcal{B}(s, r) = (\theta_1, \dots, \theta_M)$. We define $\operatorname{wt}_i(s, r) = \theta_i$ and $\operatorname{wt}(s, r) = \max_{i=1}^{M} \operatorname{wt}_i(s, r)$.

A coordinate $i \in \{1, \dots, M\}$ is called *victorious* if $\operatorname{wt}_i(s, r) = \operatorname{wt}(s, r)$. The set of all victorious coordinates of (s, r) is denoted by $\operatorname{Vict}(s, r)$. For $\mathbf{q} \in Q$ we define

$$\operatorname{Vict}([\mathbf{q}]) = \bigcap_{\substack{(s,r) \text{ small } \mathcal{B}\text{-circuit} \\ r(\varepsilon) \in [\mathbf{q}]}} \operatorname{Vict}(s, r)$$

where the empty intersection is defined as $\{1, \dots, M\}$. For $P \subseteq Q$, we let $\operatorname{Vict}(P) = \bigcap_{\mathbf{p} \in P} \operatorname{Vict}([\mathbf{p}])$. We have the following lemma.

▶ **Lemma 13.** *There is an unambiguous max-plus-WTA $\mathcal{A}'$ with $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$ if and only if for all $t \in T_\Gamma$ and all $r \in \operatorname{Acc}_\mathcal{B}(t)$ we have $\operatorname{Vict}(r(\operatorname{pos}(t))) \neq \emptyset$. The latter property is called the* dominance property *and is denoted by (**P**).*

(**P**) is decidable as follows. We can consider Q as an (unranked) alphabet and construct an FTA which accepts exactly the accepting runs of $\mathcal{B}$, i.e. all pairs $(\operatorname{pos}(t), r)$ for some $t \in T_\Gamma$ and $r \in \operatorname{Acc}_\mathcal{B}(t)$. Also, for $P \subseteq Q$ we can construct an FTA which accepts all trees in T_Q in which every $p \in P$ occurs at least once as a label. By taking the intersection of these two automata and checking for emptiness, we can decide for every $P \subseteq Q$ whether there is any $t \in T_\Gamma$ and $r \in \operatorname{Acc}_\mathcal{B}(t)$ with $P \subseteq r(\operatorname{pos}(t))$. Checking whether all P for which this is true satisfy $\operatorname{Vict}(P) \neq \emptyset$ is equivalent to checking (**P**).

▶ **Construction 14.** Let $N = \sum_{i=0}^{|Q|} \operatorname{rk}(\Gamma)^i$, $R = \bigcup_{i=1}^{M} (\mu_i(\Delta_{\mathcal{A}_i}) \cup \nu_i(Q_i))$ and $C = \max R - \min(R \setminus \{-\infty\})$. For $\mathbf{x} = (x_1, \dots, x_M) \in \mathbb{R}^M_{\max}$ we let $\check{\mathbf{x}} = \min\{x_i \mid 1 \leq i \leq M, x_i \neq -\infty\}$ and $\underline{\mathbf{x}} = \mathbf{x} - (\check{\mathbf{x}}, \dots, \check{\mathbf{x}})$.

Assume that $\mathcal{B}$ satisfies (**P**). We construct an unambiguous max-plus-WTA $\mathcal{A}' = (Q', \Gamma, \mu', \nu')$ with $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$ and $Q' \subset \mathbb{R}^M_{\max} \times Q$ as follows.

Rule 1: For $(a, \mathbf{q}) \in \Delta_\mathcal{B} \cap (\Gamma \times Q)$ with $\mathbf{x} = \mu(a, \mathbf{q}) \in \mathbb{R}^M$, we let $(\underline{\mathbf{x}}, \mathbf{q}) \in Q'$ and $\mu'(a, (\underline{\mathbf{x}}, \mathbf{q})) = \check{\mathbf{x}}$.

Rule 2: Assume for $(\mathbf{z}_1, \mathbf{p}_1), \dots, (\mathbf{z}_m, \mathbf{p}_m) \in Q'$ that we have $d = (\mathbf{p}_1, \dots, \mathbf{p}_m, a, \mathbf{p}_0) \in \Delta_{\mathcal{B}}$ for some $a \in \Gamma$, $\mathbf{p}_0 \in Q$ and $\mathbf{x} = \mu(d) \in \mathbb{R}^M$. We let $\mathbf{t} = \sum_{i=1}^m \mathbf{z}_i + \mathbf{x}$ and define $\mathbf{y} \in \mathbb{R}^M_{\max}$ through

$$y_i = \begin{cases} -\infty & \text{if } t_i < \max\{t_j \mid 1 \le j \le M\} - (2N+1)C \\ t_i & \text{otherwise.} \end{cases}$$

We let $(\underline{\mathbf{y}}, \mathbf{p}_0) \in Q'$ and $\mu'((\mathbf{z}_1, \mathbf{p}_1), \dots, (\mathbf{z}_m, \mathbf{p}_m), a, (\underline{\mathbf{y}}, \mathbf{p}_0)) = \check{\mathbf{y}}$.

Finally, assume $(\mathbf{z}, \mathbf{p}) \in Q'$ and $\mathbf{x} = \nu(\mathbf{p}) \in \mathbb{R}^M$. Then we let $\nu'(\mathbf{z}, \mathbf{p}) = \max_{i=1}^M z_i + x_i$.

▶ **Lemma 15.** *$\mathcal{A}'$ is an unambiguous max-plus-WTA with $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$.*

Proof (sketch). $\mathcal{A}'$ is unambiguous as there is a bijection between the accepting runs of $\mathcal{B}$ and $\mathcal{A}'$. The idea behind $\mathcal{A}'$ is as follows. From a bottom-up perspective, $\mathcal{A}'$ remembers in each coordinate of $\mathbf{z}$ the weight which $\mathcal{B}$ would have assigned to the run in this coordinate "so far". Since this can become unbounded, we normalize the smallest coordinate to 0 in each transition, make this coordinate's weight the transition weight, and remember only the difference to this weight in the remaining coordinates. Still, these differences can become unbounded. Therefore, once the difference exceeds the bound $(2N+1)C$, the coordinates with small weights are discarded by being set to $-\infty$.

We can show that the coordinate k which in $\mathcal{B}$ eventually yields the largest weight will not be discarded. First, we can show that a victorious coordinate of a run will never be smaller than the largest weight (over all coordinates) minus NC. Second, we can show that if l is victorious, then the weight of coordinate k will never be smaller than the weight of l minus $NC + C$. Our assumption is that (**P**) holds, so there exists some victorious coordinate in every accepting run. Therefore, the weight of k will never be smaller than the largest weight minus $(2N+1)C$ and is never discarded. ◀

We now prove that (**P**) is a necessary condition, i.e. that from the existence of an unambiguous automaton $\mathcal{A}'$ with $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$ it follows that $\mathcal{B}$ satisfies (**P**).

▶ **Lemma 16.** *If there exists an unambiguous max-plus-WTA $\mathcal{A}' = (Q', \Gamma, \mu', \nu')$ with $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$ then $\mathcal{B}$ satisfies (**P**).*

Proof. Let $t \in T_\Gamma$ and $r \in \mathrm{Run}_{\mathcal{B}}(t)$. Let $\mathcal{C} = \{(s, r_s) \text{ small } \mathcal{B}\text{-circuit} \mid [r_s(\varepsilon)] \cap r(\mathrm{pos}(t)) \neq \emptyset\}$. Let $\mathbf{p} \in r(\mathrm{pos}(t))$, $(s, r_s) \in \mathcal{C}$ and $\mathbf{q} = r_s(\varepsilon) \in [\mathbf{p}]$.

We can assume that $\mathbf{q} \in r(\mathrm{pos}(t))$ due to the following argument. Since to $\mathbf{p} \preceq \mathbf{q} \preceq \mathbf{p}$, we can find $t^{\mathbf{p}}_{\mathbf{q}}, t^{\mathbf{q}}_{\mathbf{p}} \in T_\Gamma$, $r^{\mathbf{p}}_{\mathbf{q}} \in \mathrm{Run}_{\mathcal{B}}(t^{\mathbf{p}}_{\mathbf{q}})$ and $r^{\mathbf{q}}_{\mathbf{p}} \in \mathrm{Run}_{\mathcal{B}}(t^{\mathbf{q}}_{\mathbf{p}})$ such that $r^{\mathbf{p}}_{\mathbf{q}}(\varepsilon) = \mathbf{p}$, $r^{\mathbf{q}}_{\mathbf{p}}(\varepsilon) = \mathbf{q}$ and for some $w_{\mathbf{q}} \in \mathrm{pos}(t^{\mathbf{p}}_{\mathbf{q}})$ and $w_{\mathbf{p}} \in \mathrm{pos}(t^{\mathbf{q}}_{\mathbf{p}})$ we have $r^{\mathbf{p}}_{\mathbf{q}}(w_{\mathbf{q}}) = \mathbf{q}$ and $r^{\mathbf{q}}_{\mathbf{p}}(w_{\mathbf{p}}) = \mathbf{p}$. Thus with $s' = t^{\mathbf{p}}_{\mathbf{q}}\langle t^{\mathbf{q}}_{\mathbf{p}}\langle \diamond \to w_{\mathbf{p}}\rangle \to w_{\mathbf{q}}\rangle$ we obtain a circuit $(s', r_{s'})$ with $r_{s'}(\varepsilon) = \mathbf{p}$ and $r_{s'}(w_{\mathbf{q}}) = \mathbf{q}$. We can insert $(s', r_{s'})$ into t and r to obtain a tree t' and a run $r' \in \mathrm{Acc}_{\mathcal{B}}(t')$ with $\mathbf{q} \in r'(\mathrm{pos}(t'))$.

Now let $w_{\mathbf{q}} \in \mathrm{pos}(t)$ with $r(w_{\mathbf{q}}) = \mathbf{q}$ and $w \in \mathrm{pos}(s)$ with $s(w) = \diamond$. We let $s^1 = s$ and for $n \ge 1$ define $s^{n+1} = s\langle s^n \to w\rangle$. Then from r_s we obtain a circuit $(s^{|Q'|}, r_s^{|Q'|})$ which we can insert at $w_{\mathbf{q}}$ to obtain a tree $t' \in T_\Gamma$ and a run $r' \in \mathrm{Acc}_{\mathcal{B}}(t')$. We do this for all small circuits in $\mathcal{C}$ simultaneously. We assume without loss of generality that after this the circuit $(s^{|Q'|}, r_s^{|Q'|})$ is still at position $w_{\mathbf{q}}$. Since $\mathrm{supp}\,\mathcal{B} = \mathrm{supp}\,\mathcal{A}'$, we find a run $r'' \in \mathrm{Acc}_{\mathcal{A}'}(t')$. By pigeon hole principle, we find $0 \le i_1 < i_2 \le |Q'|$ with $r''(w_{\mathbf{q}} w^{i_1}) = r''(w_{\mathbf{q}} w^{i_2})$. From this, we obtain an $\mathcal{A}'$-circuit $(s^{i_2 - i_1}, \hat{r})$ which corresponds to a $\mathcal{B}$-circuit $(s^{i_2-i_1}, r_s^{i_2-i_1})$. We can now insert $s^{i_2-i_1}$ at $w_{\mathbf{q}}$ repeatedly to create copies of these circuits. Clearly, this works for all small circuits in $\mathcal{C}$.

Let $c_1, \dots, c_n$ be an enumeration of $\mathcal{C}$. We write $c_i = (s_i, r_i)$. By $(\hat{s}_i, \hat{r}_i)$ and $(\hat{s}_i, \check{r}_i)$, we denote the circuits in $\mathcal{A}'$ and $\mathcal{B}$, respectively, we obtain from c_i in the way we obtained $(s^{i_2-i_1}, \hat{r})$ and $(s^{i_2-i_1}, r^{i_2-i_1})$ from (s, r_s). For $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{N}^n$, we denote by $t_\mathbf{v}$ the tree obtained by adding v_i copies of $\hat{s}_i$ to t for each $i \in \{1, \dots, n\}$. Since $\mathcal{B}$ and $\mathcal{A}'$ are both unambiguous, we can make the following observations.

For $i \in \{1, \dots, n\}$ we let $\rho_i = \mathrm{wt}_{\mathcal{A}'}(\hat{s}_i, \hat{r}_i)$. Then for some constant ρ_0 we have $[\![\mathcal{A}']\!](t_\mathbf{v}) = \rho_0 + v_1\rho_1 + \ldots + v_n\rho_n$. Due to the definition of victorious coordinates, for every $\mathbf{v}' = (v_2, \dots, v_n) \in \mathbb{N}^{n-1}$ there is $N^{(1)}_{\mathbf{v}'} \in \mathbb{N}$ such that for all $v_1 > N^{(1)}_{\mathbf{v}'}$ the tuple $[\![\mathcal{B}]\!](t_{(v_1,\dots,v_n)})$ has its maximum in entry j_1 for some $j_1 \in \mathrm{Vict}(\hat{s}_1, \check{r}_1)$. Then with $\rho^{(1)}_i = \mathrm{wt}_{j_1}(\hat{s}_i, \check{r}_i)$ for $i \in \{1, \dots, n\}$ and some constant $\rho^{(1)}_0$ we have for all $v_1 > N^{(1)}_{\mathbf{v}'}$ that $[\![\mathcal{B}]\!](t_\mathbf{v}) = \rho^{(1)}_0 + v_1\rho^{(1)}_1 + \ldots + v_n\rho^{(1)}_n$. By varying $\mathbf{v}$, we see that from $[\![\mathcal{A}']\!](t_\mathbf{v}) = [\![\mathcal{B}]\!](t_\mathbf{v})$ it follows that $\rho_i = \rho^{(1)}_i$ for all $i \in \{1, \dots, n\}$. We can do the same for the other circuits $(\hat{s}_2, \check{r}_2), \dots, (\hat{s}_n, \check{r}_n)$ and see that if $j_i \in \mathrm{Vict}(\hat{s}_i, \check{r}_i)$ for every $i \in \{1, \dots, n\}$ then $\mathrm{wt}_{j_1}(\hat{s}_i, \check{r}_i) = \ldots = \mathrm{wt}_{j_n}(\hat{s}_i, \check{r}_i)$ for every $i \in \{1, \dots, n\}$. In particular, $j_1 \in \mathrm{Vict}(\hat{s}_i, \check{r}_i)$ for all $i \in \{1, \dots, n\}$. This means $j_1 \in \mathrm{Vict}(r(\mathrm{pos}(t)))$ and $\mathcal{B}$ satisfies (**P**). ◀

5 The Sequentiality Problem

The sequentiality problem asks whether for a given max-plus-WTA $\mathcal{A}$ there exists a deterministic max-plus-WTA $\mathcal{A}'$ such that $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$. The term "sequentiality" stems from the fact that in the weighted setting, deterministic automata are also often called *sequential*. In this section, we show that the sequentiality problem is decidable for finitely ambiguous max-plus-WTA. For words, this is known due to [11].

Let $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ be a max-plus-WTA. We say that $\mathcal{A}$ satisfies the *twins property* [14, 3] if the following holds. Whenever for $q, q' \in Q$ there exist $t \in T_\Gamma$, $r, r' \in \mathrm{Run}_\mathcal{A}(t)$ with $r(\varepsilon) = q$, $r'(\varepsilon) = q'$ and $\mathcal{A}$-circuits $(s, r_1), (s, r_2)$ with $r_1(\varepsilon) = q$ and $r_2(\varepsilon) = q'$ then $\mathrm{wt}^\diamond_\mathcal{A}(s, r_1) = \mathrm{wt}^\diamond_\mathcal{A}(s, r_2)$.

▶ **Lemma 17.** *Let $\mathcal{A}$ be a trim unambiguous max-plus-WTA. There exists a deterministic max-plus-WTA $\mathcal{A}'$ with $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$ if and only if $\mathcal{A}$ satisfies the twins property. If it exists, it can be effectively constructed.*

Proof (sketch). If $\mathcal{A}$ satisfies the twins property, we know due to [3, Lemma 5.10] that a deterministic max-plus-WTA $\mathcal{A}'$ with $[\![\mathcal{A}']\!] = [\![\mathcal{A}]\!]$ can be effectively constructed.

To show that the twins property is also a necessary condition, we can apply an idea similar to the proof of [14, Theorem 9]. ◀

▶ **Lemma 18** ([3, Theorem 5.17])**.** *For an unambiguous max-plus-WTA $\mathcal{A}$ it is decidable whether $\mathcal{A}$ satisfies the twins property.*

▶ **Theorem 19.** *For a finitely ambiguous max-plus-WTA $\mathcal{A}$ it is decidable whether there exists a deterministic max-plus-WTA $\mathcal{A}'$ with $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$. If $\mathcal{A}'$ exists, it can be effectively constructed.*

Proof. Let $\mathcal{A}$ be a finitely ambiguous max-plus-WTA. Due to Theorem 10 we can decide whether there exists an equivalent unambiguous max-plus-WTA. If this is not the case, $\mathcal{A}$ can also not be determinizable. Otherwise we can effectively construct an unambiguous max-plus-WTA $\mathcal{A}'$ with $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$. Due to Lemma 18 we can decide whether $\mathcal{A}'$ satisfies the twins property, which according to Lemma 17 is equivalent to deciding whether $\mathcal{A}$ is determinizable. ◀

References

1 Sebastian Bala and Artur Koniński. Unambiguous automata denoting finitely sequential functions. In Adrian-Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *Proc. LATA*, volume 7810 of *LNCS*, pages 104–115. Springer, 2013.

2 Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages*. Springer, 1988.

3 Matthias Büchse, Jonathan May, and Heiko Vogler. Determinization of weighted tree automata using factorizations. *Journal of Automata, Languages and Combinatorics*, 15(3/4):229–254, 2010.

4 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 2009.

5 Stéphane Gaubert. Performance evaluation of (max,+) automata. *IEEE T. Automat. Contr.*, 40(12):2014–2025, 1995.

6 Stéphane Gaubert and Jean Mairesse. Modeling and analysis of timed Petri nets using heaps of pieces. *IEEE T. Automat. Contr.*, 44(4):683–697, 1999.

7 Kosaburo Hashiguchi. Algorithms for determining relative star height and star height. *Inf. Comput.*, 78(2):124–169, 1988.

8 Kosaburo Hashiguchi and Kenichi Ishiguro. Decidability of the equivalence problem for finitely ambiguous finance automata. *Sūri Kaiseki Kenkyūsho Kōkyūroku*, 960:23–36, 1996.

9 Kosaburo Hashiguchi, Kenichi Ishiguro, and Shuji Jimbo. Decidability of the equivalence problem for finitely ambiguous finance automata. *IJAC*, 12(3):445–461, 2002.

10 Daniel Kirsten and Sylvain Lombardy. Deciding unambiguity and sequentiality of polynomially ambiguous min-plus automata. In Susanne Albers and Jean-Yves Marion Marion, editors, *Proc. STACS*, volume 3 of *LIPIcs*, pages 589–600. LZI, 2009.

11 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theor. Comput. Sci.*, 327(3):349–373, 2004.

12 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *IJAC*, 4(3):405–426, 1994.

13 Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. Springer, 1986.

14 Mehryar Mohri. Finite-state transducers in language and speech processing. *Comput. Linguist.*, 23(2):269–311, 1997.

15 George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

16 Erik Paul. On finite and polynomial ambiguity of weighted tree automata. In Srečko Brlek and Christophe Reutenauer, editors, *Proc. DLT*, volume 9840 of *LNCS*, pages 368–379. Springer, 2016.

17 Slav Petrov. Latent variable grammars for natural language parsing. In *Coarse-to-Fine Natural Language Processing*, chapter 2, pages 7–46. Springer, 2012.

18 Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer, 1978.

19 Marcel-Paul Schützenberger. On the definition of a family of automata. *Inform. Control*, 4(2–3):245 – 270, 1961.

20 Imre Simon. Limited subsets of a free monoid. In *Proc. FOCS*, pages 143–150. IEEE Computer Society, 1978.

21 Imre Simon. Recognizable sets with multiplicities in the tropical semiring. In Michal Chytil, Ladislav Janiga, and Václav Koubek, editors, *Proc. MFCS*, volume 324 of *LNCS*, pages 107–120. Springer, 1988.

22 Andreas Weber. Finite-valued distance automata. *Theor. Comput. Sci.*, 134(1):225–251, 1994.

New Insights on the (Non-)Hardness of Circuit Minimization and Related Problems*

Eric Allender[1] and Shuichi Hirahara[2]

1 **Department of Computer Science, Rutgers University, Piscataway, NJ, USA**
allender@cs.rutgers.edu
2 **Department of Computer Science, The University of Tokyo, Tokyo, Japan**
hirahara@is.s.u-tokyo.ac.jp

Abstract

The Minimum Circuit Size Problem (MCSP) and a related problem (MKTP) that deals with time-bounded Kolmogorov complexity are prominent candidates for NP-intermediate status. We show that, under very modest cryptographic assumptions (such as the existence of one-way functions), the problem of approximating the minimum circuit size (or time-bounded Kolmogorov complexity) within a factor of $n^{1-o(1)}$ is *indeed* NP-intermediate. To the best of our knowledge, these problems are the first natural NP-intermediate problems under the existence of an arbitrary one-way function.

We also prove that MKTP is hard for the complexity class DET under non-uniform NC^0 reductions. This is surprising, since prior work on MCSP and MKTP had highlighted weaknesses of "local" reductions such as $\leq_{\mathrm{m}}^{\mathsf{NC}^0}$. We exploit this local reduction to obtain several new consequences:

- MKTP is not in $\mathsf{AC}^0[p]$.
- Circuit size lower bounds are equivalent to hardness of a relativized version MKTP^A of MKTP under a class of uniform AC^0 reductions, for a large class of sets A.
- Hardness of MCSP^A implies hardness of MKTP^A for a wide class of sets A. This is the first result directly relating the complexity of MCSP^A and MKTP^A, for any A.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases computational complexity, Kolmogorov complexity, circuit size

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.54

1 Introduction

The Minimum Circuit Size Problem (MCSP) has attracted intense study over the years, because of its close connection with the natural proofs framework of Razborov and Rudich [23], and because it is a prominent candidate for NP-intermediate status. It has been known since [18] that NP-intermediate problems exist, if $\mathsf{P} \neq \mathsf{NP}$, but "natural" candidates for this status are rare. Problems such as factoring and Graph Isomorphism are sometimes put forward as candidates, but there are not strong complexity-theoretic arguments for why these problems should not lie in P. We prove that a very weak cryptographic assumption implies that a $n^{1-o(1)}$ approximation for MCSP is NP-intermediate.

* Supported by NSF grant CCF-1555409 (Allender) and JSPS KAKENHI Grant Numbers JP16J06743 (Hirahara). Proofs of some results have been omitted due to space limits; more details can be found at [6].

MCSP is hard for SZK [4] under BPP reductions, but the situation is quite different, when more restricted notions of reducibility are considered. Recent results [14, 19, 7] have suggested that MCSP might not even be hard for P under logspace reductions (although the evidence is still inconclusive).

The input to MCSP consists of a pair (T, s), where T is a bit string of length 2^m representing the truth-table of an m-variate Boolean function, and $s \in \mathbb{N}$; $(T, s) \in$ MCSP if there is a circuit computing T having size at most s. Note that, for different models of circuit (type of gates, allowable fan-in, etc.) and different measures of size (number of gates, number of wires, size of the description of the circuit, etc.) the resulting MCSP problems might have different complexity. No efficient reduction is known between different variants of the problem. However, all prior work on MCSP (such as [16, 3, 9, 19, 4, 25, 7, 14]) applies equally well to any of these variants. MCSP is also closely related to a type of time-bounded Kolmogorov complexity known as KT, which was defined in [3]. The problem of determining KT complexity, formalized as the language MKTP $= \{(x, s) : \mathrm{KT}(x) \leq s\}$ has often been viewed as just another equivalent "encoding" of MCSP in this prior work. (In particular, our results mentioned in the paragraphs above apply also to MKTP.) Recently, however, some reductions were presented that are not currently known to apply to MCSP [5].

In this section, we outline the ways in which this paper advances our understanding of MCSP and related problems, while reviewing some of the relevant prior work.

Hardness is equivalent to circuit size lower bounds. Significant effort (e.g. [16, 19, 7, 14])) has been made in order to explain why it is so difficult to show NP-hardness of MCSP or MKTP. Most of the results along this line showed implications from hardness of MCSP to circuit size lower bounds: If MCSP or MKTP is NP-hard under some restricted types of reductions, then a circuit size lower bound (which is quite difficult to obtain via current techniques of complexity theory) follows. For example, if MCSP or MKTP is hard for TC^0 under Dlogtime-uniform $\leq_{\mathrm{m}}^{\mathsf{AC}^0}$ reductions, then NP $\not\subseteq$ P/poly and DSPACE$(n) \not\subseteq$ io-SIZE$(2^{\epsilon n})$ [19, 7].

Murray and Williams [19] asked if, in general, circuit lower bounds imply hardness of the circuit minimization problems. We answer their questions affirmatively in certain settings: A stronger lower bound DSPACE$(n) \not\subseteq$ io-SIZE$^{\mathsf{MKTP}}(2^{\epsilon n})$ implies that MKTP is hard for DET under logspace-uniform $\leq_{\mathrm{tt}}^{\mathsf{AC}^0}$ reductions (Theorem 11).

At this point, it is natural to ask if the circuit lower bounds are in fact *equivalent* to hardness of MKTP. We indeed show that this is the case, when we consider the minimum *oracle* circuit size problem. For an oracle A, MCSP^A is the set of pairs (T, s) such that T is computed by a size-s circuit that has "oracle gates" for A in addition to standard AND and OR gates. The related MKTP^A problem asks about the time-bounded Kolmogorov complexity of a string, when the universal Turing machine has access to the oracle A. For many oracles A that are hard for PH, we show that DSPACE$(n) \not\subseteq$ io-SIZE$^A(2^{\epsilon n})$ for some $\epsilon > 0$ *if and only if* MKTP^A is hard for DET under a certain class of reducibilities (Theorem 12).

That is, it is impossible to prove hardness of MKTP^A (under some reducibilities) without proving circuit lower bounds, and vice versa. Our results clearly connect the fact that it is difficult to obtain hardness of MKTP^A with the fact that circuit size lower bounds are difficult.

Hardness under local reductions, and unconditional lower bounds. Murray and Williams [19] showed that MCSP and MKTP are not hard for TC^0 under so-called *local* reductions computable in time less than $\sqrt{n}$ – and thus in particular they are not hard under NC^0

reductions that are very uniform (i.e., there is no routine computable in time $t(n) < n^{.5-\epsilon}$ that, on input (n, i) outputs the $O(1)$ queries upon which the i-th output bit of such an NC^0 circuit depends). Murray and Williams speculated that this might be a promising first step toward showing that MCSP is not hard for NP under Dlogtime-uniform AC^0 reductions, since it follows from [1] that any set that is hard for TC^0 under P-uniform AC^0 reductions is also hard for TC^0 under P-uniform NC^0 reductions. Indeed, the results of Murray and Williams led us to expect that MCSP and MKTP are not even hard for PARITY under non-uniform NC^0 reductions.

Contrary to these expectations, we show that MKTP is hard not only for TC^0 but even for the complexity class DET under non-uniform NC^0 reductions (Theorem 9). Consequently, MKTP is not in $\mathsf{AC}^0[p]$ for any prime p.[1] Note that it is still not known whether MCSP or $R_{\mathrm{KT}} = \{x : \mathrm{KT}(x) \geq |x|\}$ is in $\mathsf{AC}^0[p]$. It is known[2] [3] that neither of these problems is in AC^0. Under a plausible derandomization hypothesis, this non-uniform reduction can be converted into a logspace-uniform $\leq_{\mathrm{tt}}^{\mathsf{AC}^0}$ reduction that is an AND of NC^0-computable queries. Thus "local" reductions are more effective for reductions to MKTP than may have been suspected.

Implications among hardness conditions for MKTP and MCSP. No $\leq_{\mathrm{T}}^{\mathsf{P}}$ reductions are known between MKTP^A or MCSP^A for any A. Although most previous complexity results for one of the problems have applied immediately to the other, via essentially the same proof, there has not been any proven relationship among the problems. For the first time, we show that, for many oracles A, hardness for MCSP^A implies hardness for MKTP^A (Theorem 12).

A reduction that is not "oracle independent". Hirahara and Watanabe [14] observed that all of the then-known reductions to MCSP and MKTP were "oracle-independent", in the sense that, for any class $\mathcal{C}$ and reducibility $\leq_r$, all proofs that MCSP (or MKTP) is hard for $\mathcal{C}$ under $\leq_r$ also show that MCSP^A (MKTP^A) is also hard for $\mathcal{C}$. They showed that oracle-independent $\leq_{\mathrm{T}}^{\mathsf{P}}$-reductions cannot show hardness for any class larger than P.

This motivates the search for reductions that are *not* oracle-independent. We give a concrete example of a logspace-uniform $\leq_{\mathrm{ctt}}^{\mathsf{AC}^0}$ reduction that (under a plausible complexity assumption) reduces DET to MKTP. This is *not* an oracle independent reduction, since $\mathsf{MKTP}^{\mathsf{QBF}}$ is not hard for DET under this same class of reductions (Corollary 13).

A clearer picture of how hardness "evolves". It is instructive to contrast the evolution of the class of problems reducible to MKTP^A under different types of reductions, as A varies from very easy ($A = \emptyset$) to complex ($A = \mathsf{QBF}$). For this thought experiment, we assume the very plausible hypothesis that $\mathsf{DSPACE}(n) \not\subseteq \mathsf{io\text{-}SIZE}(2^{\epsilon n})$. Restrictions of QBF give a useful parameterization for the complexity of A. Consider A varying from being complete for each level of PH (that is, quantified Boolean formulas with $O(1)$ alternations between $\forall$ and $\exists$ quantifiers), to instances of QBF with $\log^* n$ alternations, then to $O(\log n)$ alternations etc.,

[1] Subsequent to our work, a stronger average-case lower bound against $\mathsf{AC}^0[p]$ was proved [13]. The techniques of [13] do not show how to reduce DET, or even smaller classes such as TC^0, to MKTP. Thus our work is incomparable to [13].

[2] Somewhat remarkably, Oliveira and Santhanam [20] have independently shown that MCSP and MKTP are hard for DET under non-uniform $\leq_{\mathrm{tt}}^{\mathsf{TC}^0}$ reductions. Their proof relies on self-reducibility properties of the determinant, whereas our proof relies on the fact that Graph Isomorphism is hard for DET [27]. Their results have the advantage that they apply to MCSP rather than merely to MKTP, but because it is not known whether $\mathsf{TC}^0 = \mathsf{P}$ they do not obtain unconditional lower bounds, as in Corollary 10.

through to $2^{\sqrt{\log n}}$ alternations, and until finally $A = \mathsf{QBF}$. Since $\mathsf{DSPACE}(n) \subseteq \mathsf{P}^A/\mathsf{poly}$, at some point in this evolution we have $\mathsf{DSPACE}(n) \subseteq \mathsf{io\text{-}SIZE}^A(2^{\epsilon n})$; it is plausible to assume that this doesn't happen until A has at least $\log n$ quantifier alternations, or more.

At all stages in this evolution $\mathsf{SZK} \subseteq \mathsf{BPP}^{\mathsf{MKTP}^A}$ [4], until at some point $\mathsf{BPP}^{\mathsf{MKTP}^A}$ expands to coincide with PSPACE [3]. Also, at all stages in this evolution $\mathsf{DET} \leq_{\mathrm{m}}^{\mathsf{NC}^0}$-reduces to MKTP^A (and even when $A = \mathsf{QBF}$ we do not know, for instance, if $\mathsf{NC}^3 \leq_{\mathrm{m}}^{\mathsf{NC}^0}$-reduces to MKTP^A). Thus these reductions behave "monotonically", in the sense that as the complexity of A increases, the class of problems reducible to MKTP^A does not shrink noticeably, and sometimes appears to grow markedly.

The situation is much more intriguing when we consider the *uniform* class of $\leq_{\mathrm{T}}^{\mathsf{AC}^0}$ reductions that arise from derandomizing the nonuniform $\leq_{\mathrm{m}}^{\mathsf{NC}^0}$ reductions from DET. At the start, when $A = \emptyset$, we have DET reducing to MKTP^A, and this is maintained until A becomes complex enough so that $\mathsf{DSPACE}(n) \subseteq \mathsf{io\text{-}SIZE}^A(2^{\epsilon n})$. At this point, not only does DET not reduce to MKTP^A, but neither does PARITY! (See Theorem 12.)

This helps place the results of [7] in the proper context. In [7] strong evidence was presented against $\mathsf{MCSP}^{\mathsf{QBF}}$ being hard for, say, P under $\leq_{\mathrm{m}}^{\mathsf{L}}$ reductions, and this was taken as indirect evidence that MCSP itself should not be hard for P, since $\mathsf{MCSP} \in \mathsf{NP}$ and thus is much "easier" than the PSPACE-complete problem $\mathsf{MCSP}^{\mathsf{QBF}}$. However, we expect that MCSP^A and MKTP^A should behave somewhat similarly to each other, and it *can* happen that a class can reduce to MKTP (Theorem 11) and *not* reduce to MKTP^A for a more powerful oracle A (Corollary 13).

Hardness of the Gap problem. Our new hardness results for MKTP^A share with earlier reductions the property that they hold even for "Gap" versions of the problem. That is, for some $\epsilon > 0$, the reduction works correctly for any solution to the promise problem with "yes" instances $\{(x, s) : \mathrm{KT}^A(x) \leq s\}$ and "no" instances $\{(x, s) : \mathrm{KT}^A(x) > s + |x|^\epsilon\}$. However, we do not know if they carry over to instances with a wider "gap" between the Yes and No instances; earlier hardness results such as those of [3, 9, 4, 25] hold for a much wider gap (such as with the Yes instances having $\mathrm{KT}(x) < |x|^\epsilon$, and the no instances with $\mathrm{KT}(x) \geq |x|$), and this is one reason why they applied both to MKTP and to MCSP. Thus there is interest in whether it is possible to reduce MCSP with small "gap" to MCSP with large "gap". If this were possible, then MCSP and MKTP would be interreducible in some sense.

Earlier work [7] had presented unconditional results, showing that "gap" versions of MCSP could not be hard for TC^0 under $\leq_{\mathrm{m}}^{\mathsf{AC}^0}$ reductions, unless those reductions had large "stretch" (mapping short inputs to long outputs). In [6], we show that BPP-Turing reductions among gap MCSP problems require large stretch, unless $\mathsf{MCSP} \in \mathsf{BPP}$.

Natural NP-intermediate Problems. In Section 3 we also consider gap MCSP problems where the "gap" is quite large (i.e., problems of approximating the minimum circuit size for a truth table of size n within a factor of $n^{1-o(1)}$). Problems of this sort are of interest, because of the role they play in the natural proofs framework of [23], if one is trying to prove circuit lower bounds of size $2^{o(n)}$. Our Theorem 6 shows that these problems are NP-intermediate in the sense that these do not lie in P/poly and are not NP-hard under P/poly reductions, under modest cryptographic assumptions (weaker than assuming that factoring or discrete log requires superpolynomial-size circuits, or assuming the existence of a one-way function). To the best of our knowledge, these problems are the first natural NP-intermediate problems under the existence of an arbitrary one-way function.

Our new insight on MCSP here is that, if the gap problems are NP-hard, then MCSP is

"strongly downward self-reducible": that is, any instance of MCSP of size n can be reduced to instances of size n^ϵ. In the past, many natural problems have been shown to be strongly downward self-reducible (see [8]); Our contribution is to show that MCSP also has such a property (under the assumption that the gap MCSP problems are NP-hard).

2 Preliminaries

We assume the reader is familiar with standard DTIME and DSPACE classes. We also occasionally refer to classes defined by time-bounded *alternating* Turing machines: $\mathsf{ATIME}(t(n))$, or by simultaneously bounding time and the number of alternations between existential and universal configurations: $\mathsf{ATIME\text{-}ALT}(t(n), a(n))$.

We refer the reader to the text by Vollmer [29] for background and more complete definitions of the standard circuit complexity complexity classes

$$\mathsf{NC}^0 \subsetneq \mathsf{AC}^0 \subsetneq \mathsf{AC}^0[p] \subsetneq \mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{P/poly},$$

as well as the standard complexity classes $\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE}$. Between L and P in this list, there is one more class that plays an important role for us: DET is the class of problems that are reducible to the problem of computing the determinant of integer matrices, by NC^1-Turing reductions.

This brings us to the topic of reducibility. Let $\mathcal{C}$ be either a class of functions or a class of circuits. We say that $A \leq_{\mathrm{m}}^{\mathcal{C}} B$ if there is a function $f \in \mathcal{C}$ (or f computed by a circuit family in $\mathcal{C}$, respectively) such that $x \in A$ iff $f(x) \in B$. We will make use of $\leq_{\mathrm{m}}^{\mathsf{L}}, \leq_{\mathrm{m}}^{\mathsf{TC}^0}, \leq_{\mathrm{m}}^{\mathsf{AC}^0}$ and $\leq_{\mathrm{m}}^{\mathsf{NC}^0}$ reducibility. The more powerful notion of Turing reducibility also plays an important role in this work. Here, $\mathcal{C}$ is a complexity class that admits a characterization in terms of Turing machines or circuits, which can be augmented with an "oracle" mechanism, either by providing a "query tape" or "oracle gates". We say that $A \leq_{\mathrm{T}}^{\mathcal{C}} B$ if there is a oracle machine in $\mathcal{C}$ (or a family of oracle circuits in $\mathcal{C}$) accepting A, when given oracle B. We make use of $\leq_{\mathrm{T}}^{\mathsf{P/poly}}, \leq_{\mathrm{T}}^{\mathsf{BPP}}, \leq_{\mathrm{T}}^{\mathsf{P}}, \leq_{\mathrm{T}}^{\mathsf{L}}$ and $\leq_{\mathrm{T}}^{\mathsf{AC}^0}$ reducibility; instead of writing $A \leq_{\mathrm{T}}^{\mathsf{P/poly}} B$ or $A \leq_{\mathrm{T}}^{\mathsf{BPP}} B$, we will more frequently write $A \in \mathsf{P}^B/\mathsf{poly}$ or $A \in \mathsf{BPP}^B$. Turing reductions that are "nonadaptive" – in the sense that the list of queries that are posed on input x does not depend on the answers provided by the oracle – are called *truth-table reductions*. We make use of $\leq_{\mathrm{tt}}^{\mathsf{AC}^0}$ and $\leq_{\mathrm{tt}}^{\mathsf{TC}^0}$ reducibility.

Kabanets and Cai [16] sparked renewed interest in MCSP and highlighted connections between MCSP and more recent progress in derandomization. They introduced a class of reductions to MCSP, which they called *natural reductions*. Recall that instances of MCSP are of the form (T, s) where s is a "size parameter". A $\leq_{\mathrm{m}}^{\mathsf{P}}$ reduction f is called *natural* if $f(x)$ is of the form $f(x) = (f_1(x), f_2(|x|))$. That is, the "size parameter" is the same, for all inputs x of the same length.

Whenever circuit families are discussed (either when defining complexity classes, or reducibilities), one needs to deal with the issue of *uniformity*. For example, the class AC^0 (corresponding to families $\{C_n : n \in \mathbb{N}\}$ of unbounded fan-in AND, OR, and NOT gates having size $n^{O(1)}$ and depth $O(1)$) comes in various flavors, depending on the complexity of computing the mapping $1^n \mapsto C_n$. When this is computable in polynomial time (or logarithmic space), then one obtains P-uniform AC^0 (logspace-uniform AC^0, respectively). If no restriction at all is imposed, then one obtains non-uniform AC^0. As discussed in [29], the more restrictive notion of Dlogtime-uniform AC^0 is frequently considered to be the "right" notion of uniformity to use when discussing small complexity classes such as $\mathsf{AC}^0, \mathsf{AC}^0[p]$ and TC^0. If these classes are mentioned with no explicit mention of uniformity,

then Dlogtime-uniformity is intended. For uniform NC^1 the situation is somewhat more complicated, as discussed in [29]; there is wide agreement that the "correct" definition coincides with $\mathsf{ATIME}(O(\log n))$.

There are many ways to define time-bounded Kolmogorov complexity. The definition $\mathrm{KT}(x)$ was proposed in [3], and has the advantage that it is polynomially-related to circuit size (when a string x is viewed as the truth-table of a function). $\mathrm{KT}(x)$ is the minimum, over all d and t, of $|d| + t$, such that the universal Turing machine U, on input (d, i, b) can determine in time t if the i-th bit of x is b. (More formal definitions can be found in [3].)

A *promise problem* consists of a pair of disjoint subsets (Y, N). A language A is a *solution* to the promise problem (Y, N) if $Y \subseteq A \subseteq \overline{N}$. A language B reduces to a promise problem via a type of reducibility $\leq_r$ if $B \leq_r A$ for *every* set A that is a solution to the promise problem.

3 GapMCSP

In this section, we consider the "gap" versions of MCSP and MKTP. We focus primarily on MCSP, and for simplicity of exposition we consider the "size" of a circuit to be the number of AND and OR gates of fan-in two. (NOT gates are "free"). The arguments can be adjusted to consider other circuit models and other reasonable measures of "size" as well. Given a truth-table T, let $\mathrm{CC}(T)$ be the size of the smallest circuit computing T, using this notion of "size".

▶ **Definition 1.** For any function $\epsilon\colon \mathbb{N} \to (0,1)$, let $\mathrm{Gap}_\epsilon\mathrm{MCSP}$ be the approximation problem that, given a truth-table T, asks for outputting a value $f(T) \in \mathbb{N}$ such that

$$\mathrm{CC}(T) \leq f(T) \leq |T|^{1-\epsilon(|T|)} \cdot \mathrm{CC}(T).$$

Note that this approximation problem can be formulated as the following promise problem. (See also [11] for similar comments.)

▶ **Fact 2.** $\mathrm{Gap}_\epsilon\mathrm{MCSP}$ *is polynomial-time Turing equivalent to the following promise problem* (Y, N)*:*

$$Y := \{\, (T,s) \mid \mathrm{CC}(T) < s/|T|^{1-\epsilon(|T|)} \,\},$$
$$N := \{\, (T,s) \mid \mathrm{CC}(T) > s+1 \,\},$$

where T *is a truth-table and* $s \in \mathbb{N}$*.*

Note that $\mathrm{Gap}_\epsilon\mathrm{MCSP}$ becomes easier when ϵ becomes smaller. If $\epsilon(n) = o(1)$, then (using the promise problem formulation) it is easy to see that $\mathrm{Gap}_\epsilon\mathrm{MCSP}$ has a solution in $\mathsf{DTIME}(2^{n^{o(1)}})$, since the Yes instances have witnesses of length $|T|^{o(1)}$. However, it is worth emphasizing that, even when $\epsilon(n) = o(1)$, $\mathrm{Gap}_\epsilon\mathrm{MCSP}$ is a canonical example of a combinatorial property that is useful in proving circuit size lower bounds of size $2^{o(n)}$, in the sense of [23]. Thus it is of interest that MCSP cannot reduce to $\mathrm{Gap}_\epsilon\mathrm{MCSP}$ in this regime under very general notions of reducibility, unless MCSP itself is easy.

▶ **Theorem 3.** *For any polynomial-time-computable nonincreasing* $\epsilon(n) = o(1)$*, if* $\mathsf{MCSP} \in \mathsf{BPP}^{\mathrm{Gap}_\epsilon\mathrm{MCSP}}$ *then* $\mathsf{MCSP} \in \mathsf{BPP}$*.*

A new idea is that $\mathrm{Gap}_\epsilon\mathrm{MCSP}$ is "strongly downward self-reducible." We will show that any $\mathrm{Gap}_\epsilon\mathrm{MCSP}$ instance of length n is reducible to $n^{1-\epsilon}$ MCSP instances of length n^ϵ. To this end, we will exploit the following simple fact.

▶ **Lemma 4.** *For a function $f\colon \{0,1\}^n \to \{0,1\}$, a string $x \in \{0,1\}^k$ and $k \in \mathbb{N}$, let $f_x\colon \{0,1\}^{n-k} \to \{0,1\}$ be a function defined as $f_x(y) := f(x,y)$. Then, the following holds:*

$$\max_{x\in\{0,1\}^k} \mathrm{CC}(f_x) \leq \mathrm{CC}(f) \leq 2^k \cdot \left(\max_{x\in\{0,1\}^k} \mathrm{CC}(f_x) + 3 \right),$$

(In other words, $\max_{x\in\{0,1\}^k} \mathrm{CC}(f_x)$ gives an approximation of $\mathrm{CC}(f)$ within a factor of 2^k.)

Proof of Theorem 3. Let M be an oracle BPP Turing machine which reduces MCSP to Gap_ϵMCSP. Let $|T|^c$ be an upper bound for the running time of M, given a truth-table T, and let $|T| = 2^n$.

We recursively compute the circuit complexity of T by the following procedure: Run M on input T. If M makes a query S to the Gap_ϵMCSP oracle, then divide S into consecutive substrings $S_1, \cdots, S_{2^k}$ of length $|S| \cdot 2^{-k}$ such that $S_1 \cdot S_2 \cdots S_{2^k} = S$ (where k is a parameter, chosen later, that depends on $|S|$), and compute the circuit complexity of each S_i recursively for each $i \in [2^k]$. Then continue the simulation of M, using the value $2^k \cdot \left(\max_{i\in[2^k]} \mathrm{CC}(S_i) + 3\right)$ as an approximation to $\mathrm{CC}(S)$.

We claim that the procedure above gives the correct answer. For simplicity, let us first assume that the machine M has zero error probability. It suffices to claim that the simulation of M is correct in the sense that every query of M is answered with a value that satisfies the approximation criteria of Gap_ϵMCSP. Suppose that M makes a query S. By the assumption on the running time of M, we have $|S| \leq |T|^c = 2^{nc}$. By Lemma 4, we have

$$\mathrm{CC}(S) \leq 2^k \cdot \left(\max_{i\in[2^k]} \mathrm{CC}(S_i) + 3 \right) \leq 2^k \cdot (\mathrm{CC}(S) + 3).$$

In particular, the estimated value satisfies the promise of Gap_ϵMCSP if $2^k \cdot (\mathrm{CC}(S) + 3) \leq |S|^{1-\epsilon(|S|)} \cdot \mathrm{CC}(S)$. Since we may assume without loss of generality that $\mathrm{CC}(S) \geq 3$, it suffices to make sure that $2^{k+1} \cdot \mathrm{CC}(S) \leq |S|^{1-\epsilon(|S|)} \cdot \mathrm{CC}(S)$. Let $|S| = 2^m$. Then, in order to satisfy $k + 1 \leq (1 - \epsilon(|S|)) \cdot m$, let us define $k := (1 - \epsilon(|S|)) \cdot m - 1$. For this particular choice of k, the estimated value $2^k \cdot \left(\max_{i\in[2^k]} \mathrm{CC}(S_i) + 3\right)$ of the circuit complexity of S satisfies the promise of Gap_ϵMCSP, which implies that the reduction M computes the correct answer for MCSP.

Now we analyze the time complexity of the algorithm. Each recursive step makes at most 2^{2cn} many recursive calls, because there are potentially 2^{cn} many queries S of M, each of which may produce at most $2^k \leq 2^{cn}$ recursive calls. The length of each truth-table S_i that arises in one of the recursive calls is $|S_i| = |S| \cdot 2^{-k} = 2^{m-k} = 2^{\epsilon(|S|)\cdot m+1}$. We claim that $|S_i| \leq 2^{1+(n/2)}$ holds for sufficiently large n. Let us take n to be large enough so that $\epsilon(2^{n/2}) \leq 1/2c$. If $m \geq n/2$, then $|S_i| \leq 2^{\epsilon(2^m)\cdot m+1} \leq 2^{\epsilon(2^{n/2})\cdot cn+1} \leq 2^{1+(n/2)}$. Otherwise, since $m \leq n/2$ and $\epsilon(|S|) < 1$, we obtain $|S_i| \leq 2^{\epsilon(|S|)\cdot m+1} \leq 2^{1+(n/2)}$. Therefore, on inputs of length 2^n, each recursive call produces instances of length at most $2^{1+(n/2)}$. The overall time complexity can be estimated as $2^{c'n} \cdot 2^{c'n/2} \cdot 2^{c'n/4} \cdots = 2^{2c'n}$ for some constant c' (say, $c' = 3c$), which is a polynomial in the input length 2^n.

We note that the analysis above works even for *randomized* reductions that may err with exponentially small probability. Since we have proved that the algorithm runs in polynomial time, the probability that the algorithm makes an error is at most a polynomial times an exponentially small probability, which is still exponentially small probability (by the union bound). ◀

▶ Remark. If we drop the assumption that $\epsilon(n)$ be computable, then the proof of Theorem 3 still shows that if $\mathsf{MCSP} \in \mathsf{P}^{\mathrm{Gap}_\epsilon\mathrm{MCSP}}/\mathsf{poly}$ then $\mathsf{MCSP} \in \mathsf{P}/\mathsf{poly}$.

▶ **Corollary 5.** *Let* $\epsilon(n) = o(1)$. *If* $\mathrm{Gap}_\epsilon\mathsf{MCSP}$ *has no solution in* $\mathsf{P/poly}$ *then* $\mathrm{Gap}_\epsilon\mathsf{MCSP}$ *is not hard for* NP *(or even for* MCSP*) under* $\leq_{\mathrm{T}}^{\mathsf{P/poly}}$ *reductions, and is thus* NP*-intermediate.*

Proof. This is immediate from the preceding remark. If $\mathsf{MCSP} \in \mathsf{P}^{\mathrm{Gap}_\epsilon\mathsf{MCSP}}/\mathsf{poly}$ then $\mathsf{MCSP} \in \mathsf{P/poly}$, which in turn implies that $\mathrm{Gap}_\epsilon\mathsf{MCSP}$ has a solution in $\mathsf{P/poly}$. ◀

In what follows, we show that the assumption of Corollary 5 is true under very modest cryptographic assumptions. It is known that, for any constant $\epsilon > 0$, $\mathrm{Gap}_\epsilon\mathsf{MCSP}$ is SZK-hard under $\leq_{\mathrm{T}}^{\mathsf{P/poly}}$ reductions [4]. Here, we show that if SZK is not in $\mathsf{P/poly}$, then for some $\epsilon(n) = o(1)$, $\mathrm{Gap}_\epsilon\mathsf{MCSP}$ has no solution in $\mathsf{P/poly}$. In fact, we can prove something *stronger*: If auxiliary-input one-way functions exist, then $\mathrm{Gap}_\epsilon\mathsf{MCSP}$ is not in $\mathsf{P/poly}$. We now describe auxiliary-input one-way functions.

Usually, the existence of cryptographically-secure one-way functions is considered to be essential for meaningful cryptography. That is, one requires a function f computed in polynomial time such that, for any algorithm A computed by polynomial-sized circuits, $\Pr_x[f(A(f(x))) = f(x)] = 1/n^{\omega(1)}$ where x is chosen uniformly at random from $\{0,1\}^n$. A weaker notion that has been studied in connection with SZK goes by the name *auxiliary-input one-way functions*. This is an indexed family of functions $f_y(x) = F(y,x)$, where $|x| = p(|y|)$ for some polynomial p, and F is computable in time polynomial in $|y|$, such that for some infinite set I, for any algorithm[3] A computed by polynomial-sized circuits, for all $y \in I$, $\Pr_x[f_y(A(f_y(x))) = f_y(x)] = 1/n^{\omega(1)}$ where $n = |y|$ and x is chosen uniformly at random from $\{0,1\}^{p(n)}$. It is known that there are promise problems in SZK that have no solution in $\mathsf{P/poly}$ only if auxiliary-input one-way functions exist. (This is due to [22]; a good exposition can be found in [28, Theorems 7.1 & 7.5], based on earlier work of [21].)

▶ **Theorem 6.** *If auxiliary-input one-way functions exist, then there is a function* $\epsilon(n) = o(1)$ *such that* $\mathrm{Gap}_\epsilon\mathsf{MCSP}$ *is* NP*-intermediate. (Namely,* $\mathrm{Gap}_\epsilon\mathsf{MCSP}$ *has no solution in* $\mathsf{P/poly}$ *and* $\mathrm{Gap}_\epsilon\mathsf{MCSP}$ *is not* NP*-hard under* $\leq_{\mathrm{T}}^{\mathsf{P/poly}}$ *reductions.)*

▶ Remark. In particular, either one of the following implies that some $\mathrm{Gap}_\epsilon\mathsf{MCSP}$ is NP-intermediate, since each implies the existence of auxiliary-input one-way functions:

1. the existence of cryptographically-secure one-way functions.
2. SZK is not in $\mathsf{P/poly}$.

4 Hardness for DET

In this section, we give some of our main contributions. We show that MKTP is hard for DET under $\leq_{\mathrm{m}}^{\mathsf{NC}^0}$ reductions (Theorem 9); prior to this, no variant of MCSP was known to be hard for any complexity class under any type of many-one reducibility. The $\leq_{\mathrm{m}}^{\mathsf{NC}^0}$ reduction that we present is nonuniform; we show that hardness under *uniform* reductions is related to lower bounds in circuit complexity, and in some cases we show that circuit lower bounds are *equivalent* to hardness results under uniform notions of reducibility (Theorem 12). These techniques yield the first results relating the complexity of MCSP^A and MKTP^A problems.

Here is the outline of this section. We will build on a randomized reduction of Allender, Grochow and Moore [5]: They showed that there is a ZPP reduction from the rigid[4] graph

[3] We have chosen to define one-way functions in terms of security against non-uniform adversaries. It is also common to use the weaker notion of security against probabilistic polynomial-time adversaries, as in [28].

[4] A graph is *rigid* if it has no nontrivial automorphisms.

isomorphism problem to MKTP. Here we show that the reduction is in fact an AC^0 reduction (Corollary 8). Combining Torán's AC^0 reduction [27] from DET to the rigid graph isomorphism as well as the Gap theorem [2], we will show $\mathsf{DET} \leq_{\mathrm{m}}^{\mathsf{NC}^0} \mathsf{MKTP}$ (Theorem 9).

To show that circuit size lower bounds are equivalent to hardness under uniform AC^0 reductions, we will derandomize the reduction of [5] (Theorem 11). To this end, we give an AC^0 reduction f from the rigid graph isomorphism problem to MKTP and an "encoder" e that encodes random binary strings into a random permutation in Lemma 7 below.

▶ **Lemma 7.** *Let A be any oracle. There is a function f computable in Dlogtime-uniform AC^0 and a function e computable in Dlogtime-uniform TC^0 such that, for any two rigid graphs G, H with n vertices:*

- $\Pr_r[f(G, H, e(r)) \notin \mathsf{MKTP}^A] > 1 - \frac{1}{2^{4n^2}}$ *if $G \not\equiv H$, and*
- $\Pr_r[f(G, H, e(r)) \in \mathsf{MKTP}^A] = 1$ *if $G \equiv H$.*

▶ **Corollary 8.** *Let A be any oracle. The rigid graph isomorphism problem is reducible to MKTP^A via a non-uniform $\leq_{\mathrm{m}}^{\mathsf{AC}^0}$ reduction.*

Proof. A standard counting argument shows that there is a value of $e(r)$ that can be hardwired into the reduction of Lemma 7 that works correctly for all pairs (G, H) of n-vertex graphs. (Note that the input length is $2n^2$, and the error probability is at most $1/2^{4n^2}$.) ◀

▶ **Theorem 9.** *Let A be any oracle.* DET *is reducible to MKTP^A via a non-uniform $\leq_{\mathrm{m}}^{\mathsf{NC}^0}$ reduction. Furthermore, this reduction is "natural" in the sense of [16].*

Proof. Since DET is closed under $\leq_{\mathrm{m}}^{\mathsf{TC}^0}$ reductions, it suffices to show that MKTP^A is hard under $\leq_{\mathrm{m}}^{\mathsf{AC}^0}$ reductions, and then appeal to the "Gap" theorem of [2], to obtain hardness under $\leq_{\mathrm{m}}^{\mathsf{NC}^0}$ reducibility. Torán [27] shows that DET is AC^0-reducible to GI, and the proofs of Theorem 5.3 and Corollary 5.4 of [27] show that DET is AC^0-reducible to GI via a reduction that outputs only pairs of rigid graphs. Composing this reduction with the non-uniform AC^0 reduction given by Corollary 8 completes the argument. (Since DET is closed under complement, there is also a non-uniform $\leq_{\mathrm{m}}^{\mathsf{AC}^0}$ reduction to the complement of MKTP^A.)

Since the same θ is used for all inputs of the same length, the reduction is "natural". ◀

The lower bounds of Razborov and Smolensky [24, 26] yield the following corollary:

▶ **Corollary 10.** *MKTP^A is not in $\mathsf{AC}^0[p]$ for any oracle A and any prime p.*

(An alternate proof of this circuit lower bound can be obtained by applying the pseudorandom generator of [10] that has sublinear stretch and is secure against $\mathsf{AC}^0[p]$. Neither argument seems easy to extend, to provide a lower bound for MCSP.)

One may wonder if the non-uniform reduction can be made uniform under a derandomization hypothesis. We do not know how to obtain a uniform $\leq_{\mathrm{m}}^{\mathsf{AC}^0}$ reduction, but we can come close, if A is not too complex. Recall the definition of ctt-reductions: $B \leq_{\mathrm{ctt}}^{\mathcal{C}} C$ if there is a function $f \in \mathcal{C}$ with the property that $f(x)$ is a list $f(x) = (y_1, \ldots, y_m)$, and $x \in B$ if and only if $y_j \in C$ for all j. Furthermore, we say that f is a *natural* logspace-uniform $\leq_{\mathrm{ctt}}^{\mathsf{AC}^0}$-reduction to MKTP if each query y_j has the same length (and this length depends only on $|x|$), and also each y_j is of the form (z_j, θ) where the threshold θ depends only on $|x|$.

The following theorem can be viewed as a "partial converse" to results of [19, 7], which say that problems in $\mathsf{LTH} \subseteq \mathsf{E}$ require exponential size circuits if MCSP or MKTP is hard for TC^0 under Dlogtime-uniform $\leq_{\mathrm{m}}^{\mathsf{AC}^0}$ reductions. That is, the earlier results show that very uniform hardness results imply circuit lower bounds, whereas the next theorem shows that somewhat stronger circuit lower bounds imply uniform hardness results (for a less-restrictive

notion of uniformity, but hardness for a larger class). Later on, in Theorem 12, we present a related condition on reductions to MKTP^A that is *equivalent* to circuit lower bounds.

▶ **Theorem 11.** *Let A be any oracle. If there is some $\epsilon > 0$ such that $\mathsf{DSPACE}(n) \not\subseteq \mathsf{io\text{-}SIZE}^{\mathsf{MKTP}^A}(2^{\epsilon n})$, then every language in DET reduces to MKTP^A via a natural logspace-uniform $\leq^{\mathsf{AC}^0}_{\mathrm{ctt}}$-reduction.*

Proof. Let $B \in \mathsf{DET}$. Thus there is an AC^0 reduction g reducing B to the Rigid Graph Isomorphism Problem [27]. Consider the following family of statistical tests $T_x(r)$, indexed by strings x:

> On input r:
> Compute $z = f(g(x), e(r))$, where $f(G, H, e(r))$ is the function from Lemma 7. Accept iff ($x \in B$ iff $z \in \mathsf{MKTP}^A$).

Since $B \in \mathsf{DET} \subseteq \mathsf{P}$, the test $T_x(r)$ has a polynomial-size circuit with one MKTP^A oracle gate. (In fact, the statistical test is an NC^2 circuit with one oracle gate.) If $x \in B$, then T_x accepts *every* string r, whereas if $x \notin B$, T_x accepts most strings r.

Klivans and van Melkebeek [17] (building on the work of Impagliazzo and Wigderson [15]) show that, if $\mathsf{DSPACE}(n)$ requires exponential-size circuits from a given class $\mathcal{C}$, then there is a hitting set generator computable in logspace that hits all large sets computable by circuits from $\mathcal{C}$ having size n^k. In particular, under the given assumption, there is a function h computable in logspace such that $h(0^n) = (r_1, r_2, \ldots, r_{n^c})$ with the property that, for all strings x of length n, there is an element of $h(0^n)$ that is accepted by T_x.

Now consider the logspace-uniform AC^0 oracle circuit family, where the circuit for inputs of length n has the strings $e(h(0^n)) = (e(r_1), e(r_2), \ldots, e(r_{n^c}))$ hardwired into it. The circuit computes the queries $f(g(x), e(r_i))$ for $1 \leq i \leq n^c$, and accepts if, for all i, $f(g(x), e(r_i)) \in \mathsf{MKTP}^A$. Note that if $x \notin B$, then one of the r_i is accepted by T_x, which means that $f(g(x), e(r_i)) \notin \mathsf{MKTP}^A$; if $x \in B$, then $f(g(x), e(r_i)) \in \mathsf{MKTP}^A$ for all i. This establishes that the reduction is correct. ◀

Theorem 11 deals with the oracle problem MKTP^A, but the most interesting case is the case where $A = \emptyset$. The hypothesis is false when $A = \mathsf{QBF}$, since the KT^A measure is essentially the same as the KS measure studied in [3], where it is shown that $\mathsf{PSPACE} = \mathsf{ZPP}^{R_{\mathrm{KS}}}$, and thus has polynomial-size $\mathsf{MKTP}^{\mathsf{QBF}}$-circuits. Strikingly, not only is the hypothesis false in this case – but the conclusion is false as well. (See Corollary 13.)

For certain oracles (and we discuss below how broad this class of oracles is), the existence of uniform reductions is *equivalent to* certain circuit lower bounds.

▶ **Theorem 12.** *Let $\mathsf{MKTP}^A \in \mathsf{P}^A/\mathsf{poly}$. Then the following are equivalent:*

- PARITY *reduces to* MKTP^A *via a natural logspace-uniform* $\leq^{\mathsf{AC}^0}_{\mathrm{ctt}}$*-reduction.*
- *For some* $\epsilon > 0$, $\mathsf{DSPACE}(n) \not\subseteq \mathsf{io\text{-}SIZE}^A(2^{\epsilon n})$.
- *For some* $\epsilon > 0$, $\mathsf{DSPACE}(n) \not\subseteq \mathsf{io\text{-}SIZE}^{\mathsf{MKTP}^A}(2^{\epsilon n})$.
- DET *reduces to* MKTP^A *via a natural logspace-uniform* $\leq^{\mathsf{AC}^0}_{\mathrm{ctt}}$*-reduction.*

Furthermore, if PARITY *reduces to* MCSP^A *via a natural logspace-uniform* $\leq^{\mathsf{AC}^0}_{\mathrm{ctt}}$*-reduction, then all of the above hold.*

Proof. First, we show that the first condition implies the second.

Let $\{C_n : n \in \mathbb{N}\}$ be a logspace-uniform family of oracle circuits computing PARITY, consisting of AC^0 circuitry feeding into oracle gates, which in turn are connected to an AND gate as the output gate. Let the oracle gates in C_n be $g_1, g_2, \ldots, g_{n^c}$. On any input string x,

let the value fed into gate g_i on input x be $(q_i(x), \theta)$, and recall that, since the reduction is natural, the threshold θ depends only on n, and thus it is a constant in C_n.

Now, we appeal to [7, Claim 3.11], and conclude that each $\mathsf{MKTP}^{\mathsf{QBF}}$ oracle gate can be replaced by a DNF formula of size at most $n^{O(1)}2^{O(\theta^2 \log \theta)}$. Inserting these DNF formulae into C_n (in place of each oracle gate) results in a circuit of size $n^{O(1)}2^{O(\theta^2 \log \theta)}$ computing PARITY. Let the depth of this circuit be some constant d. It follows from [12] that $n^{O(1)}2^{O(\theta^2 \log \theta)} \geq 2^{\Omega(n^{1/(d-1)})}$, and hence that $\theta \geq n^{1/4d}$.

Note that all of the oracle gates g_i must output 1 on input $0^{n-1}1$, and one of the oracle gates g_{i_0} must output 0 on input 0^n. Thus we have $\mathrm{KT}^A(q_{i_0}(0^n)) \geq \theta \geq n^{1/4d}$. It follows from [3, Theorem 11] that the function with truth-table $q_{i_0}(0^n)$ has no circuit (with oracle gates for A) of size less than $(\mathrm{KT}^A(q_{i_0}(0^n)))^{1/3} \geq \theta^{1/3} \geq n^{1/12d}$.

Note that, in order to compute the j-th bit of some query $q_i(0^n)$, it suffices to evaluate a logspace-uniform AC^0 circuit where all of the input bits are 0. Since this computation can be done in logspace on input $(0^n 1^i 0^j)$, note that the language $H = \{(n, i, j) : \text{the } j\text{-th bit of query } q_i(0^n) \text{ is } 1\}$ is in linear space. Let $m = |(n, i, j)|$, and let $s(m)$ be the size of the smallest circuit D_m computing H for inputs of length m. Hardwire the bits for n and also set the bits for i to i_0. The resulting circuit on $|j| < m$ bits computes the function given by $q_{i_0}(0^n)$, and it was observed above that this circuit has size at least $n^{1/12d} \geq 2^{m/12d}$.

This establishes the first implication. (Note also that a similar argument yields the same conclusion from the assumption that PARITY reduces to MCSP^A via a natural logspace-uniform $\leq^{\mathsf{AC}^0}_{\mathrm{ctt}}$-reduction.)

The assumption that $\mathsf{MKTP}^A \in \mathsf{P}^A/\mathsf{poly}$ suffices to show that the second condition implies the third. More formally, we'll consider the contrapositive. Assume that $\mathsf{DSPACE}(n) \subseteq \mathsf{io\text{-}SIZE}^{\mathsf{MKTP}^A}(2^{\epsilon n})$ for every $\epsilon > 0$. An oracle gate for MKTP^A on inputs of size m can be replaced by a circuit (with oracle gates for A) of size m^c for some constant c. Carrying out this substitution in a circuit (with oracle gates for MKTP^A) of size $2^{\epsilon n}$ yields a circuit of size at most $2^{\epsilon n} + 2^{\epsilon n}(2^{\epsilon n})^c$.

Let $\delta > 0$. Then we can pick ϵ so that $2^{\epsilon n} + 2^{\epsilon n}(2^{\epsilon n})^c < 2^{\delta n}$, thereby establishing that $\mathsf{DSPACE}(n) \subseteq \mathsf{io\text{-}SIZE}^A(2^{\delta n})$ for every $\delta > 0$. This establishes the second implication.

The 3rd condition implies the 4th by Theorem 11. The 4th obviously implies the 1st. ◀

To the best of our knowledge, this is the first theorem that has given conditions where the existence of a reduction to MCSP^A implies the existence of a reduction to MKTP^A. We know of no instance where the implication goes in the opposite direction.

At this point, we should consider the class of oracles for which Theorem 12 applies. That is, what is the set of oracles A for which $\mathsf{MKTP}^A \in \mathsf{P}^A/\mathsf{poly}$? First, we observe that this condition holds for any PSPACE-complete set, which yields the following corollary:

▶ **Corollary 13.** PARITY *does not reduce to either* $\mathsf{MKTP}^{\mathsf{QBF}}$ *or* $\mathsf{MCSP}^{\mathsf{QBF}}$ *via a natural logspace-uniform* $\leq^{\mathsf{AC}^0}_{\mathrm{ctt}}$*-reduction.*

Another example is $A = \{(M, x, 1^m) : M$ is an alternating Turing machine that accepts x, and runs in time at most m and makes at most $\log m$ alternations$\}$. A is complete for the class $\mathsf{ATIME\text{-}ALT}(n^{O(1)}, O(\log n))$ under $\leq^{\mathsf{AC}^0}_{\mathrm{m}}$ reductions. Note that $\mathsf{MKTP}^A \in \mathsf{ATIME\text{-}ALT}(n^{O(1)}, O(\log n))$, and thus $\mathsf{MKTP}^A \in \mathsf{P}^A$. (Other examples can easily be created in this way, using an even smaller number of alternations. Note that, for this oracle A, it seems plausible that all four conditions in Theorem 12 hold.

Nonetheless, we grant that this seems to be a strong condition to place upon the oracle A – and it has even stronger consequences than are listed in Theorem 12. For instance, note that the proof that the first condition in Theorem 12 implies the second relies only on the

fact that PARITY requires large AC^0 circuits. Thus, an identical proof shows that these four conditions are also equivalent to the condition that PARITY is reducible to MKTP^A via a natural ctt-reduction where the queries are computed by logspace-uniform $\mathsf{AC}^0[7]$ circuits. (One can substitute any other problem and class of mod circuits, where an exponential lower bound follows from [24, 26].) In fact, as in [7, Lemma 3.10] we can apply random restrictions in a logspace-uniform way (as described in [1]) and obtain a reduction from PARITY to MKTP^A where the queries are computed by logspace-uniform NC^0 circuits! That is, for example, MAJORITY is reducible to MKTP^A via reductions of this sort computed by logspace-uniform $\mathsf{AC}^0[3]$ circuits iff PARITY is reducible to the same set via reductions where the queries are computed by logspace-uniform NC^0 circuits. We find these implications to be surprising. The "gap" phenomenon that was described in [2] (showing that completeness under one class of reductions is equivalent to completeness under a more restrictive class of reductions) had not previously been observed to apply to $\mathsf{AC}^0[p]$ reducibility.

We want to highlight some contrasts between Theorem 11 and Corollary 13. $\mathsf{MKTP}^{\mathsf{QBF}}$ is hard for PSPACE under ZPP-Turing reductions [3], whereas MKTP is in NP. Thus $\mathsf{MKTP}^{\mathsf{QBF}}$ appears to be much harder than MKTP. Yet, under a plausible hypothesis, MKTP is hard for a well-studied subclass of P under a type of reducibility, where the "harder" problem $\mathsf{MKTP}^{\mathsf{QBF}}$ cannot even be used as an oracle for PARITY under this same reducibility.

In other words, the (conditional) natural logspace-uniform $\leq^{\mathsf{AC}^0}_{\mathrm{ctt}}$ reductions from problems in DET to MKTP given in Theorem 11 are not "oracle independent" in the sense of [14]. Prior to this work, there had been no reduction to MCSP or MKTP that did not work for every MCSP^A or MKTP^A, respectively.

Prior to this work, it appears that there was no evidence for any variant of MCSP or MKTP being hard for a reasonable complexity class under $\leq^{\mathsf{L}}_{\mathrm{T}}$ reductions. All prior reductions (such as those in [4, 3, 5]) had been probabilistic and/or non-uniform, or (even under derandomization hypotheses) seemed difficult to implement in NC. We had viewed the results of [7] as providing evidence that none of these variants would be hard for P under, say, logspace reducibility. Now, we are no longer sure what to expect.

5 Conclusions and Open Questions

Conclusions. At a high level, we have advanced our understanding about MCSP and MKTP in the following two respects:

1. On one hand, under a very weak cryptographic assumption, the problem of approximating MCSP or MKTP is indeed NP-intermediate under *general* types of reductions when the approximation factor is quite *huge*. This complements the work of [19] for very *restricted* reductions.
2. On the other hand, if the gap is *small*, MKTP is DET-hard under nonuniform NC^0 reductions (contrary to previous expectations). This suggests that nonuniform reductions are crucial to understanding hardness of MCSP. While there are many results showing that NP-hardness of MCSP under *uniform* reductions is as difficult as proving circuit lower bounds, can one show that MCSP is NP-hard under P/poly reductions (without proving circuit lower bounds)?

Open Questions. It should be possible to prove unconditionally that MCSP is not in $\mathsf{AC}^0[2]$; we conjecture that the hardness results we give for MKTP hold also for MCSP.

We suspect that it should be possible to prove more general results of the form "If MCSP^A is hard for class $\mathcal{C}$, then so is MKTP^A". We view Theorem 12 to be just a first step in this

direction. One way to prove such a result would be to show that MCSP^A reduces to MKTP^A, but (with a few exceptions such as $A = \mathsf{QBF}$) no such reduction is known. Of course, the case $A = \emptyset$ is the most interesting case.

Is MKTP hard for P? Or for some class between DET and P? Is it more than a coincidence that DET arises both in this investigation of MKTP and in the work of [20] on MCSP?

Is there evidence that Gap_ϵMCSP has intermediate complexity when ϵ is a fixed constant, similar to the evidence that we present for the case when $\epsilon(n) = o(1)$?

Acknowledgments. We thank Ryan Williams, Rahul Santhanam, Salil Vadhan, Marina Knittel, and Prashant Nalini Vasudevan for helpful discussions.

References

1. Manindra Agrawal. The isomorphism conjecture for constant depth reductions. *Journal of Computer and System Sciences*, 77(1):3–13, 2011. doi:10.1145/28395.28404.
2. Manindra Agrawal, Eric Allender, and Steven Rudich. Reductions in circuit complexity: An isomorphism theorem and a gap theorem. *Journal of Computer and System Sciences*, 57(2):127–143, 1998. doi:10.1006/jcss.1998.1583.
3. Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35:1467–1493, 2006. doi:10.1137/050628994.
4. Eric Allender and Bireswar Das. Zero knowledge and circuit minimization. *Information and Computation*, 2017. to appear. doi:10.1016/j.ic.2017.04.004.
5. Eric Allender, Joshua Grochow, and Cristopher Moore. Graph isomorphism and circuit size. Technical Report TR15-162, Electronic Colloquium on Computational Complexity, 2015. URL: https://eccc.weizmann.ac.il/report/2015/162/.
6. Eric Allender and Shuichi Hirahara. New insights on the (non)-hardness of circuit minimization and related problems. Technical Report TR17-073, Electronic Colloquium on Computational Complexity, 2017. URL: https://eccc.weizmann.ac.il/report/2017/073/.
7. Eric Allender, Dhiraj Holden, and Valentine Kabanets. The minimum oracle circuit size problem. *Computational Complexity*, 26(2):469–496, 2017. doi:10.1007/s00037-016-0124-0.
8. Eric Allender and Michal Koucký. Amplifying lower bounds by means of self-reducibility. *Journal of the ACM*, 57:14:1–14:36, 2010. doi:10.1145/1706591.1706594.
9. Eric Allender, Michal Koucký, Detlef Ronneburger, and Sambuddha Roy. The pervasive reach of resource-bounded Kolmogorov complexity in computational complexity theory. *Journal of Computer and System Sciences*, 77:14–40, 2010. doi:10.1016/j.jcss.2010.06.004.
10. Bill Fefferman, Ronen Shaltiel, Christopher Umans, and Emanuele Viola. On beating the hybrid argument. *Theory of Computing*, 9:809–843, 2013. doi:10.4086/toc.2013.v009a026.
11. Oded Goldreich. On promise problems: A survey. In Oded Goldreich, Arnold L. Rosenberg, and Alan L. Selman, editors, *Theoretical Computer Science, Essays in Memory of Shimon Even*, volume 3895 of *Lecture Notes in Computer Science*, pages 254–290. Springer, 2006. doi:10.1007/11685654_12.
12. Johan Håstad. *Computational Limitations for Small Depth Circuits*. MIT Press, Cambridge, MA, 1987.
13. Shuichi Hirahara and Rahul Santhanam. On the average-case complexity of mcsp and its variants. In *32nd Conference on Computational Complexity, CCC*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. to appear.

14 Shuichi Hirahara and Osamu Watanabe. Limits of minimum circuit size problem as oracle. In *31st Conference on Computational Complexity, CCC*, LIPIcs, pages 18:1–10:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CCC.2016.18.

15 Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *STOC'97*, pages 220–229, 1997. doi:10.1145/258533.258590.

16 Valentine Kabanets and Jin-Yi Cai. Circuit minimization problem. In *ACM Symposium on Theory of Computing (STOC)*, pages 73–79, 2000. doi:10.1145/335305.335314.

17 Adam Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002. doi:10.1137/S0097539700389652.

18 Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975. doi:10.1145/321864.321877.

19 Cody Murray and Ryan Williams. On the (non) NP-hardness of computing circuit complexity. In *30th Conference on Computational Complexity, CCC*, volume 33 of *LIPIcs*, pages 365–380. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CCC.2015.365.

20 Igor Oliveira and Rahul Santhanam. Conspiracies between learning algorithms, circuit lower bounds and pseudorandomness. In *32nd Conference on Computational Complexity, CCC*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. to appear.

21 Rafail Ostrovsky. One-way functions, hard on average problems, and statistical zero-knowledge proofs. In *IEEE Conference on Structure in Complexity Theory*, pages 133–138. IEEE Computer Society, 1991. doi:10.1109/SCT.1991.160253.

22 Rafail Ostrovsky and Avi Wigderson. One-way fuctions are essential for non-trivial zero-knowledge. In *Second Israel Symposium on Theory of Computing Systems (ISTCS)*, pages 3–17. IEEE Computer Society, 1993. doi:10.1109/ISTCS.1993.253489.

23 Alexander Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55:24–35, 1997. doi:10.1006/jcss.1997.1494.

24 Alexander A. Razborov. Lower bounds on the size of bounded depth networks over a complete basis with logical addition. *Matematicheskie Zametki*, 41:598–607, 1987. In Russian. English translation in *Mathematical Notes of the Academy of Sciences of the USSR* 41:333–338, 1987.

25 Michael Rudow. Discrete logarithm and minimum circuit size. Technical Report TR16-23, Electronic Colloquium on Computational Complexity, 2016. URL: https://eccc.weizmann.ac.il/report/2016/108/.

26 Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings 19th Symposium on Theory of Computing*, pages 77–8. ACM Press, 1987. doi:10.1145/28395.28404.

27 Jacobo Torán. On the hardness of graph isomorphism. *SIAM Journal on Computing*, 33(5):1093–1108, 2004. doi:10.1137/S009753970241096X.

28 Salil P. Vadhan. An unconditional study of computational zero knowledge. *SIAM Journal on Computing*, 36(4):1160–1214, 2006. doi:10.1137/S0097539705447207.

29 Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York Inc., 1999. doi:10.1007/978-3-662-03927-4.

Strategy Complexity of Concurrent Safety Games*

Krishnendu Chatterjee[1], Kristoffer Arnsfelt Hansen[2], and Rasmus Ibsen-Jensen[3]

1 **IST Austria, Klosterneuburg, Austria**
krish@ist.ac.at
2 **Aarhus University, Aarhus, Denmark**
arnsfelt@cs.au.dk
3 **IST Austria, Klosterneuburg, Austria**
ribsen@ist.ac.at

Abstract

We consider two player, zero-sum, finite-state concurrent reachability games, played for an infinite number of rounds, where in every round, each player simultaneously and independently of the other players chooses an action, whereafter the successor state is determined by a probability distribution given by the current state and the chosen actions. Player 1 wins iff a designated goal state is eventually visited. We are interested in the complexity of stationary strategies measured by their *patience*, which is defined as the inverse of the smallest non-zero probability employed.

Our main results are as follows: We show that: (i) the optimal bound on the patience of optimal and ϵ-optimal strategies, for both players is doubly exponential; and (ii) even in games with a single non-absorbing state exponential (in the number of actions) patience is necessary.

1998 ACM Subject Classification I.2.1 Games

Keywords and phrases Concurrent games, Reachability and safety, Patience of strategies

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.55

1 Introduction

Concurrent reachability games

Concurrent reachability games[8] are played on finite-state graphs by 2 players for an infinite number of rounds. In every round, each player simultaneously and independently of the other player chooses moves (or actions). The current state and the chosen moves of the players determine a probability distribution over the successor state. The result of playing the game (or a *play*) is an infinite sequence of states and actions. The play starts in a designated *start state*. Player 1 wins the play iff the play ever enters a designated *goal state*. We say that player 1 is the *reachability player* and player 2 the *safety player*. These games were introduced in a seminal work by Shapley [23], and have been one of the most fundamental and well-studied game models in stochastic graph games. Matrix games (or normal form games) can model a wide range of problems with diverse applications, when there is a finite number of interactions [19, 26]. Concurrent reachability games can be viewed as a finite set of matrix games, such that the choices made in the current game determine which game is played next, and is the appropriate model for many applications [11]. Moreover, in analysis of reactive systems, concurrent games provide the appropriate model for reactive systems with components that interact synchronously [6, 7, 1].

* Some proofs are missing. See full version https://arxiv.org/abs/1506.02434, [4]

Relevance

Concurrent reachability games are relevant in many applications. For example, the synthesis problem in control theory (e.g., discrete-event systems as considered in [22]) corresponds to reactive synthesis of [21]. The synthesis problem for synchronous reactive systems is appropriately modeled as concurrent games [6, 7, 8]. Other than control theory, concurrent reachability games also provide the appropriate model to study several other interesting problems, such as two-player poker games [18].

Properties of strategies

Given a concurrent reachability games, the player-1 *value* $v_1(s)$ of the game at a state s is the limit probability with which he can guarantee that the play will eventually enter the goal state against all strategies of player 2. The player-2 *value* $v_2(s)$ is analogously the limit probability with which player 2 can ensure his own objective against all strategies of player 1. Concurrent reachability games are determined [10], i.e., for each state s we have $v_1(s) + v_2(s) = 1$. A *strategy* for a player, given a history (i.e., finite prefix of a play) specifies a probability distribution over the actions. A *stationary* strategy does not depend on the history, but only on the current state. For $\varepsilon \geq 0$, a strategy is ε-optimal for a state s for player i if it ensures his own objective with probability at least $v_i(s) - \varepsilon$ against all strategies of the opponent. A 0-optimal strategy is an *optimal* strategy. In concurrent reachability games, there exist stationary optimal strategies for the safety player [20, 14]; whereas in contrast, for the reachability player, optimal strategies do not exist in general, however, for every $\varepsilon > 0$ there exists stationary ε-optimal strategies [10].

The significance of patience and roundedness of strategies

The basic decision problem is as follows: given a concurrent reachability game and a rational threshold λ, decide whether $v_1(s) \geq \lambda$. The basic decision problem is in PSPACE and is *square-root sum* hard [9][1]. Given the hardness of the basic decision problem, the next most relevant computational problem is to compute an approximation of the value. The computational complexity of the approximation problem is closely related to the size of the description of ε-optimal strategies. Even for special cases of concurrent reachability game, namely *turn-based* reachability games, where in each state at most one player can choose between multiple moves, the best known complexity results are obtained by guessing an optimal strategy and computing the value in the game obtained after fixing the guessed strategy. A strategy has patience p if p is the inverse of the smallest non-zero probability used by a distribution describing the strategy. A rational valued strategy has roundedness q if q is the greatest denominator of the probabilities used by the distributions describing the strategy. Note that if a strategy has roundedness q, then it also has patience at most q. The description complexity of a stationary strategy can be bounded by the roundedness. A stationary strategy with exponential roundedness, can be described using polynomially many bits, whereas the explicit description of stationary strategies with doubly-exponential patience is not polynomial. Thus obtaining upper bounds on the roundedness and lower bounds on the patience is at the heart of the computational complexity analysis of concurrent reachability games. Also see [27, 28, 24] for the significance of computing strategies in concurrent stochastic games.

[1] The square-root sum problem is an important problem from computational geometry, where given a set of natural numbers $n_1, n_2, \ldots, n_k$, the question is whether the sum of the square roots exceed an integer b. The problem is not known to be in NP.

Previous results and our contributions

In this work we consider concurrent reachability games. We first describe the relevant previous results and then our contributions.

Previous results

For concurrent reachability game, the optimal bound on patience and roundedness for ε-optimal strategies for the reachability player, for $\varepsilon > 0$, is doubly exponential [13, 12]. The doubly-exponential lower bound is obtained by presenting a family of games (namely, Purgatory) where the reachability player requires doubly-exponential patience (however, in this game the patience of the safety player is 1) [13, 12]; whereas the doubly-exponential upper bound is obtained by expressing the values in the existential theory of reals [13, 12]. In contrast to the reachability player that in general do not have optimal strategies, similar to the safety player there are two related classes of concurrent stochastic games that admit optimal stationary strategies, namely, discounted-sum, and ergodic concurrent games. For both these classes the optimal bound on patience and roundedness for ε-optimal strategies, for $\varepsilon > 0$, is exponential [5, 15]. The optimal bound on patience and roundedness for optimal and ε-optimal strategies, for $\varepsilon > 0$, for the safety player has been an open problem.

Our contributions

Our main results are as follows:

1. *Lower bound: general.* We show that in concurrent reachability games, a lower bound on patience of optimal and ε-optimal strategies, for $\varepsilon > 0$, for the safety player is doubly exponential (in contrast to the above mentioned related classes of games that only require exponential patience). We present a family of games (namely, Purgatory Duel) where optimal and ε-optimal strategies, for $\varepsilon > 0$, for both players require doubly-exponential patience.
2. *Lower bound: three states.* We show that even in concurrent reachability games with three states of which two are absorbing (sink states with only self-loop transitions) the patience required for optimal and ε-optimal strategies, for $\varepsilon > 0$, is exponential (in the number of actions). An optimal (resp., ε-optimal, for $\varepsilon > 0$) strategy in a game with three states (with two absorbing states) is basically an optimal (resp., ε-optimal) strategy of a matrix game, where some entries of the matrix game depend on the value of the non-absorbing state (as some transitions of the non-absorbing state can lead to itself). In standard matrix games, the patience for ε-optimal strategies, for $\varepsilon > 0$, is only logarithmic [17]; and perhaps surprisingly in contrast we show that the patience for ε-optimal strategies in concurrent reachability games with only three states is exponential (i.e., there is a doubly-exponential increase from logarithmic to exponential).
3. *Upper bound.* We show that in concurrent reachability games, an upper bound on the patience of optimal strategies and an upper bound on the patience and roundedness of ε-optimal strategies, for $\varepsilon > 0$, is as follows: (a) doubly exponential in general; and (b) exponential for the safety player if the number of value classes (i.e., the number of different values in the game) is constant. Hence our upper bounds on roundedness match our lower bound results for patience. Our results also imply that if the number of value classes is constant, then the basic decision problem is in coNP.

In summary, we present a complete picture of the patience and roundedness required in concurrent reachability games.

Table 1 Strategy complexity (i.e., patience and roundedness of ε-optimal strategies, for $\varepsilon > 0$) of the reachability vs safety player depending on the number of value classes. Our results are bold faced, and LB (resp., UB) denotes lower (resp., upper) bound on patience (resp., roundedness).

# Value classes	Reachability	Safety
1	Linear	One
2	Double-exponential	One
3	Double-exponential	**Exponential LB, Theorem 13**
Constant	Double-exponential	**Exponential UB, Theorem 14**
General	Double-exponential	**Double-exponential LB, Theorem 12 UB, Theorem 14**

Distinguishing aspects of safety and reachability

While the optimal bound on patience and roundedness we establish in concurrent reachability games for the safety player matches that for the reachability player, there are many distinguishing aspects for safety as compared to reachability in terms of the number of value classes (as shown in Table 1). For the reachability player, if there is one value class, then the patience and roundedness required is linear: it follows from the results of [2] that if there is one value class then all the values must be either 1 or 0; and if all states have value 0, then any strategy is optimal, and if all states have value 1, then it follows from [8, 3] that there is an almost-sure winning strategy (that ensures the objective with probability 1) from all states and the optimal bound on patience and roundedness is linear. The family of game graphs defined by Purgatory has two value classes, and the reachability player requires doubly exponential patience and roundedness, even for two value classes. In contrast, if there are (at most) two value classes, then again the values are 1 and 0; and in value class 1, the safety player has an optimal strategy that is stationary and deterministic (i.e., a positional strategy) and has patience and roundedness 1 [8], and in value class 0 any strategy is optimal. While for two value classes, the patience and roundedness is 1 for the safety player, we show that for three value classes (even for three states) the patience and roundedness is exponential, and in general the patience and roundedness is doubly exponential (and such a finer characterization does not exist for the reachability player).

Our main ideas

Our most interesting results are the doubly-exponential and exponential lower bound on the patience and roundedness. We now present a brief overview about the lower bound example.

The game of *Purgatory* [13, 12] is a concurrent reachability game that was defined as an example showing that the *reachability* player must, in order to play near optimally, use a strategy with non-zero probabilities that are *doubly exponentially* small in the number of states of the game (i.e., the patience is doubly exponential).

In this paper we present another example of a reachability game where this is the case for the *safety* player as well. The game Purgatory consists of a (potentially infinite) sequence of *escape attempts*. In an escape attempt one player is given the role of the *escapee* and the other player is given the role as the *guard*. An escape attempt consists of at most N

rounds. In each round, the guard selects and hides a number between 1 and m, and the escapee must try to guess the number. If the escapee successfully guesses the number N times, the game ends with the escapee as the winner. If the escapee incorrectly guesses a number which is strictly larger than the hidden number, the game ends with the guard as the winner. Otherwise, if the escapee incorrectly guesses a number which is strictly smaller than the hidden number, the escape attempt is over and the game continues.

The game of Purgatory is such that the reachability player is always given the role of the escapee, and the safety player is always given the role of the guard. If neither player wins during an escape attempt (meaning there is an infinite number of escape attempts) the safety player wins. Purgatory may be modeled as a concurrent reachability game consisting of N non-absorbing positions in which each player has m actions. The value of each non-absorbing position is 1. This means that the reachability player has, for any $\varepsilon > 0$, a stationary strategy that wins from each non-absorbing position with probability at least $1 - \varepsilon$ [10], but such strategies must have doubly-exponential patience. In fact for N sufficiently large and $m \geq 2$, such strategies must have patience at least $2^{m^{N/3}}$ for $\varepsilon = 1 - 4m^{-N/2}$ [12]. For the safety player however, the situation is simple: *any* strategy is optimal.

We introduce a game we call the *Purgatory Duel* in which the safety player must also use strategies of doubly-exponential patience to play near optimally. The main idea of the game is that it forces the safety player to behave as a reachability player. We can describe the new game as a variation on the above description of the Purgatory game. The Purgatory Duel consists also of a (potentially infinite) sequence of escape attempts. But now, before each escape attempt the role of the escapee is given to each player with probability $\frac{1}{2}$, and in each escape attempt the rules are as described above. The game remains asymmetric in the sense that if neither player wins during an escape attempt, the safety player wins. The Purgatory Duel may be modeled as a concurrent reachability game consisting of $2N + 1$ non-absorbing positions, in which each player has m actions, except for a single position where the players each have just a single action.

Technical contributions

The key non-trivial aspects of our proof are as follows: first, is to come up with the family of games, namely, Purgatory Duel, where the ε-optimal strategies, for $\varepsilon \geq 0$, for the players are symmetric, even though the objectives are complementary; and then the precise analysis of the game needs to combine and extend several ideas, such as refined analysis of matrix games, and analysis of perturbed Markov decision processes (MDPs) which are one-player stochastic games.

Highlights

We highlight two features of our results, namely, the surprising aspects and the significance (see Section Discussion and Conclusion of the full version for further details).

1. *Surprising aspects.* The first surprising aspect of our result is the doubly-exponential lower bound for the safety player in concurrent reachability games. The properties of strategies for the safety player in concurrent reachability games resemble concurrent discounted games, as in both cases optimal stationary strategies exist, and locally optimal strategies are optimal. We show that in contrast to concurrent discounted games where exponential patience suffices for the safety player in concurrent reachability games doubly-exponential patience is necessary. The second surprising aspect is the lower bound example itself. The lower bound example is obtained as follows: (i) given Purgatory we first obtain

simplified Purgatory by changing the start state such that it deterministically goes to the next state; (ii) we then consider its dual where the roles of the players are exchanged; and (iii) Purgatory duel is obtained by merging the start states of simplified Purgatory and its dual. Both in simplified Purgatory and its dual, there are only two value classes, and positional optimal strategies exist for the safety player. Surprisingly we show that a simple merge operation gives a game with linear number of value classes and the patience increases from 1 to doubly-exponential. Finally, the properties of strategies for the reachability- and safety-player in concurrent reachability games differ substantially. An important aspect of our lower bound example is that we show how to modify an example for the reachability player to obtain the result for safety player.

2. *Significance.* Our most important results are the lower bounds, and the main significance is threefold. First, the most well-studied way to obtain computational complexity result in games is to explicitly guess strategies, and then verify the game obtained fixing the strategy. The lower bound for the reachability player by itself did not rule out that better complexity results can be obtained through better strategy complexity for the safety player (indeed, for constant number of value classes, we obtain a better complexity result than known before due to the exponential bound on roundedness). Our doubly-exponential lower bound shows that in general the method of explicitly guessing strategies would require exponential space, and would not yield NP or coNP upper bounds. Second, one of the most well-studied algorithm for games is the strategy-iteration algorithm. Our result implies that any natural variant of the strategy-iteration algorithm for the safety player that explicitly compute strategies require exponential space in the worst-case. Finally, in games, strategies that are witness to the values and specify how to play the game, are as important as values, and our results establish the precise strategy complexity (matching upper bound of roundedness with lower bounds of patience).

Full-version: Proofs and non-zero-sum games.

In the full version [4], we give full proofs of all our lemmas and also consider non-zero-sum and non-two-player concurrent games, but where each player has either a reachability or safety objective (concurrent reachability games is then the special case of 1 player with a reachability objective and 1 player with the complementary safety objective).

2 Definitions

Other number

Given a number $i \in \{1, 2\}$ let $\widehat{i}$ be the other number, i.e., if $i = 1$, then $\widehat{i} = 2$ and vice-versa.

Probability distributions

A *probability distribution* d over a finite set Z, is a map $d : Z \to [0, 1]$, such that $\sum_{z \in Z} d(z) = 1$. Fix a probability distribution d over a set Z. The distribution d is *pure (Dirac)* if $d(z) = 1$ for some $z \in Z$ and for convenience we overload the notation and let $d = z$. The *support* $\mathrm{Supp}(d)$ is the subset Z' of Z, such that $z \in Z'$ if and only if $d(z) > 0$. The distribution d is *totally mixed* if $\mathrm{Supp}(d) = Z$. The *patience* of d is $\max_{z \in \mathrm{Supp}(d)} \frac{1}{d(z)}$, i.e., the inverse of the minimum non-zero probability. The *roundedness* of d, if $d(z)$ is a rational number for all $z \in Z$, is the greatest denominator of $d(z)$. Note that roundness of d is always at least the patience of d. Given two elements $z, z' \in Z$, the probability distribution $d = \mathsf{U}(z, z')$ over Z is such that $d(z) = d(z') = \frac{1}{2}$. Let $\Delta(Z)$ be the set of all probability distributions over Z.

Concurrent reachability games

A concurrent reachability game, consists of (1) a finite set of *states* S, of size N; and (2) for each state $s \in S$ and each player i a set A^i_s of *actions* (and $A^i = \bigcup_s A^i_s$ is the set of all actions for player i, for each i; and $A = \bigcup_i A^i$ is the set of all actions) such that A^i_s consists of at most m actions; and (3) a stochastic *transition function* $\delta : S \times A^1 \times A^2 \to \Delta(S)$; and (4) a designated *goal state* $g \in S$. A state s is *deterministic* if $\delta(s, a_1, a_2)$ is pure (deterministic), for all $a_i \in A^i_s$ and for all i. A state s is called *absorbing* if $A^i_s = \{a\}$ for all i and $\delta(s, a, a) = s$. The number $\delta_{\min}$ is the smallest non-zero transition probability.

How to play a concurrent reachability game

The game G, starting in state s, is played as follows: initially a pebble is placed on $v_0 := s$. In each time step $T \geq 0$, the pebble is on some state v_T and each player selects (simultaneously and independently of the other players, like in the game rock-paper-scissors) an action $a^i_{T+1} \in A^i_{v_T}$. Then, the game selects v_{T+1} according to the probability distribution $\delta(v_T, a^1_{T+1}, a^2_{T+1})$ and moves the pebble onto v_{T+1}. The game then continues with time step $T + 1$ (i.e., the game consists of infinitely many time steps). For a round T, let a_{T+1} be the pair of choices of the actions for the players, i.e., $a_{T+1,i}$ is the choice of player i, for each i. Round 0 is identified by v_0 and round $T > 0$ is then identified by the pair (a_T, v_T). A *play* P_s, starting in state $v_0 = s$, is then a sequence of rounds $(v_0, (a_1, v_1), (a_2, v_2), \ldots, (a_T, v_T), \ldots)$, and for each ℓ a prefix of P^ℓ_s of length ℓ is then $(v_0, (a_1, v_1), (a_2, v_2), \ldots, (a_T, v_T), \ldots, (a_\ell, v_\ell))$, and we say that P^ℓ_s *ends in* v_ℓ. Player 1 wins a play P_s iff $v_T = g$ for some T. Similarly, player 2 wins a play P_s iff $v_T \neq g$ for *all* T We refer to player 1 as the reachability player and player 2 as the safety player.

Strategies

Fix a player i. A strategy is a recipe to choose a probability distribution over actions given a finite prefix of a play. Formally, a strategy σ_i for player i is a map from P^ℓ_s, for a play P_s of length ℓ starting at state s, to a distribution over $A^i_{v_\ell}$. Player i *follows* a strategy σ_i, if given the current prefix of a play is P^ℓ_s, he selects $a_{\ell+1}$ according to $\sigma_i(P^\ell_s)$, for all plays P_s starting at s and all lengths ℓ. A strategy σ_i for player i, is *stationary*, if for all ℓ and ℓ', and all pair of plays P_s and $P'_{s'}$, starting at states s and s' respectively, such that P^ℓ_s and $(P')^{\ell'}_{s'}$ ends in the same state t, we have that $\sigma_i(P^\ell_s) = \sigma_i((P')^{\ell'}_{s'})$; and we write $\sigma_i(t)$ for the unique distribution used for prefix of plays ending in t. The *patience* (resp., *roundedness*) of a strategy σ_i is the supremum of the patience (resp. roundedness) of the distribution $\sigma_i(P^\ell_s)$, over all plays P_s starting at state s, and all lengths ℓ. Also, a strategy σ_i is *pure* (resp., *totally mixed*) if $\sigma_i(P^\ell_s)$ is pure (resp., totally mixed), for all plays P_s starting at s and all lengths ℓ. A strategy is *positional* if it is pure and stationary. Let Σ^i be the set of all strategies for player i.

Strategy profiles

A *strategy profile* $\sigma = (\sigma_1, \sigma_2)$ is a pair of strategies, one for each player. A strategy profile σ defines a unique probability measure on plays, denoted $\Pr_\sigma$, when the players follow their respective strategies [25]. We say that a strategy profile has a property (e.g., is stationary) if each of the strategies in the profile has that property.

Values

Let $u(G, s, \sigma)$ be the probability that player 1 wins the game G when the players follow σ and the play starts in s (i.e., the utility or payoff for player 1). Also if the game G is clear from context we drop it from the notation. Given a concurrent reachability game G, the *upper value* $\overline{\mathrm{val}}(G, s)$ (resp., *lower value* $\underline{\mathrm{val}}(G, s)$) of G starting in s is

$$\overline{\mathrm{val}}(G, s) = \sup_{\sigma_1 \in \Sigma^1} \inf_{\sigma_2 \in \Sigma^2} u(G, s, \sigma_1, \sigma_2) \ ; \quad \underline{\mathrm{val}}(G, s) = \inf_{\sigma_2 \in \Sigma^2} \sup_{\sigma_1 \in \Sigma^1} u(G, s, \sigma_1, \sigma_2) \ .$$

As shown by [10] we have that $\mathrm{val}(G, s) := \overline{\mathrm{val}}(G, s) = \underline{\mathrm{val}}(G, s)$; which is called the *value* of s. We will sometimes write $\mathrm{val}(s)$ for $\mathrm{val}(G, s)$ if G is clear from the context. We will also write val for the vector where $\mathrm{val}_s = \mathrm{val}(s)$.

(ε-)optimal strategies for concurrent reachability games

For an $\varepsilon \geq 0$, a strategy σ_1 for player 1 (resp., σ_2 for player 2) is called *ε-optimal* if for each state s we have that $\mathrm{val}(s) - \varepsilon \leq \inf_{\sigma_2 \in \Sigma^2} u(s, \sigma_1, \sigma_2)$ (resp., $\mathrm{val}(s) + \varepsilon \geq \sup_{\sigma_1 \in \Sigma^1} u(s, \sigma_1, \sigma_2)$). For each i, a strategy σ_i for player i is called *optimal* if it is 0-optimal. There exist concurrent reachability games in which player 1 does not have optimal strategies, see [10] for an example. On the other hand in all concurrent reachability games G player 1 has a stationary ε-optimal strategy for each $\varepsilon > 0$. In all concurrent reachability games player 2 has an optimal stationary strategy (thus also an ε-optimal stationary strategy for all $\varepsilon > 0$) [20, 14]. Also, given a stationary strategy σ_1 for player 1 we have that there exists a positional strategy σ_2, such that $u(s, \sigma_1, \sigma_2) = \inf_{\sigma_2' \in \Sigma^2} u(s, \sigma_1, \sigma_2')$, i.e., we only need to consider positional strategies for player 2. Similarly, we only need to consider positional strategies for player 1, if we are given a stationary strategy for player 2.

Markov decision processes and Markov chains

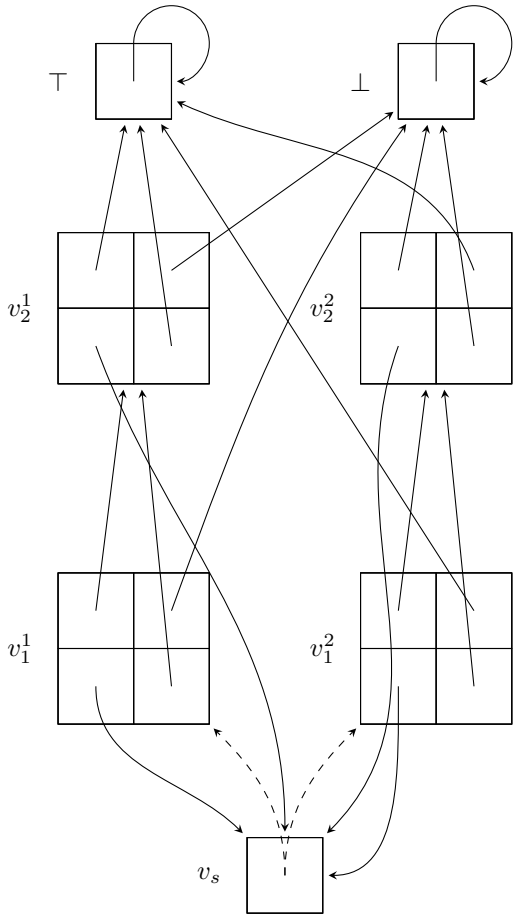
For each player i, a *Markov decision process (MDP) for player i* is a concurrent game where the size of A_s^j is 1 for all s and $j \neq i$. A *Markov chain* is an MDP for each player (that is the size of A_s^j is 1 for all s and j). A *closed recurrent set* of a Markov chain G is a maximal (i.e., no closed recurrent set is a subset of another) set $S' \subseteq S$ such that for all pairs of states $s, s' \in S$, the play starting at s reaches state s' eventually with probability 1 (note that it does not depend on the choices of the players as we have a Markov chain). For all starting states, eventually a closed recurrent set is reached with probability 1, and then plays stay in the closed reccurent set. Observe that fixing a stationary strategy for all but one player in a concurrent game, the resulting game is an MDP for the remaining player. Hence, fixing a stationary strategy for each player gives a Markov chain.

Game illustration

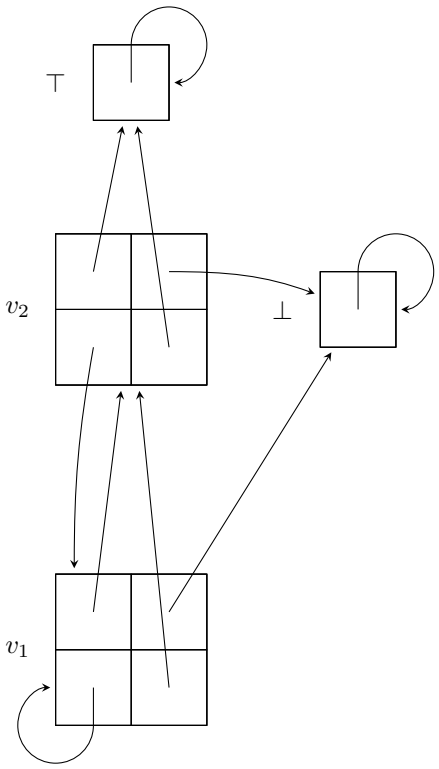
When we illustrate our games, we illustrate each state as a matrix, where the rows corresponds to the actions of the reachability player, the columns corresponds to the actions of the safety player. Thus, each entry e corresponds to an pair of actions (i, j) and a state s and we have an edge to $\delta(s, i, j)$ from e.

3 Patience Lower Bound

In this section we will establish the doubly-exponential lower bound on patience for concurrent reachability games. First we define the game family, namely, *Purgatory Duel* and we also recall the family *Purgatory*.



(a) Illustration of the Purgatory Duel with $m = n = 2$. The dashed edges have probability $\frac{1}{2}$ each.



(b) Illustration of Purgatory with $m = n = 2$.

Figure 1 Illustration of the games used for lower bounds.

The Purgatory Duel

In this paper we specifically focus on the following concurrent reachability game, the *Purgatory Duel*, defined on a pair of parameters (n, m). The game consists of $N = 2n+3$ states, namely $\{v_1^1, v_2^1, \ldots, v_n^1, v_1^2, v_2^2, \ldots, v_n^2, v_s, \top, \bot\}$ and all but v_s are deterministic. To simplify the definition of the game, let $v_0^1 = v_{n+1}^2 = \bot$ and $v_0^2 = v_{n+1}^1 = \top$. The states $\top$ and $\bot$ are absorbing. For each $i \in \{1, 2\}$ and $j \in \{1, \ldots, n\}$, the state v_j^i is such that $A^1_{v_j^i} = A^2_{v_j^i} = \{1, 2, \ldots, m\}$ and for each a_1, a_2 we have that $\delta(v_j, a_1, a_2)$ is (1) v_s if $a_1 > a_2$, (2) v_0^i if $a_1 < a_2$ and (3) v_{j+1}^i if $a_1 = a_2$. Finally, $A^1_{v_s} = A^2_{v_s} = \{a\}$ and $\delta(v_s, a, a) = \mathsf{U}(v_1^1, v_1^2)$. There is an illustration of the Purgatory Duel with $m = n = 2$ in Figure 1a.

The game Purgatory

We will also use the game *Purgatory* as defined by [12] (and also in [13] for the case of $m = 2$). Purgatory is similar to the Purgatory Duel and hence the similarity in names. Purgatory is also defined on a pair of parameters (n, m). The game consists of $N = n + 2$ states, namely, $\{v_1, v_2, \ldots, v_n, \top, \bot\}$ and each state is deterministic. To simplify the definition of the game,

let $v_{n+1} = \top$. For each $j \in \{1, \dots, n\}$, the state v_j is such that $A^1_{v_j} = A^2_{v_j} = \{1, 2, \dots, m\}$ and for each a_1, a_2 we have that $\delta(v_j, a_1, a_2)$ is (1) v_1 if $a_1 > a_2$, (2) $\bot$ if $a_1 < a_2$ and (3) v_{j+1} if $a_1 = a_2$. The states $\top$ and $\bot$ are absorbing. Furthermore, $S^1 = \{\top\}$. For an illustration of Purgatory with $m = n = 2$ see Figure 1b.

3.1 The patience of optimal strategies

In this section we present an approximation of the values of the states and the patience of the optimal strategies in the Purgatory Duel. We first show that the values of the states (besides $\top$ and $\bot$) are strictly between 0 and 1.

▶ **Lemma 1.** *Each state* $v \in \{v^1_1, v^1_2, \dots, v^1_n, v^2_1, v^2_2, \dots, v^1_2, v_s\}$ *is such that* $\mathrm{val}(v) \in [\frac{1}{m^{n+2}}, 1 - \frac{1}{m^{n+2}}]$

The proof of the above lemma is obtained by considering the strategy, for either player, that plays uniformly at random all available actions at every state. Next we show that every optimal stationary strategy for player 2 must be totally mixed.

▶ **Lemma 2.** *Let* σ_2 *be an optimal stationary strategy for player 2. The distribution* $\sigma_2(v^i_j)$ *is totally mixed and* $\mathrm{val}(v^1_j) > \mathrm{val}(v_s) > \mathrm{val}(v^2_j)$*, for all* i, j*.*

Next, we show that if either player follows a stationary strategy that is totally mixed on at least one side (that is, if there is an i', such that for each j the stationary strategy plays totally mixed in $v^{i'}_j$), then eventually either $\top$ or $\bot$ is reached with probability 1. The proof relies on the analysis of the Markov chain obtained given the strategies.

▶ **Lemma 3.** *For any* i *and* i'*, let* σ_i *be a stationary strategy for player* i*, such that* $\sigma_i(v^{i'}_j)$ *is totally mixed for all* j*. Let* $\sigma_{\widehat{i}}$ *be some stationary strategy for the other player. Then, each closed recurrent set in the Markov chain given by the game,* σ_i*, and* $\sigma_{\widehat{i}}$*, consists of only the state* $\top$ *or only the state* $\bot$*.*

The following definition basically *"mirrors"* a strategy σ_i for player i, for each i and gives it to the other player. We show (in Lemma 5) that if σ_2 is optimal for player 2, then the mirror strategy is optimal for player 1. We also show that if σ_2 is an ε-optimal strategy for player 2, for $0 < \varepsilon < \frac{1}{3}$, then so is the mirror strategy for player 1 (in Lemma 8).

▶ **Definition 4** (Mirror strategy)**.** Given a stationary strategy σ_i for player i, for either i, let the mirror strategy $\sigma^{\sigma_i}_{\widehat{i}}$ for player $\widehat{i}$ be the stationary strategy where $\sigma^{\sigma_i}_{\widehat{i}}(v^{\widehat{i'}}_j) = \sigma_i(v^{i'}_j)$ for each i' and j.

We next show that player 1 has optimal stationary strategies in the Purgatory Duel and give expressions for the values.

▶ **Lemma 5.** *Let* σ_2 *be some optimal stationary strategy for player 2. Then the mirror strategy* $\sigma_1^{\sigma_2}$ *is optimal for player 1. We have* $\mathrm{val}(v_s) = \frac{1}{2}$ *and* $\mathrm{val}(v^i_j) = 1 - \mathrm{val}(\widehat{v^i_j})$*, for all* i, j*.*

Finally, we give an approximation of the values of states in the Purgatory Duel and a lower bound on the patience of any optimal strategy of $2^{(m-1)^2 m^{n-2}}$.

▶ **Theorem 6.** *For each* j *in* $\{1, \dots, n\}$*, the value of state* v^1_j *in the Purgatory Duel is less than* $\frac{1}{2} + 2^{(1-m)\cdot m^{n-j}-1}$ *and for any optimal stationary strategy* σ_i *for either player* i*, the patience of* $\sigma_i(v^1_j)$ *is at least* $2^{(m-1)^2 m^{n-j-1}}$*.*

3.2 The patience of ε-optimal strategies

In this section we consider the patience of ε-optimal strategies for $0 < \varepsilon < \frac{1}{3}$. First we argue that each such strategy for player 2 is totally mixed on one side.

▶ **Lemma 7.** *For all $0 < \varepsilon < \frac{1}{2}$, each ε-optimal stationary strategy σ_2 for player 2 is such that $\sigma_2(v_j^2)$ is totally mixed, for all j.*

The idea is that against any strategy σ_2 that does not play totally mixed in some v_j^2, player 1 can ensure that if v_1^2 is entered, then $\perp$ is not reached before v_s is entered again (by playing 1 in $v_{j'}^2$, for $j' < j$ and some action not played by σ_2 in v_j^2). This allows player 1 to play a near optimal strategy from Purgatory in the states $v_{j'}^1$, ensuring that $\top$ is eventually reached with probability close to 1 from v_s and showing that σ_2 is far from optimal. We now show that if we mirror an ε-optimal strategy, then we get an ε-optimal strategy.

▶ **Lemma 8.** *For all $0 < \varepsilon < \frac{1}{3}$, each ε-optimal stationary strategy σ_2 for player 2 in the Purgatory Duel, is such that the mirror strategy $\sigma_1^{\sigma_2}$ is ε-optimal for player 1.*

Next we give a definition and a lemma, which is similar to Lemma 6 in [16]. The purpose of the lemma is to identify certain cases where one can change the transition function of an MDP in a specific way and obtain a new MDP with larger values.

▶ **Definition 9.** Let G be an MDP for a safety player. A *replacement set* is a set of triples of states, actions and distributions over the states $Q = \{(s_1, a_1, \delta_1), \ldots, (s_\ell, a_\ell, \delta_\ell)\}$. Given the replacement set Q, the MDP $G[Q]$ is an MDP over the same states as G, with the same set of safe states, and where the transition function δ' is similar to δ, except that $\delta'(s, a) = \delta_i$ if $s = s_i$ and $a = a_i$ for some i.

▶ **Lemma 10.** *Let G be an MDP with a safety player. Consider some replacement set $Q = \{(s_1, a_1, \delta_1), \ldots, (s_\ell, a_\ell, \delta_\ell)\}$, such that for all t and i we have that $\sum_{s \in S}(\delta(s_i, a_i)(s) \cdot \overline{v}_s^t) \leq \sum_{s \in S}(\delta_i(s) \cdot \overline{v}_s^t)$. Let $\overline{v'}^t$ be the value vector for $G[Q]$ with finite horizon t. (1) For all states s and time limits t we have that $\overline{v}_s^t \leq \overline{v'}_s^t$. (2) For all states s, we have that* $\mathrm{val}(G, s) \leq \mathrm{val}(G[Q], s)$.

The proof of the lemma is in the full version [4]. We next show that for player 1, the patience of ε-optimal strategies is high.

▶ **Lemma 11.** *For all $0 < \varepsilon < \frac{1}{3}$, each ε-optimal stationary strategy σ_1 for player 1 in the Purgatory Duel has patience at least $2^{m^{\Omega(n)}}$. For $N = 5$ the patience is $2^{\Omega(m)}$.*

The proof of the lemma is in the full version

We present the main theorem of this section. The proof follows easily from the previous lemmas (and is presented in details in the full version [4]).

▶ **Theorem 12.** *For all $0 < \varepsilon < \frac{1}{3}$, every ε-optimal stationary strategy, for either player, in the Purgatory Duel (that has $N = 2n + 3$ states and at most m actions for each player at all states) has patience $2^{m^{\Omega(n)}}$. For $N = 5$ the patience is $2^{\Omega(m)}$.*

3.3 The patience lower bound for three states

We show that the patience of all ε-optimal strategies, for all $0 < \varepsilon < \frac{1}{3}$, for both players in a concurrent reachability game G with three states of which two are absorbing, and the non-absorbing state has m actions for each player, can be as large as $2^{\Omega(m)}$. The key steps of

the proof are as follows: (1) First we consider the Purgatory duel with $n = 1$, and compress it down to 3 states by considering two steps of the Purgatory Duel in a single step. This gives us a game that has three states with one non-absorbing state (which we call 3-state Purgatory Duel) where ε-optimal strategies for the players require exponential patience in m. However, since two steps are simulated by a single step, this game increases the number of actions M from m to m^2. Hence, our patience bound for 3-state Purgatory Duel is only $2^{\Omega(\sqrt{M})}$. (2) We then show that we can restrict the above game to $2m - 1$ of the m^2 actions and still get the same patience as a function of m. We refer to this game as the restricted 3-state Purgatory Duel. Formally, we establish the following result.

▶ **Theorem 13.** *For all $0 < \varepsilon < \frac{1}{3}$, every ε-optimal stationary strategy, for either player, in the restricted 3-state Purgatory Duel (that has three states, two of which are absorbing, and the non-absorbing state has $O(m)$ actions for each player) has patience $2^{\Omega(m)}$.*

4 Patience Upper Bound

In this section we present the upper bounds. The values of concurrent reachability games can be expressed in the existential theory of reals. Using a refined analysis we present a formula where the number of variables depends only on the number of value classes, rather than the number of states. Using techniques similarly to [13] (such as quantifier elimination, sampling, and root separation for analysis of strategies in games), for concurrent reachability games with K value classes, we show that there is an optimal stationary strategy for the safety player where each probability is a real algebraic number, defined by a polynomial of degree $m^{O(K^2)}$ and the maximum coefficient bit-size is $\tau m^{O(K^2)}$, where τ is the bit-size of numbers in the input. We obtain the following theorem.

▶ **Theorem 14.** *For all concurrent reachability games with at most K different value-classes and probabilities that are rational numbers defined using at most τ bits, the following hold:*

1. *For all $\varepsilon > 0$, there exists an ε-optimal stationary strategy with roundedness at most $\frac{1}{\varepsilon} \lg \frac{1}{\varepsilon} 2^{N\tau m^{O(K^2)}}$.*
2. *For a fixed constant K, a state s and a number λ, given in binary, the problem of deciding whether $\mathrm{val}(s) \geq \lambda$ is in* coNP.

References

1 R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49:672–713, 2002.

2 K. Chatterjee. Concurrent games with tail objectives. *Theoretical Computer Science*, 388:181–198, 2007.

3 K. Chatterjee, L. de Alfaro, and T. Henzinger. Qualitative concurrent parity games. *ACM ToCL*, 2011.

4 K. Chatterjee, K. A. Hansen, and R. Ibsen-Jensen. Strategy complexity of concurrent stochastic games with safety and reachability objectives. *CoRR*, abs/1506.02434, 2015.

5 K. Chatterjee and R. Ibsen-Jensen. The Complexity of Ergodic Mean-payoff Games. In *ICALP 2014*, pages 122–133, 2014.

6 L. de Alfaro, T. Henzinger, and F. Mang. The control of synchronous systems. In *CONCUR'00*, LNCS 1877, pages 458–473. Springer, 2000.

7 L. de Alfaro, T. Henzinger, and F. Mang. The control of synchronous systems, Part II. In *CONCUR'01*, LNCS 2154, pages 566–580. Springer, 2001.

8 L. de Alfaro, T. A. Henzinger, and O. Kupferman. Concurrent reachability games. *Theor. Comput. Sci*, 386(3):188–217, 2007.

9 K. Etessami and M. Yannakakis. Recursive concurrent stochastic games. In *ICALP'06 (2)*, LNCS 4052, Springer, pages 324–335, 2006.

10 H. Everett. Recursive games. In *CTG*, volume 39 of *AMS*, pages 47–78, 1957.

11 J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.

12 K. A. Hansen, R. Ibsen-Jensen, and P. B. Miltersen. The complexity of solving reachability games using value and strategy iteration. In *CSR*, pages 77–90, 2011.

13 K. A. Hansen, M. Koucký, and P. B. Miltersen. Winning concurrent reachability games requires doubly-exponential patience. In *LICS*, pages 332–341, 2009.

14 C. J. Himmelberg, T. Parthasarathy, T. E. S. Raghavan, and F. S. V. Vleck. Existence of p-equilibrium and optimal stationary strategies in stochastic games. *Proc. Amer. Math. Soc.*, 60:245–251, 1976.

15 R. Ibsen-Jensen. *Strategy complexity of two-player, zero-sum games*. PhD thesis, Aarhus University, 2013.

16 R. Ibsen-Jensen and P. B. Miltersen. Solving simple stochastic games with few coin toss positions. In *ESA*, pages 636–647, 2012.

17 R. Lipton, E. Markakis, and A. Mehta. Playing large games using simple strategies. In *EC 03: Electronic Commerce*, pages 36–41. ACM Press, 2003.

18 P. B. Miltersen and T. B. Sørensen. A near-optimal strategy for a heads-up no-limit texas hold'em poker tournament. In *AAMAS'07*, pages 191–197, 2007.

19 G. Owen. *Game Theory*. Academic Press, 1995.

20 T. Parthasarathy. Discounted and positive stochastic games. *Bull. Amer. Math. Soc*, 77:134–136, 1971.

21 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of POPL*, pages 179–190. ACM Press, 1989.

22 P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

23 L. Shapley. Stochastic games. *PNAS*, 39:1095–1100, 1953.

24 E. Solan and N. Vieille. Computing uniformly optimal strategies in two-player stochastic games. *Economic Theory*, 42(1):237–253, 2010.

25 M. Vardi. Automatic verification of probabilistic concurrent finite-state systems. In *FOCS'85*, pages 327–338. IEEE, 1985.

26 J. von Neumann and O. Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1947.

27 O. Vrieze and F. Thuijsman. On equilibria in repeated games with absorbing states. *International Journal of Game Theory*, 18(3):293–310, 1989.

28 O. Vrieze and S. Tijs. Fictitious play applied to sequences of games and discounted stochastic games. *International Journal of Game Theory*, 11(2):71–85, 1982.

A Characterisation of Π^0_2 Regular Tree Languages

Filippo Cavallari[*1], **Henryk Michalewski**[2], **and Michał Skrzypczak**[†3]

1 **University of Lausanne, Department of Information Systems, Faculty of Business and Economics, Lausanne, Switzerland and University of Turin, Turin, Italy**
filippo.cavallari@unito.it

2 **University of Warsaw, Department of Mathematics, Informatics and Mechanics, Warsaw, Poland**
h.michalewski@mimuw.edu.pl

3 **University of Warsaw, Institute of Informatics, Warsaw, Poland**
m.skrzypczak@mimuw.edu.pl

Abstract

We show an algorithm that for a given regular tree language L decides if $L \in \mathbf{\Pi}^0_2$, that is if L belongs to the second level of Borel Hierarchy. Moreover, if $L \in \mathbf{\Pi}^0_2$, then we construct a weak alternating automaton of index $(0, 2)$ which recognises L. We also prove that for a given language L, L is definable by a weak alternating $(1, 3)$-automaton if and only if it is definable by a weak non-deterministic $(1, 3)$-automaton.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases infinite trees, Rabin-Mostowski hierarchy, regular languages

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.56

1 Introduction

Automata on infinite trees and the corresponding Monadic Second Order Logic provide a rich framework for expressing properties of regular languages of trees. Characterising natural subclasses of regular tree languages can be considered one of the fundamental problems related to automata on infinite trees. When we additionally require that the characterisation should be of an algorithmic nature, the problem usually turns to be very difficult and so far solved only in few instances.

▶ **Problem 1** (The Characterisation Problem). *For a given class of sets of trees $\mathcal{C}$ design an algorithm which decides if a given regular tree language L belongs to $\mathcal{C}$.*

Let us consider the problem for the following classes of tree languages: 1) languages definable in First Order Logic, 2) languages definable in Weak Monadic Second Order Logic, or 3) Borel languages. Providing an effective characterisation for any of the above classes among all regular languages of trees seems to be beyond the reach of currently available methods. In all the above instances it would be desirable to prove a dichotomy *simple* versus *difficult* languages; with *difficult* languages being characterised by existence of an embedding of a standard difficult language. In this work we resolve the **Characterisation Problem**

* The first author has been supported by ERC Consolidator grant LIPA (683080).
† The second and third authors have been supported by Polish National Science Centre grants 2014/13/B/ST6/03595 and 2016/22/E/ST6/00041 respectively.

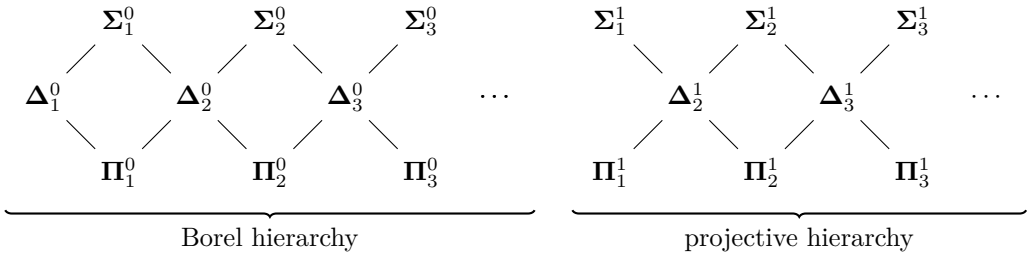


Figure 1 Borel and projective hierarchies.

for the case of $\mathcal{C} = \mathbf{\Pi}^0_2$, i.e. for the set of languages that belong to the second level of the Borel hierarchy. A posteriori, it turns out that $\mathcal{C}$ is the class of languages recognisable by weak alternating automata of *index* $(0, 2)$, that is the class of parity automata that involve priorities 0, 1, and 2; such that the transitions of the automaton are monotone wrt. priorities.

▶ **Theorem 2.** *If L is a regular tree language then either:*

- *L can be recognised by a weak alternating $(0, 2)$-automaton and so $L \in \mathbf{\Pi}^0_2$,*
- *L cannot be recognised by a weak alternating $(0, 2)$-automaton, $L \notin \mathbf{\Pi}^0_2$, and L is $\mathbf{\Sigma}^0_2$-hard.*

Moreover, it can be effectively decided which of the cases holds. If $L \in \mathbf{\Pi}^0_2$ then a weak alternating $(0, 2)$-automaton can effectively be constructed for L.

All regular languages of trees are $\mathbf{\Sigma}^1_2$ sets and from Rabin's Complementation Theorem [20] follows that every regular language of trees is in the class $\mathbf{\Delta}^1_2$. In the case of weak alternating automata (see e.g. [13]) one can provide a much more precise upper bound for the complexity:

▶ **Lemma 3** (See e.g. [6])**.** *If L is a language recognised by a weak alternating $(0, n)$-automaton then $L \in \mathbf{\Pi}^0_n$.*

Combining [24] and [6] we obtain that this is the optimal upper bound for the languages definable in Weak Monadic Second Order Logic.

The **Characterisation Problem** seems to be settled only for few families $\mathcal{C}$. The following list summarises all the cases which according to authors' knowledge have been considered in the literature so far, with the unrestricted input of general regular tree languages:

1. The simple case of clopen sets considered a mathematical folklore.
2. The case of open and closed sets, see [1, page 1] or [14, page 83-84]
3. The case of Boolean combinations of open sets settled in [1] using sophisticated algebraic methods.
4. The techniques of [1] were further reused in [7] to provide an effective characterisation of the class $\mathbf{\Delta}^0_2$. However, as it turned out, the application of the tools of [1] presented in that paper was not correct and the proofs contain some missing arguments[1]; as a corollary of the present article we also obtain another algorithm solving Problem 1 for the case $\mathcal{C} = \mathbf{\Delta}^0_2$. Additionally, this paper provides a new automata theoretic observation: regular languages in $\mathbf{\Delta}^0_2$ belong to the respective delta class of the weak alternating index hierarchy: for any regular language L in $\mathbf{\Delta}^0_2$ there exist a weak alternating $(0, 2)$-automaton and a weak alternating $(1, 3)$-automaton that recognise L.

[1] The statement of Theorem 1 in [7] is correct but a critical combinatorial Proposition 5 requires a different and a more sophisticated argument which will be presented in a journal version of that work.

5. From [9] we know that regular tree languages occupy the first ω–levels of Kolmogorov's $\mathcal{R}$-hierarchy.

The following problem that can be seen as a reversed version of Lemma 3 requires a fine-grained analysis of $\mathbf{\Pi}^0_n$ regular languages of trees:

▶ **Problem 4.** *Given a regular tree language that belongs to $\mathbf{\Pi}^0_n$, does there exist a weak alternating $(0,n)$-automaton that recognises it?*

In the case of $n = 1$ the positive answer is considered folklore (it also follows from [1]), however for every $n > 1$ the problem was open. Our article gives the positive answer for the specific case when $n = 2$.

Related work

Characterising WMSO among Büchi automata. The proof presented in this paper is inspired by a characterisation [23] of Borel languages (as well as recognisable by all weak alternating automata) among the languages recognisable by Büchi automata. Although the similar structure of the proof and the idea behind the characterisation game $\mathcal{F}$, there are certain differences between the two proofs. Firstly, the structure of the game $\mathcal{F}$ is much simpler here than in [23]. Moreover, the construction of the automata $\mathcal{G}_K$ from Section 6 requires certain new ideas because we deal with arbitrary parity automata (in [23] the input is restricted to Büchi automata and the construction is just an unravelling of the respective game $\mathcal{G}$). In particular, in this work we introduce the concept of K-acceptance.

Deterministic and other special classes of languages. In absence of a method solving Problem 1 for all regular languages, a number of attempts was made for special families of regular languages [18, 17, 16, 8] that are recognised by automata with restricted forms of non-determinism.

Cost-MSO and counter automata. Another take on characterising various classes of languages via games can be found in [5, 4, 3]. The authors of this paper are not aware of results directly applicable to weak alternating parity automata of index $(1,3)$. However, as shown in this paper, weak non-deterministic parity automata of index $(1,3)$ are equivalent with weak alternating parity automata of the same index. Therefore, it seems feasible to obtain the automata-theoretic part (without the correspondence to topological classes) of Theorem 2 using the tools presented in [5, 12, 4].

2 Basic notions

Trees. Let us fix a finite alphabet A, that is just a finite non-empty set of symbols (e.g. $A = \{a, b\}$). We work on the space of labelled complete infinite binary trees over A. An *A-labelled tree* t (shortly *tree*) is a function $t\colon \{\mathtt{L},\mathtt{R}\}^* \to A$ where the symbols $\mathtt{L}$, $\mathtt{R}$ are called *directions*. The set of all A-labelled trees is denoted by Tr_A.

Elements $u \in \{\mathtt{L},\mathtt{R}\}^*$ are called *nodes* of a tree. The elements $u\mathtt{L}$, $u\mathtt{R}$ are called *children* of u. The empty sequence ϵ is called the root of a tree. A *branch* of a tree is just an infinite sequence of directions $\beta \in \{\mathtt{L},\mathtt{R}\}^\omega$. A node u is on a branch α if it is a prefix of α, i.e. $u \prec \alpha$. In that case $u = \alpha\restriction_{|u|}$. If u is a node of a tree then by $t\restriction_u$ we indicate the tree t truncated in u in the usual sense: $t\restriction_u(w) \stackrel{\text{def}}{=} t(uw)$.

Topological complexity. In this work we use the standard topological notions for the space of infinite trees, see [11, 26] . The relevant topological notions in the context of infinite trees are described in Sections 1.6.1 and 1.6.2 of [22].

The space Tr_A with the standard product topology is known to be an uncountable Polish space (homeomorphic with the Cantor set). Thus, all the standard notions of Descriptive Set Theory naturally apply to trees.

Assume that $\mathcal{X}$ is a topological space known from the context. By $\mathbf{\Sigma}^0_1$ (resp. $\mathbf{\Pi}^0_1$) we denote the set of open (resp. closed) sets in $\mathcal{X}$. The classes $\mathbf{\Sigma}^0_{n+1}$ and $\mathbf{\Pi}^0_{n+1}$ are defined inductively: $\mathbf{\Sigma}^0_{n+1}$ contains countable unions of sets in $\mathbf{\Pi}^0_n$; $\mathbf{\Pi}^0_{n+1}$ contains countable intersections of sets in $\mathbf{\Sigma}^0_n$. In particular $\mathbf{\Sigma}^0_2$ are countable unions of closed sets; this class is often denoted F_σ. Similarly, $\mathbf{\Pi}^0_2$ are countable intersections of open sets; often denoted G_δ.

For a class Γ of sets (e.g. $\mathbf{\Pi}^0_2$), we say that a set $Y \subseteq \mathcal{X}$ is Γ-*hard* if for every set $Y' \in \mathcal{X}'$ that is in Γ there exists a continuous reduction $f\colon \mathcal{X}' \to \mathcal{X}$ such that $Y' = f^{-1}[Y]$. A set Y is Γ-complete if Y is Γ-hard and belongs to Γ.

Automata Theory. In this work we use both notions of *non-deterministic* and *alternating* parity tree automata. Again, we refer the reader to [19, 25]. The notation we use comes from Sections 1.3 and 1.4 of [22].

A *parity tree automaton* $\mathcal{A}$ is a tuple $\mathcal{A} = \langle A^{\mathcal{A}}, Q^{\mathcal{A}}, q_{\mathrm{I}}^{\mathcal{A}}, \Delta^{\mathcal{A}}, \Omega^{\mathcal{A}} \rangle$, where: $A^{\mathcal{A}}$ is the alphabet we are working on; $Q^{\mathcal{A}}$ is the set of *states* of the automaton $\mathcal{A}$; $q_{\mathrm{I}}^{\mathcal{A}}$ is a particular element of $Q^{\mathcal{A}}$ and it is called the *initial state*; $\Delta^{\mathcal{A}}$ will be defined in a moment; and $\Omega^{\mathcal{A}}$ is a function $\Omega^{\mathcal{A}}\colon Q^{\mathcal{A}} \to \omega$ that assigns a *priority* to every state of the automaton. If the automaton is known from the context then we omit the superscript $\mathcal{A}$.

An automaton $\mathcal{A}$ is *non-deterministic* if $\Delta \subseteq Q \times A \times Q \times Q$ contains *transitions* of the form $(q, a, q_{\mathtt{L}}, q_{\mathtt{R}})$. A non-deterministic automaton $\mathcal{A}$ *accepts* a tree $t \in \mathrm{Tr}_A$ if there exists an *accepting run* ρ, i.e. a $Q^{\mathcal{A}}$-labelled tree that is consistent with the transitions of $\mathcal{A}$ and the parity condition is satisfied on every branch β of t: $\limsup_{n\to\infty} \Omega\big(\rho(\beta\restriction_n)\big)$ is even.

An automaton $\mathcal{A}$ is *alternating* if Δ is a function that assigns to each pair $q \in Q$, $a \in A$ a finite positive Boolean combination of pairs (d, q') where $d \in \{\mathtt{L}, \mathtt{R}\}$ is a direction and $q' \in Q$ is the consecutive state. For instance $\Delta(q, a)$ can be of the form $\big((\mathtt{L}, q'_{\mathtt{L}}) \wedge (\mathtt{L}, q''_{\mathtt{L}})\big) \vee (\mathtt{R}, q''_{\mathtt{R}})$.

An alternating automaton $\mathcal{A}$ induces, for every tree $t \in \mathrm{Tr}_A$, a parity game $\mathcal{A}(t)$ called the *acceptance game* of $\mathcal{A}$ on t. $\mathcal{A}$ *accepts* t if $\exists$ has a winning strategy in the game $\mathcal{A}(t)$.

We require our automata to be *complete*, meaning that for every state $q \in Q$ and letter $a \in A$ there needs to be some transition.

For both non-deterministic and alternating tree automata $\mathcal{A}$ we define the *language* of $\mathcal{A}$ (denoted $\mathrm{L}(\mathcal{A})$) as the set of all trees accepted by $\mathcal{A}$. It is known that the expressive power of non-deterministic and alternating automata is the same:

▶ **Theorem 5** ([10]). *Let $L \subseteq \mathrm{Tr}_A$. There exists an alternating parity tree automaton $\mathcal{A}$ such that $\mathrm{L}(\mathcal{A}) = L$ if and only if there exists a non-deterministic parity tree automaton $\mathcal{B}$ such that $\mathrm{L}(\mathcal{B}) = L$. Moreover, both translations are effective.*

If for $L \subseteq \mathrm{Tr}_A$ there exists a non-deterministic (equivalently alternating) automaton $\mathcal{A}$ such that $L = \mathrm{L}(\mathcal{A})$ then we say that L is *regular*. A parity automaton is *weak* if the values of Ω are non-decreasing along transitions. The *index* of an automaton is the pair (i, j) where i is the minimal and j is the maximal value of Ω on Q.

3 Overview of the proof of Theorem 2

Let L be a regular language and let us fix once and for all two non-deterministic parity tree automata $\mathcal{A}$ and $\mathcal{B}$ that recognise respectively: $\mathrm{L}(\mathcal{B}) = L$ is the given language and $\mathrm{L}(\mathcal{A}) = L^{\mathrm{c}}$ is its complement. The proof will consist of the following steps:

- First we define a game $\mathcal{F}$ of infinite duration and perfect information. The game $\mathcal{F}$ is played by two players: Eve ($\exists$) and Adam ($\forall$). Player $\exists$ constructs a tree t together with three runs: one of the automaton $\mathcal{A}$ and two of the automaton $\mathcal{B}$. The second of them is influenced by $\forall$ who can ask $\exists$ to *restart* whenever he wants. The crucial property of the game $\mathcal{F}$ is that it is played over a finite arena and the winning condition is ω-regular.
- If $\exists$ wins $\mathcal{F}$ then her winning strategy can be used to prove that the language $\mathrm{L}(\mathcal{B})$ is actually $\mathbf{\Sigma}^0_2$-hard. In particular it cannot be recognised by a weak alternating automaton of index $(0, 2)$. To prove this topological hardness we test the winning strategy of $\exists$ against a well-designed family of strategies of $\forall$. In terms of Descriptive Set Theory it can be seen as finding an embedding of the Cantor set 2^ω that intersects $\mathrm{L}(\mathcal{B})$ on rationals.
- If $\forall$ wins $\mathcal{F}$ then we use his finite-memory winning strategy to construct a finite approximation of the automaton $\mathcal{B}$ that is denoted $\mathcal{G}_{K_0}$. The construction ensures that $\mathcal{G}_{K_0}$ is a weak alternating automaton of index $(0, 2)$ that recognises $\mathrm{L}(\mathcal{B})$.

4 The game $\mathcal{F}$

We start by defining a game $\mathcal{F}$ of infinite duration that is based on the non-deterministic parity tree automata $\mathcal{A} = \langle A, Q^{\mathcal{A}}, q_{\mathrm{I}}^{\mathcal{A}}, \Delta^{\mathcal{A}}, \Omega^{\mathcal{A}} \rangle$ and $\mathcal{B} = \langle A, Q^{\mathcal{B}}, q_{\mathrm{I}}^{\mathcal{B}}, \Delta^{\mathcal{B}}, \Omega^{\mathcal{B}} \rangle$ for L^{c} and L respectively. The purpose of $\mathcal{F}$ is to satisfy the following two propositions.

▶ **Proposition 6.** *If $\exists$ wins $\mathcal{F}$ then $\mathrm{L}(\mathcal{B})$ is $\mathbf{\Sigma}^0_2$-hard.*

▶ **Proposition 7.** *If $\forall$ wins $\mathcal{F}$ then $\mathrm{L}(\mathcal{B})$ is recognised by a weak alternating $(0, 2)$-automaton.*

The above propositions together with Lemma 3 give a complete characterisation of the topological complexity and the weak index of $\mathrm{L}(\mathcal{B})$.

Positions of $\mathcal{F}$. The positions of $\mathcal{F}$ are of the form $(p, q, s) \in Q^{\mathcal{A}} \times Q^{\mathcal{B}} \times Q^{\mathcal{B}}$ where: $p \in Q^{\mathcal{A}}$ is called an *$\mathcal{A}$-state*, $q \in Q^{\mathcal{B}}$ is called a *$\mathcal{B}$-state*, $s \in Q^{\mathcal{B}}$ is called an *active state*. The initial position of $\mathcal{F}$ is $(q_{\mathrm{I}}^{\mathcal{A}}, q_{\mathrm{I}}^{\mathcal{B}}, q_{\mathrm{I}}^{\mathcal{B}})$.

Rounds of $\mathcal{F}$. Assume that a round of $\mathcal{F}$ starts in a position (p, q, s). The choices done by the players are as follows:

1. $\forall$ can choose to *restart* by letting $s' = q$ or to *stay* by keeping $s' = s$.
2. $\exists$ declares: (i) a letter $a \in A$; (ii) a transition $(p, a, p_{\mathtt{L}}, p_{\mathtt{R}}) \in \Delta^{\mathcal{A}}$ of $\mathcal{A}$; (iii) a transition $(q, a, q_{\mathtt{L}}, q_{\mathtt{R}}) \in \Delta^{\mathcal{B}}$ of $\mathcal{B}$; (iv) another transition $(s', a, s'_{\mathtt{L}}, s'_{\mathtt{R}}) \in \Delta^{\mathcal{B}}$ of $\mathcal{B}$.
3. $\forall$ responds by selecting a direction $d \in \{\mathtt{L}, \mathtt{R}\}$.

After such a round the game proceeds to the position (p_d, q_d, s'_d). Four example rounds of $\mathcal{F}$ are presented in Figure 2.

If π is a finite or infinite play of $\mathcal{F}$, a *trace* is a finite or infinite sequence of active states s in consecutive rounds in which $\forall$ has not *restarted*. Thus, those active states come from successive transitions of the automaton $\mathcal{B}$.

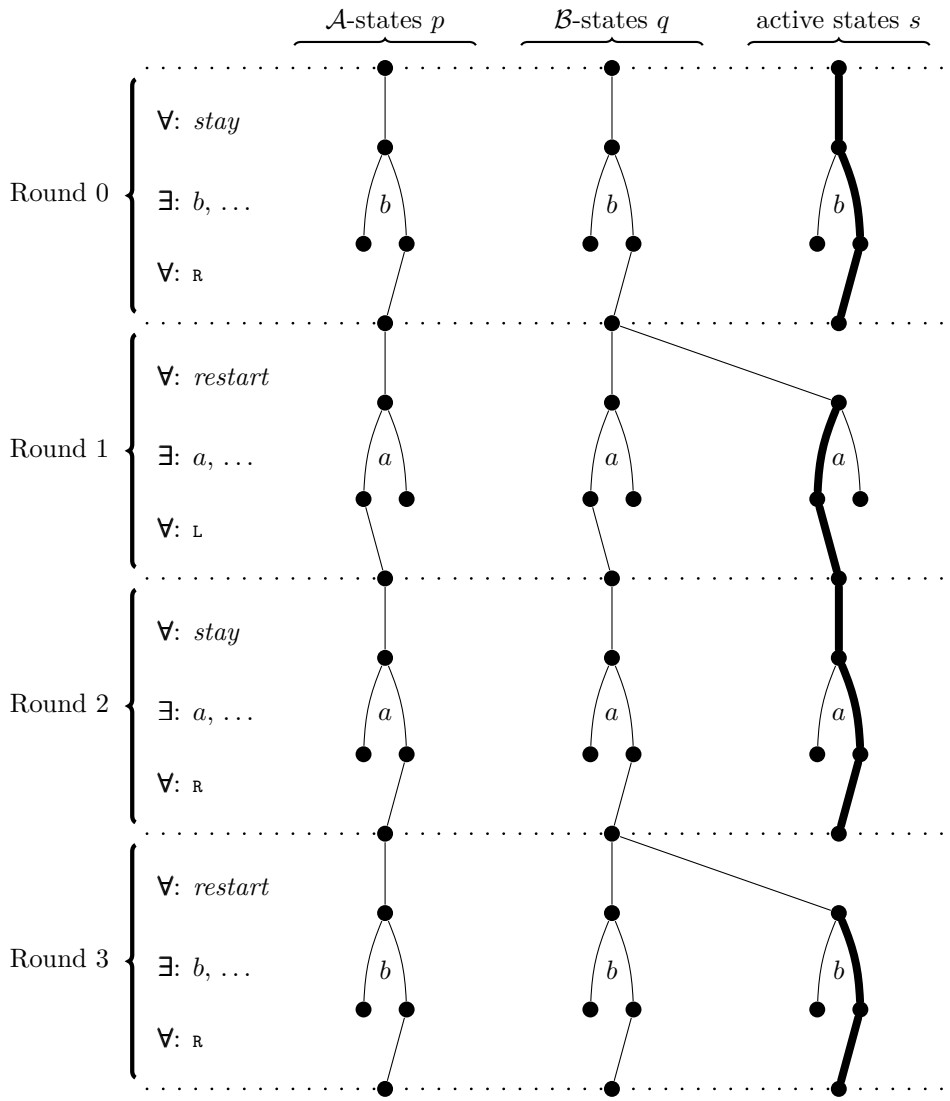


Figure 2 Four consecutive rounds of the game $\mathcal{F}$. The black dots are the states of the automata $\mathcal{A}$ and $\mathcal{B}$. Each round consists of three choices: first ∀ either *restarts* or *stays*, then ∃ provides a letter and three transitions (depicted by those Λ-shaped gadgets), finally ∀ chooses a direction. The three boldfaced paths are three traces formed by the active states: the first one lasts in Round 0; the second one in Rounds 1 and 2; the third one starts in Round 3.

Winning condition of $\mathcal{F}$. Now we will define the winning condition for ∃ in $\mathcal{F}$. It will depend on a Boolean combination of the following three properties, speaking about the sequence of rounds that were played:

(WR) ∀ has *restarted* infinitely many times.

(WA) The sequence of $\mathcal{A}$-states p is accepting in $\mathcal{A}$.

(WB) The sequence of active states s is accepting in $\mathcal{B}$ (i.e. it satisfies the parity condition).

A play of $\mathcal{F}$ is winning for ∃ if it satisfies

$$\big((\mathrm{WR}) \wedge (\mathrm{WA})\big) \ \vee \ \big(\neg(\mathrm{WR}) \wedge (\mathrm{WB})\big). \tag{1}$$

In other words, there are two cases: If ∀ has restarted infinitely many times then ∃ wins iff the sequence of visited $\mathcal{A}$-states satisfies the parity condition. If ∀ has restarted only finitely many times then ∃ wins iff the sequence of visited active states satisfies the parity condition. Notice that a priori both (WA) and (WB) can happen simultaneously, because for example there may exist two runs $\rho_{\mathcal{A}}$ and $\rho_{\mathcal{B}}$ of the automata $\mathcal{A}$ and $\mathcal{B}$ that both satisfy the parity condition on some particular branch.

By the definition, the winning condition of $\mathcal{F}$ is an ω-regular property of sequences of rounds. Additionally, there are only finitely many positions of $\mathcal{F}$ and each round allows finitely many possible choices by the players. Therefore, we obtain the following fact.

▶ **Fact 8** ([2]). *The winner of $\mathcal{F}$ can be effectively found and he/she can win using a finite memory winning strategy.*

5 Proof of Proposition 6

In this section we prove that if ∃ wins $\mathcal{F}$ then $\mathrm{L}(\mathcal{B})$ is $\mathbf{\Sigma}^0_2$-hard. Let $\sigma_\exists$ be her winning strategy. Let $C \subseteq \{0,1\}^\omega$ be the set of sequences containing only finitely many 1s.

It is known that C is $\mathbf{\Sigma}^0_2$-complete [26]. We will construct a continuous reduction from C to $\mathrm{L}(\mathcal{B})$ and so we obtain that $\mathrm{L}(\mathcal{B})$ is $\mathbf{\Sigma}^0_2$-hard.

We will say that σ is a *quasi-strategy* of ∀ in $\mathcal{F}$ if σ specifies when to *restart* and leaves undecided the choice of directions d. Notice that if σ is a quasi-strategy of ∀ then we can construct a tree t consisting of the letters a played by $\sigma_\exists$ against σ: the letter $t(u)$ is the $(|u|+1)$th letter played by $\sigma_\exists$ against ∀ playing accordingly to σ and choosing successive directions of u.

To each sequence $\alpha \in \{0,1\}^\omega$ we will assign a quasi-strategy σ_α of ∀ in $\mathcal{F}$. Consider $\alpha \in \{0,1\}^\omega$ and an Mth round of $\mathcal{F}$ for $M = 0, 1, \ldots$

- If $\alpha(M) = 0$ then σ_α *stays* by keeping $s' = s$.
- If $\alpha(M) = 1$ then σ_α *restarts* by putting $s' = q$.

Let the tree t_α be the effect of confronting the strategy $\sigma_\exists$ against the quasi-strategy σ_α. Since the behaviour of the strategy σ_α in an Mth round of $\mathcal{F}$ depends only on the first M bits of α, the function $\alpha \mapsto t_\alpha$ is continuous. A routine verification (see below) shows that

$$\alpha \in C \iff t_\alpha \in \mathrm{L}(\mathcal{B}). \tag{2}$$

When $\alpha \in C$. First assume that $\alpha \in C$, i.e. that there are only finitely many 1s in α. Let M be the maximal number such that $\alpha(M-1) = 1$ (or $M = 0$ if there is no such M). Let $\rho^{\mathcal{B}}$ be the run of $\mathcal{B}$ defined as follows:

- For $|u| \leq M$ let $\rho^{\mathcal{B}}(u)$ be the $\mathcal{B}$-state q from the beginning of the $|u|$th round of the play consistent with $\sigma_\exists$ and σ_α in which the sequence of directions chosen by ∀ was u.
- For $|u| > M$ let $\rho^{\mathcal{B}}(u)$ be the active state s from the beginning of the $|u|$th round of the play consistent with $\sigma_\exists$ and σ_α in which the sequence of directions chosen by ∀ was u.

It is easy to see that $\rho^{\mathcal{B}}$ is in fact a run of $\mathcal{B}$ over t_α. It remains to see that it is accepting. Consider an infinite branch of t_α. This branch corresponds to an infinite play of $\mathcal{F}$ consistent with $\sigma_\exists$ and the quasi-strategy σ_α. Since in all the rounds after the Mth one, ∀ has *stayed* by putting $s' = s$, the states of $\rho^{\mathcal{B}}$ form an infinite trace in that play. Therefore, the condition ¬(WR) holds. As the play is won by ∃, also (WB) must hold. It means that the trace must be accepting in $\mathcal{B}$, thus the run $\rho^{\mathcal{B}}$ is accepting on our branch. This way we have proved that $\rho^{\mathcal{B}}$ is accepting and $t_\alpha \in \mathrm{L}(\mathcal{B})$.

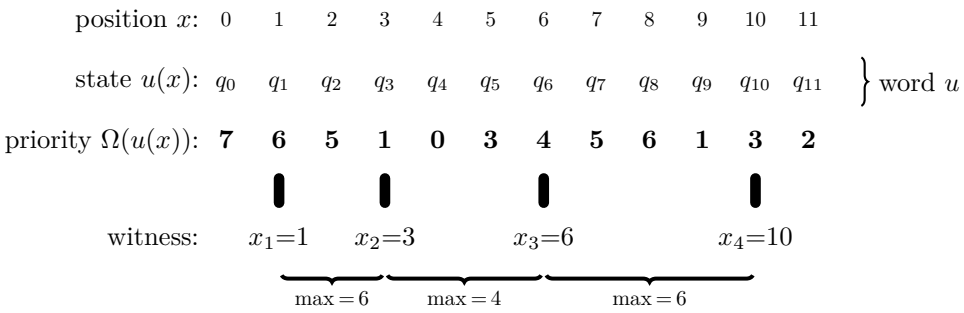


Figure 3 A word u that is 4-accepting. The sequence x_1, x_2, x_3, x_4 witnesses that. If we loop u between the positions 1 and 6, we get the sequence $q_0, q_1, \ldots, q_6, q_1, q_2, \ldots, q_6, q_1 \ldots$ that satisfies the parity condition.

When $\alpha \notin C$. Now assume that $\alpha \notin C$, i.e. that there are infinitely many 1s in α. Our aim is to prove that the run $\rho^{\mathcal{A}}$ formed by the $\mathcal{A}$-states p played by $\exists$ in all the plays consistent with $\sigma_\exists$ and σ_α is accepting. Consider an infinite branch of t_α and the corresponding play π of $\mathcal{F}$. Since infinitely many times $\forall$ has *restarted*, this play satisfies (WR). As the play is won by $\exists$, also (WA) must hold. It means that the sequence of $\mathcal{A}$-states p must be accepting in $\mathcal{A}$. Thus, we have proved that $t_\alpha \in \mathrm{L}(\mathcal{A})$ and therefore $t_\alpha \notin \mathrm{L}(\mathcal{B})$. This concludes the proof of $\mathbf{\Sigma}_2^0$-hardness of $\mathrm{L}(\mathcal{B})$.

6 Proof of Proposition 7

In this section we prove that if $\forall$ wins $\mathcal{F}$ then $\mathrm{L}(\mathcal{B})$ can be recognised by a weak alternating parity automaton of index $(0, 2)$. Since the winning condition of $\mathcal{F}$ is ω-regular, we can assume that $\forall$ wins using a strategy $\sigma_\forall$ based on a finite-memory structure M. Our aim is to construct an automaton recognising $\mathrm{L}(\mathcal{B})$.

***K*-accepting runs.** We start by defining a notion of K-accepting sequences — sequences of states of $\mathcal{B}$ that are similar to accepting ones. We will show that the strategy $\sigma_\forall$ must avoid such sequences.

Let u be a finite or infinite sequence of states of $\mathcal{B}$. Consider a number $K \in \omega$. We say that u is *K-accepting* if there exists a sequence of positions $0 \leq x_1 < x_2 < \ldots < x_K < |u|$ such that for every $n = 1, 2, \ldots, K-1$ we have:

$$\max\left\{\Omega^{\mathcal{B}}\big(u(x)\big) \;\middle|\; x_n \leq x \leq x_{n+1}\right\} \text{ is even.} \tag{3}$$

In other words, for $n = 1, \ldots, K-1$, the maximal priority of states between positions x_n and x_{n+1} of u must be even. We call such a sequence of positions $(x_1, \ldots, x_K)$ a *witness* of K-acceptance, see Figure 3.

The above definition is constructed in such a way to guarantee the following properties:

(P1) If u is K-accepting then it contains K positions such that each cycle built using an interval between two of them gives us a sequence of states satisfying the parity condition.

(P2) For every K, the set of all finite words that are K-accepting is regular. It is not obvious how a regular expression for this language should look like, however, the definition of the property of being K-accepting is clearly MSO-definable, thus by the results of Rabin, Scott [21], and Trakhtenbrot [27] (cf. e.g. [19]), we know that this language is regular.

(P3) If u is K-accepting then every word of the form uw is also K-accepting.
(P4) If $\alpha \in \left(Q^{\mathcal{B}}\right)^{\omega}$ satisfies the parity condition then for every $K \in \omega$ there exists a finite prefix of α that is K-accepting.

▶ **Lemma 9.** *There exists a value $K_0 \in \omega$ such that if π is an infinite play of $\mathcal{F}$ consistent with $\sigma_{\forall}$ then no trace of π is K_0-accepting.*

Proof. Let $K_0 \stackrel{\text{def}}{=} \left(\left|Q^{\mathcal{A}}\right| \times \left|Q^{\mathcal{B}}\right| \times \left|Q^{\mathcal{B}}\right|\right) \times \left|M\right| + 1$ where inside the brackets is the number of positions of $\mathcal{F}$ and M is the memory structure of $\sigma_{\forall}$.

Assume for the sake of contradiction that there exists a play π that is consistent with $\sigma_{\forall}$ and contains a K_0-accepting trace. For a round number $x \in \omega$ during π let (v_x, m_x) be the configuration of the game at the moment when x rounds were played: $v_x = (p_x, q_x, s_x)$ for an $\mathcal{A}$-state p_x, $\mathcal{B}$-state q_x, and active state s_x; and m_x is the current memory value of $\sigma_{\forall}$.

By the assumption we know that for some $x < y < \omega$ the sequence of active states s_x, s_{x+1}, …, s_y is a trace (i.e. there is no *restart* during these rounds) and it is K_0-accepting. Let $x \leq x_1, \ldots, x_{K_0} \leq y$ be a sequence of numbers of rounds that is a witness for the K_0-acceptance of this trace, see Equation (3).

Figure 4 provides an illustration for this construction. The upper picture presents a play π seen in the product of the game $\mathcal{F}$ and the memory structure M used by $\sigma_{\forall}$. Small dots mark positions before successive rounds of this play. The lower picture presents the play π in a chronological way. The boldfaced vertical snake-like shape is a trace that is 5-accepting; the rounded shapes indicate a witness of this fact: $x_1 = 2$, $x_2 = 3$, $x_3 = 6$, $x_4 = 8$, and $x_5 = 10$. Since 5 is bigger than the number of available pairs (v, m) we have a repetition: $(v_3, m_3) = (v_8, m_8)$. This allows us to construct a new play π', by staying forever on the loop between the rounds 3 and 8. The play π' obtained this way contains an infinite trace that satisfies the parity condition.

By the choice of K_0 we know that for some $1 \leq n < n' \leq K_0$ we have: $v_{x_n} = v_{x_{n'}}$ and $m_{x_n} = m_{x_{n'}}$; i.e. there must be a repetition of the position of $\mathcal{F}$ and the memory of $\sigma_{\forall}$ among the positions witnessing K_0-acceptance of the trace.

Consider a play π' of $\mathcal{F}$ which starts as π for the first x_n rounds. Then π' follows the loop between the rounds $x_n + 1$ and $x_{n'}$. Notice that π' is in fact a play because $v_{x_n} = v_{x_{n'}}$. Since we have chosen the positions x_n, $x_{n'}$ from a trace, this loop does not contain a *restart*. Clearly π' is consistent with $\sigma_{\forall}$ because the memory values m_{x_n} and $m_{x_{n'}}$ are equal.

Because x_n and $x_{n'}$ are chosen from a witness of K_0-acceptance of the trace, Property (P1) implies that π' contains an infinite accepting trace. Therefore, the play π' satisfies $\big(\neg(\mathrm{WR}) \wedge (\mathrm{WB})\big)$ and thus is winning for $\exists$ in $\mathcal{F}$, what contradicts the assumption that $\sigma_{\forall}$ was a winning strategy of $\forall$. ◀

Construction of automata $\mathcal{G}_K$. Take a number $K \in \omega$. We will now define a weak alternating parity automaton $\mathcal{G}_K$ of index $(0, 2)$. The language $\mathrm{L}(\mathcal{G}_K)$ will be an over-approximation of $\mathrm{L}(\mathcal{B})$. Later on we will prove that the strategy $\sigma_{\forall}$ witnesses the fact that $\mathrm{L}(\mathcal{B})$ actually equals $\mathrm{L}(\mathcal{G}_K)$ for some $K \in \omega$ (in fact for $K = K_0$ from Lemma 9).

The idea behind the automaton $\mathcal{G}_K$ is the following: $\mathcal{G}_K$ accepts a tree t if there exists a run $\rho_0^{\mathcal{B}}$ of $\mathcal{B}$ over t such that for every node u of the tree, it is possible to find another run of $\mathcal{B}$ over the subtree $t\restriction_u$ starting from the state $\rho_0^{\mathcal{B}}(u)$ that is K-accepting on every branch of this subtree.

Assume that $\mathcal{D}(K) = \langle Q^{\mathcal{B}}, Q^{\mathcal{D}}, q_1^{\mathcal{D}}, \Delta^{\mathcal{D}}, F^{\mathcal{D}} \rangle$ is a deterministic automaton over finite words (DFA) with the alphabet $Q^{\mathcal{B}}$ that recognises the language of K-accepting sequences of states of $\mathcal{B}$, see Property (P2).

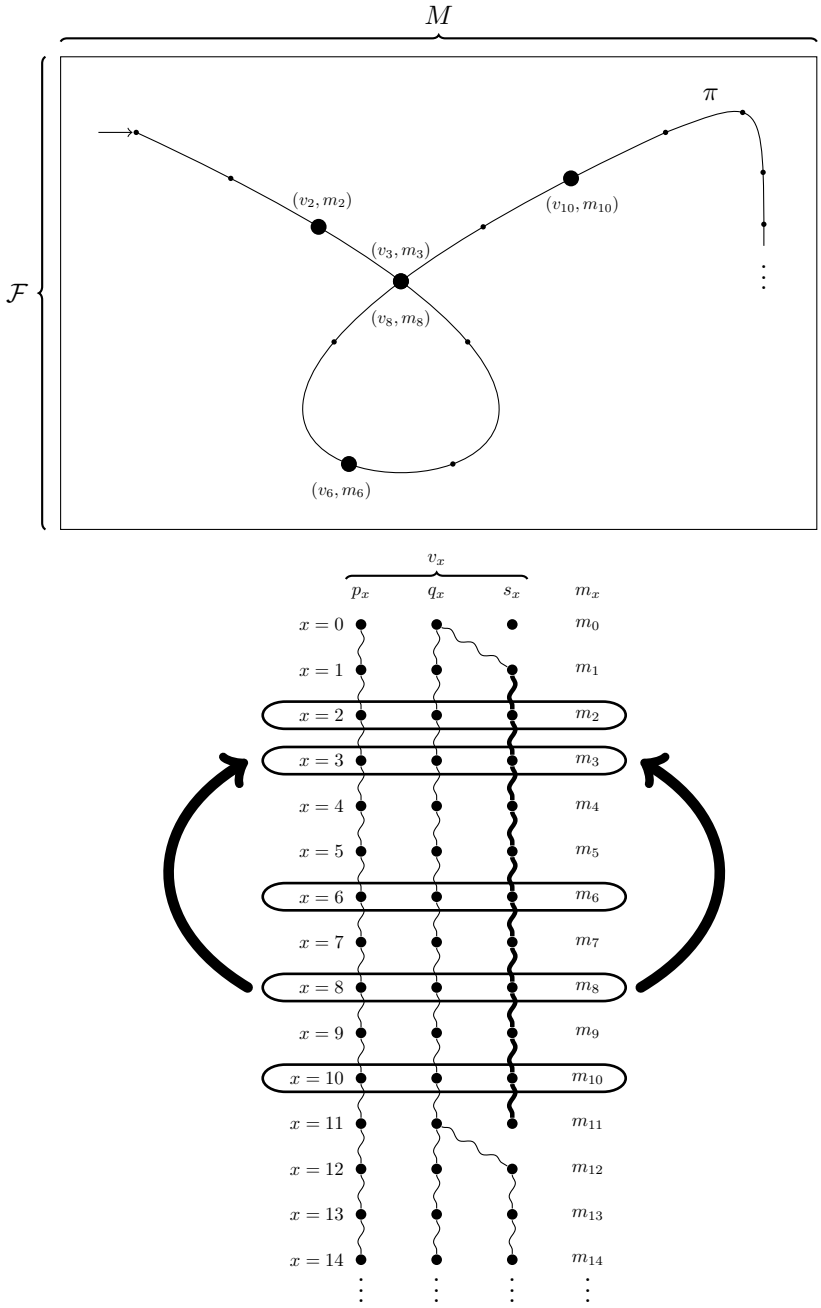


Figure 4 An illustration to the proof of Lemma 9.

The states of $\mathcal{G}_K$ are of the form (q, τ) where $q \in Q^{\mathcal{B}}$ is a state of $\mathcal{B}$ and $\tau \in \{?\} \sqcup Q^{\mathcal{D}}$ is either $?$ or a state of $\mathcal{D}(K)$.

The initial state of $\mathcal{G}_K$ is $(q_{\mathrm{I}}^{\mathcal{B}}, ?)$. The transitions of $\mathcal{G}_K$ are built by the following rules. Given a state (q, τ) and a letter a, the successive state and direction are constructed in the following way (formally the following choices should be encoded as a finite positive Boolean combination of the consecutive directions and states).

1. If $\tau = ?$ then $\forall$ can choose to *start* by letting $\tau' = q_{\mathrm{I}}^{\mathcal{D}}$ or to *skip* by keeping $\tau' = ?$. If $\tau \in Q^{\mathcal{D}}$ then $\forall$ has no choice and in that case $\tau' = \tau$.

2. If $\tau' \in Q^{\mathcal{D}}$ then we let[2] $\tau'' = \Delta^{\mathcal{D}}(\tau, q)$, otherwise $\tau'' = \tau' = \mathbf{?}$ is unchanged.
3. $\exists$ proposes a transition of $\mathcal{B}$ of the form $\big(q, a, q_{\mathtt{L}}, q_{\mathtt{R}}\big)$.
4. $\forall$ chooses a direction $d \in \{\mathtt{L}, \mathtt{R}\}$.

After these choices are done, the automaton moves in the direction d to the state (q_d, τ''). Let the priority of a state (q, τ) of $\mathcal{G}_K$ be: (i) If $\tau = \mathbf{?}$ then the priority is 0. (ii) If $\tau \in Q^{\mathcal{D}} \setminus F^{\mathcal{D}}$ then the priority is 1. (iii) If $\tau \in F^{\mathcal{D}}$ then the priority is 2.

Notice that because of the structure of the transitions of $\mathcal{G}_K$, the above defined condition is a weak parity condition of index $(0, 2)$. It is important to notice that Property (P3) implies that once a state of priority 2 is reached then we never move to a state of priority 1.

▶ **Lemma 10.** *For every $K \in \omega$ we have* $\mathrm{L}(\mathcal{B}) \subseteq \mathrm{L}(\mathcal{G}_K)$.

Proof. Take a tree $t \in \mathrm{L}(\mathcal{B})$ and let $\rho^{\mathcal{B}}$ be an accepting run of $\mathcal{B}$ on t. Then clearly $\exists$ can win the acceptance game $\mathcal{G}_K(t)$ by just playing consecutive transitions of $\rho^{\mathcal{B}}$. When $\forall$ chooses at some point to *start*, ultimately a state with $\tau \in F^{\mathcal{D}}$ will be reached because of Property (P4). Therefore, every play will be won by $\exists$. ◀

Equivalence. We will now conclude the proof of Proposition 7 using the following lemma.

▶ **Lemma 11.** *For K_0 from Lemma 9 we have* $\mathrm{L}(\mathcal{G}_{K_0}) = \mathrm{L}(\mathcal{B})$.

Proof. Assume contrarily that $\mathrm{L}(\mathcal{G}_{K_0}) \neq \mathrm{L}(\mathcal{B})$. Lemma 10 says that $\mathrm{L}(\mathcal{B}) \subseteq \mathrm{L}(\mathcal{G}_{K_0})$, so there must exists a tree $t \in \mathrm{L}(\mathcal{G}_{K_0}) \setminus \mathrm{L}(\mathcal{B})$. From that assumption we know that:

- there exists an accepting run $\rho^{\mathcal{A}}$ of $\mathcal{A}$ over t,
- $\exists$ has a winning strategy $\delta_{\exists}$ in the acceptance game $\mathcal{G}_{K_0}(t)$.

Our aim is to prove that $\exists$ can win in $\mathcal{F}$ against $\sigma_{\forall}$ by using a strategy $\sigma_{\exists}$ that is based on $\rho^{\mathcal{A}}$ and $\delta_{\exists}$. Let us define the strategy $\sigma_{\exists}$.

First, $\sigma_{\exists}$ plays the letters a and the transitions of $\mathcal{A}$ from the $\mathcal{A}$-states p according to the tree t and the run $\rho^{\mathcal{A}}$. This way we guarantee that every play of this strategy will satisfy (WA). Additionally $\sigma_{\exists}$ chooses the transitions of $\mathcal{B}$ from the $\mathcal{B}$-states q according to the strategy $\delta_{\exists}$ simulating the situation that $\forall$ has never *started*. Thus, at every moment of a play consistent with $\sigma_{\exists}$, there is a unique play of $\delta_{\exists}$ ending in a node u and a state of $\mathcal{G}_{K_0}$ of the form $(q, \mathbf{?})$ with u being the sequence of directions played so-far by $\forall$ in $\mathcal{F}$ and q being the current $\mathcal{B}$-state. What remains is the choice of transitions of $\mathcal{B}$ from the active states s. For that, $\exists$ will keep track of a play of the acceptance game $\mathcal{G}_{K_0}(t)$ with $\tau \in Q^{\mathcal{D}}$. At the initial position of $\mathcal{F}$ the play is the one which begins by $\forall$ *starting* (i.e. $\tau = q_{\mathrm{I}}^{\mathcal{D}}$). Whenever $\forall$ *restarts* in $\mathcal{F}$, $\exists$ forgets about the previously tracked play of $\mathcal{G}_{K_0}(t)$ and begins to track the play that comes with the current $\mathcal{B}$-state q by simulating the situation in which $\forall$ has just *started* in $\mathcal{G}_{K_0}(t)$.

Consider the play π that is consistent with both $\sigma_{\exists}$ and $\sigma_{\forall}$. We need to prove that π is winning for $\exists$. As we have already observed, such a play satisfies (WA). We will prove that it also satisfies (WR) by proving the following claim. This concludes the proof of Lemma 11 by giving a contradiction: $\sigma_{\forall}$ is a winning strategy of $\forall$ but π is a play consistent with $\sigma_{\forall}$ that is winning for $\exists$.

▶ **Claim 12.** *In the play π Player $\forall$ must have* restarted *infinitely many times.*

[2] We follow the transition of $\mathcal{D}$ from τ over q: $\tau \in Q^{\mathcal{D}}$ is a state of $\mathcal{D}$ and $q \in Q^{\mathcal{B}}$ is a letter read by $\mathcal{D}$.

Assume contrarily that from some point on $\forall$ has not *restarted*. Thus, π contains an infinite trace on which $\exists$ has played successive transitions of $\mathcal{B}$ in the active states s according to her strategy $\delta_\exists$ in $\mathcal{G}_{K_0}(t)$. Since the strategy $\delta_\exists$ is winning, some prefix of the considered trace must be K_0-accepting. This gives a contradiction with Lemma 9. ◀

7 Weak non-deterministic $(1,3)$-automata

In this section we prove the following additional result that may be considered folklore, although we have not found it in the literature. The construction is based on the standard de-alternation techniques together with the idea from [15].

▶ **Theorem 13.** *If L is a language that can be recognised by a weak alternating parity automaton $\mathcal{A}$ of index $(1,3)$ then L can be recognised by a weak non-deterministic parity automaton $\mathcal{B}$ of index $(1,3)$.*

This observation was important to properly define the game $\mathcal{F}$. However, somehow surprisingly, it does not play any role in the final proof of Theorem 2.

The idea of the proof is as follows. Given a tree $t \in \mathrm{Tr}_A$ the automaton $\mathcal{B}$ will guess a positional strategy of $\exists$ in the acceptance game $\mathcal{A}(t)$. Then it will verify that the guessed strategy is in fact winning. Therefore, it will track all the possible choices performed by $\forall$ along all the branches of the tree t. Thus, the set of states of $\mathcal{B}$ is the power set $\mathsf{P}(Q^{\mathcal{A}})$. Notice that the guessed strategy of $\exists$ is winning if for every branch of t the following conditions are satisfied: (i) no state of $\mathcal{A}$ of priority 3 is ever reached, (ii) every play ultimately reaches a state of priority 2.

An easy application of König's lemma shows that the second condition above actually implies that at some point no state of priority 1 can belong to the set of reachable states of $\mathcal{A}$. This way, the automaton $\mathcal{B}$ can be seen as a naïve power set construction over $\mathcal{A}$. Such a construction can be performed for any alternating automaton (even not weak), however, in most of the cases the assignment of priorities to the states of the power set automaton is not correct. The crucial ingredient of this construction relies on the fact that the weak parity condition of index $(1,3)$ admits a correct priority assignment for the power set automaton.

8 Conclusions and further work

This work provides a relatively simple effective characterisation of the class of regular languages in $\mathbf{\Pi}^0_2$. Additionally, it proves that the considered class of languages coincides with the respective level of the alternating index hierarchy (i.e. weak alternating $(0,2)$-automata).

The simplicity of involved techniques comes from certain specific properties of the considered classes. Firstly, there are ω-regular languages that are complete for the class $\mathbf{\Pi}^0_2$. In our case the examples are: the language C of infinite binary sequences containing infinitely many 1s; and the property (WR) used in the winning condition of the game $\mathcal{F}$. Secondly, similarly to the case of Büchi languages, the class of weak alternating $(1,3)$-automata admits a dealternation technique, see Theorem 13. Although this dealternation result does not play any role in the proof of Theorem 2, it was used in the design of the game $\mathcal{F}$ and stays behind the fact that the game actually characterises the class of languages recognisable by weak alternating $(0,2)$-automata.

We plan to investigate generalisations of Theorem 2 to $\mathbf{\Pi}^0_n$ for $n \geq 3$. Since decidability results related to the **Characterisation Problem** are not that easy to obtain, we propose for further investigation the topological problems related to the **Characterisation Problem** and Problem 4 formulated in Introduction.

Moreover, our work gives a quite clear situation up to the second level of Borel Hierarchy in terms of decidability and correspondence between the Borel index and the weak index, but there are still some "holes" that have to be filled regarding e.g. *Wadge Hierarchy*:

▶ **Problem 14.** *Find all Wadge degrees inhabited by regular* $\mathbf{\Delta}^0_2$ *languages of trees.*

From [6] it follows that every Wadge degree less than ω^ω is inhabited by a regular language. We believe that this is the maximum regular languages can get:

▶ **Conjecture 15.** *There is no regular tree language in* $\mathbf{\Delta}^0_2$ *with Wadge degree above* ω^ω.

References

1 Mikołaj Bojańczyk and Thomas Place. Regular languages of infinite trees that are Boolean combinations of open sets. In *ICALP*, pages 104–115, 2012.
2 Julius Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.
3 Thomas Colcombet. Fonctions régulières de coût. Habilitation thesis, Université Paris Diderot—Paris 7, 2013.
4 Thomas Colcombet, Denis Kuperberg, Christof Löding, and Michael Vanden Boom. Deciding the weak definability of Büchi definable tree languages. In *CSL*, pages 215–230, 2013.
5 Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *ICALP (2)*, pages 398–409, 2008.
6 Jacques Duparc and Filip Murlak. On the topological complexity of weakly recognizable tree languages. *Fundamentals of computation theory*, 2007.
7 Alessandro Facchini and Henryk Michalewski. Deciding the Borel complexity of regular tree languages. In *CiE 2014*, pages 163–172, 2014.
8 Alessandro Facchini, Filip Murlak, and Michał Skrzypczak. Index problems for game automata. *ACM Trans. Comput. Log.*, 17(4):24:1–24:38, 2016.
9 Tomasz Gogacz, Henryk Michalewski, Matteo Mio, and Michał Skrzypczak. Measure properties of game tree languages. In *MFCS*, pages 303–314, 2014.
10 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.
11 Alexander Kechris. *Classical descriptive set theory*. Springer-Verlag, New York, 1995.
12 Denis Kuperberg and Michael Vanden Boom. Quasi-weak cost automata: A new variant of weakness. In *FSTTCS*, volume 13 of *LIPIcs*, pages 66–77, 2011.
13 Orna Kupferman and Moshe Y. Vardi. The weakness of self-complementation. In *STACS*, pages 455–466, 1999.
14 Christof Löding. Logic and automata over infinite trees. Habilitation thesis, RWTH Aachen, Germany, 2009.
15 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theor. Comput. Sci.*, 32:321–330, 1984.
16 Filip Murlak. The Wadge hierarchy of deterministic tree languages. *Logical Methods in Computer Science*, 4(4), 2008.
17 Damian Niwiński and Igor Walukiewicz. A gap property of deterministic tree languages. *Theor. Comput. Sci.*, 1(303):215–231, 2003.
18 Damian Niwiński and Igor Walukiewicz. Deciding nondeterministic hierarchy of deterministic tree automata. *Electr. Notes Theor. Comput. Sci.*, 123:195–208, 2005.
19 Dominique Perrin and Jean-Éric Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Elsevier, 2004.

20 Michael Oser Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. of the American Math. Soc.*, 141:1–35, 1969.

21 Michael Oser Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, April 1959.

22 Michał Skrzypczak. *Descriptive Set Theoretic Methods in Automata Theory - Decidability and Topological Complexity*, volume 9802 of *Lecture Notes in Computer Science*. Springer, 2016.

23 Michał Skrzypczak and Igor Walukiewicz. Deciding the topological complexity of Büchi languages. In *ICALP (2)*, pages 99:1–99:13, 2016.

24 Jerzy Skurczyński. The Borel hierarchy is infinite in the class of regular sets of trees. *Theoretical Computer Science*, 112(2):413–418, 1993.

25 Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455. Springer, 1996.

26 Wolfgang Thomas and Helmut Lescow. Logical specifications of infinite computations. In *REX School/Symposium*, pages 583–621, 1993.

27 Boris A. Trakhtenbrot. Finite automata and the monadic predicate calculus. *Siberian Mathematical Journal*, 3(1):103–131, 1962.

On the Exact Amount of Missing Information That Makes Finding Possible Winners Hard

Palash Dey[1] and Neeldhara Misra*[2]

1 **Tata Institute of Fundamental Research, Mumbai, India**
palash.dey@tifr.res.in
2 **Indian Institute of Technology, Gandhinagar, India**
neeldhara.m@iitgn.ac.in

Abstract

We consider election scenarios with incomplete information, a situation that arises often in practice. There are several models of incomplete information and accordingly, different notions of outcomes of such elections. In one well-studied model of incompleteness, the votes are given by partial orders over the candidates. In this context we can frame the problem of finding a *possible winner*, which involves determining whether a given candidate wins in at least one completion of a given set of partial votes for a specific voting rule.

The Possible Winner problem is well-known to be NP-complete in general, and it is in fact known to be NP-complete for several voting rules where the number of undetermined pairs in every vote is bounded only by some constant. In this paper, we address the question of determining precisely the smallest number of undetermined pairs for which the Possible Winner problem remains NP-complete. In particular, we find the exact values of t for which the Possible Winner problem transitions to being NP-complete from being in P, where t is the maximum number of undetermined pairs in every vote. We demonstrate tight results for a broad subclass of scoring rules which includes all the commonly used scoring rules (such as plurality, veto, Borda, and k-approval), Copeland$^{\alpha}$ for every $\alpha \in [0, 1]$, maximin, and Bucklin voting rules. A somewhat surprising aspect of our results is that for many of these rules, the Possible Winner problem turns out to be hard even if every vote has at most one undetermined pair of candidates.

1998 ACM Subject Classification F.2.0 Analysis Of Algorithms And Problem Complexity - General

Keywords and phrases Computational Social Choice, Dichotomy, NP-completeness, Maxflow, Voting, Possible winner

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.57

1 Introduction

In many real life situations including multiagent systems, agents often need to aggregate their preferences and agree upon a common decision (candidate). Voting is an immediate natural tool in these situations. Common and classic applications of voting in multiagent systems include collaborative filtering and recommender systems [26], spam detection [9], computational biology [20], winner determination in sports competition [5] etc. We refer the readers to [25] for an elaborate treatment of computational voting theory.

Usually, in a voting setting, it is assumed that the votes are complete orders over the candidates. However, due to many reasons, for example, lack of knowledge of voters about

* The second author acknowledges support from the DST-INSPIRE faculty grant (IFA12-ENG-31).

some candidates, a voter may be indifferent between some pairs of candidates. Hence, it is both natural and important to consider scenarios where votes are partial orders over the candidates. When votes are only partial orders over the candidates, the winner cannot be determined with certainty since it depends on how these partial orders are extended to linear orders. This leads to a natural computational problem called the Possible Winner problem [21]: given a set of partial votes $\mathcal{P}$ and a distinguished candidate c, is there a way to extend the partial votes to complete votes where c wins? The Possible Winner problem has been studied extensively in the literature [23, 27, 28, 29, 7, 8, 4, 1, 22, 18] following its definition in [21]. Betzler et al. [6] and Baumeister et al. [2] show that the Possible Winner winner problem is NP-complete for all scoring rules except for the plurality and veto voting rules; the Possible Winner winner problem is in P for the plurality and veto voting rules. The Possible Winner problem is known to be NP-complete for many common voting rules, for example, a class of scoring rules, maximin, Copeland, Bucklin etc. even when the maximum number of undetermined pairs of candidates in every vote is bounded above by small constants [29]. Walsh showed that the Possible Winner problem can be solved in polynomial time for all the voting rules mentioned above when we have a constant number of candidates [28].

1.1 Our Contribution

Our main contribution lies in pinning down exactly the minimum number of undetermined pairs allowed per vote so that the Possible Winner winner problem continues to be NP-complete for a large class of scoring rules, Copeland$^\alpha$, maximin, and Bucklin voting rules. To begin with, we describe our results for scoring rules. We work with a class of scoring rules that we call smooth, which are essentially scoring rules where the score vector for $(m+1)$ candidates can be obtained by either duplicating an already duplicated score in the score vector for m candidates, or by extending the score vector for m candidates at one of the endpoints with an arbitrary new value. The smooth rules account for all commonly used scoring rules (such as Borda, plurality, veto, k-approval). Using t to denote the maximum number of undetermined pairs of candidates in every vote, we show the following (note that the Possible Winner problem is in P for all scoring rules when $t = 0$):

- The Possible Winner problem is NP-complete even when $t = 1$ for scoring rules which have two distinct nonzero differences between consecutive coordinates in the score vector (we call them differentiating) and in P when $t = 1$ for other scoring rules [Theorem 7].
- Else the Possible Winner problem is NP-complete when $t \geqslant 2$ and in P when $t \leqslant 1$ for scoring rules that contain $(\alpha+1, \alpha+1, \alpha)$ for any $\alpha \in \mathbb{N}$ [Theorem 8].
- Else the Possible Winner problem is NP-complete when $t \geqslant 3$ and in P when $t \leqslant 2$ for scoring rules which contain $(\alpha+2, \alpha+1, \alpha+1, \alpha)$ for any $\alpha \in \mathbb{N}$ [Theorem 9].
- The Possible Winner problem is NP-complete when $t \geqslant 4$ and in P when $t \leqslant 3$ for k-approval and k-veto voting rules for any $k > 1$ [Theorem 10].
- The Possible Winner problem is NP-complete when $t \geqslant m-1$ and in P when $t \leqslant m-2$ for the scoring rule $(2, 1, 1, \ldots, 1, 0)$ [Theorem 10].

We summarize our results for the Copeland$^\alpha$, maximin, and Bucklin voting rules in Table 1. We observe that the Possible Winner problem for the Copeland$^\alpha$ voting rule is NP-complete even when every vote has at most 2 undetermined pairs of candidates for $\alpha \in \{0, 1\}$. However, for $\alpha \in (0, 1)$, the Possible Winner problem for the Copeland$^\alpha$ voting rule is NP-complete even when every vote has at most 1 undetermined pairs of candidates. Our results show that the Possible Winner winner problem continues to be NP-complete

Table 1 Summary and comparison of results from the literature for Copeland$^{\alpha}$, maximin, and Bucklin voting rules. *The result was proved for the simplified Bucklin voting rule but the proof can be modified easily for the Bucklin voting rule.

Voting rules	NP-complete	Poly time	Known from literature [29]
Copeland^{0,1}	$t \geqslant 2$ [Theorem 11]	$t \leqslant 1$ [Theorem 12]	NP-complete for $t \geqslant 8$
Copeland$^{\alpha}$ $\alpha \in (0,1)$	$t \geqslant 1$ [Theorem 15]	–	
Maximin	$t \geqslant 2$ [Theorem 17]	$t \leqslant 1$ [Theorem 18]	NP-complete for $t \geqslant 4$
Bucklin	$t \geqslant 2$ [Theorem 19]	$t \leqslant 1$ [Theorem 19]	NP-complete for $t \geqslant 16^{\star}$

for all the common voting rules studied here (except k-approval) even when the number of undetermined pairs of candidates per vote is at most 2. Other than finding the exact number of undetermined pairs needed per vote to make the Possible Winner problem NP-complete for common voting rules, we also note that all our proofs are much simpler and shorter than most of the corresponding proofs from the literature subsuming the work in [29, 6, 2].

2 Preliminaries

Let us denote the set $\{1, 2, \ldots, n\}$ by $[n]$ for any positive integer n. Let $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ be a set of candidates or alternatives and $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ a set of voters. If not mentioned otherwise, we denote the set of candidates by $\mathcal{C}$, the set of voters by $\mathcal{V}$, the number of candidates by m, and the number of voters by n. Every voter v_i has a preference or vote $\succ_i$ which is a complete order over $\mathcal{C}$. We denote the set of complete orders over $\mathcal{C}$ by $\mathcal{L}(\mathcal{C})$. We call a tuple of n preferences $(\succ_1, \succ_2, \cdots, \succ_n) \in \mathcal{L}(\mathcal{C})^n$ an n-voter preference profile. An election is defined as a set of candidates together with a voting profile. It is often convenient to view a preference as a subset of $\mathcal{C} \times \mathcal{C}$ — a preference $\succ$ corresponds to the subset $\mathcal{A} = \{(x, y) \in \mathcal{C} \times \mathcal{C} : x \succ y\}$. For a preference $\succ$ and a subset $\mathcal{A} \subseteq \mathcal{C}$ of candidates, we define $\succ(\mathcal{A})$ be the preference $\succ$ restricted to $\mathcal{A}$, that is $\succ(\mathcal{A}) = \succ \cap (\mathcal{A} \times \mathcal{A})$. Let $\uplus$ denote the disjoint union of sets. A map $r : \uplus_{n, |\mathcal{C}| \in \mathbb{N}^+} \mathcal{L}(\mathcal{C})^n \longrightarrow 2^{\mathcal{C}} \setminus \{\emptyset\}$ is called a *voting rule*. For a voting rule r and a preference profile $\succ = (\succ_1, \ldots, \succ_n)$, we say a candidate x wins uniquely if $r(\succ) = \{x\}$ and x co-wins if $x \in r(\succ)$. For a vote $\succ \in \mathcal{L}(\mathcal{C})$ and two candidates $x, y \in \mathcal{C}$, we say x is placed before y in $\succ$ if $x \succ y$; otherwise we say x is placed after y in $\succ$. A candidate is said to be at the i^{th} position from the top (bottom) if there are $(i-1)$ candidates after (before) it. For any two candidates $x, y \in \mathcal{C}$ with $x \neq y$ in an election $\mathcal{E}$, let us define the margin $\mathcal{D}_{\mathcal{E}}(x, y)$ of x from y to be $|\{i : x \succ_i y\}| - |\{i : y \succ_i x\}|$. Examples of some common voting rules are as follows.

Positional scoring rules. A collection $(\overrightarrow{s_m})_{m \in \mathbb{N}^+}$ of m-dimensional vectors $\overrightarrow{s_m} = (\alpha_m, \alpha_{m-1}, \ldots, \alpha_1) \in \mathbb{N}^m$ with $\alpha_m \geqslant \alpha_{m-1} \geqslant \ldots \geqslant \alpha_1$ and $\alpha_m > \alpha_1$ for every $m \in \mathbb{N}^+$ naturally defines a voting rule — a candidate gets score α_i from a vote if it is placed at the i^{th} position from the bottom, and the score of a candidate is the sum of the scores it receives from all the votes. The winners are the candidates with maximum score. Scoring rules remain unchanged if we multiply every α_i by any constant $\lambda > 0$ and/or add any constant μ. Hence, we assume without loss of generality that for any score vector $\overrightarrow{s_m}$, there exists a j such that $\alpha_k = 0$ for all $k < j$ and the greatest common divisor of $\alpha_1, \ldots, \alpha_m$ is one. Such a $\overrightarrow{s_m}$ is called a normalized score vector. Without loss of generality, we will work with

normalized scoring rules only in this work. If α_i is 0 for $i \in [m-k]$ and 1 otherwise, then we get the k-approval voting rule. For the k-veto voting rule, α_i is 0 for $i \in [k]$ and 1 otherwise. 1-approval is called the plurality voting rule and 1-veto is called the veto voting rule. We note that our notation is slightly unconventional, this is in the interest of convenience in some of the computations that we will encounter with score vectors.

Copeland$^\alpha$. Given $\alpha \in [0,1]$, the Copeland$^\alpha$ score of a candidate x is $|\{y \neq x : \mathcal{D}_\mathcal{E}(x,y) > 0\}| + \alpha|\{y \neq x : \mathcal{D}_\mathcal{E}(x,y) = 0\}|$. The winners are the candidates with maximum Copeland$^\alpha$ score. If not mentioned otherwise, we will assume α to be zero.

Maximin. The maximin score of a candidate x in an election E is $\min_{y \neq x} \mathcal{D}_\mathcal{E}(x,y)$. The winners are the candidates with maximum maximin score.

Bucklin. Let ℓ be the minimum integer such that there exists at least one candidate $x \in \mathcal{C}$ whom more than half of the voters place in their top ℓ positions. Then the Bucklin winner is the candidate who is placed most number of times within top ℓ positions of the votes.

Elections with Incomplete Information A more general setting is an *election* where the votes are only *partial orders* over candidates. A *partial order* is a relation that is *reflexive, antisymmetric*, and *transitive*. A partial vote can be extended to possibly more than one linear vote depending on how we fix the order for the unspecified pairs of candidates. Given a partial vote $\succ$, we say that an extension $\succ'$ of $\succ$ *places the candidate c as high as possible* if $a \succ' c$ implies $a \succ'' c$ for every extension $\succ''$ of $\succ$.

▶ **Definition 1** (r–Possible Winner)**.** Given a set of partial votes $\mathcal{P}$ over a set of candidates $\mathcal{C}$ and a candidate $c \in \mathcal{C}$, does there exist an extension $\mathcal{P}'$ of $\mathcal{P}$ such that $c \in r(\mathcal{P}')$?

3 Results

For ease of exposition, we present all our results for the co-winner case. All our proofs extend easily to the unique winner case too. We begin with our results for the scoring rules.

3.1 Scoring Rules

In this section, we establish a dichotomous result describing the status of the Possible Winner problem for a large class of scoring rules when the number of undetermined pairs in every vote is at most one, two, three, or four. We begin by introducing some terminology. Instead of working directly with score vectors, it will sometimes be convenient for us to refer to the "vector of differences", which, for a score vector s with m coordinates, is a vector $d(s)$ with $m-1$ coordinates with each entry being the difference between adjacent scores corresponding to that location and the location left to it. This is formally stated below.

▶ **Definition 2.** Given a normalized score vector $\overrightarrow{s_m} = (\alpha_m, \alpha_{m-1}, \ldots, \alpha_1 = 0) \in \mathbb{N}^m$, the associated difference vector $d(\overrightarrow{s_m})$ is given by $(\alpha_m - \alpha_{m-1}, \alpha_{m-1} - \alpha_{m-2}, \ldots, \alpha_2 - \alpha_1) \in \mathbb{N}^{m-1}$. We also employ the following notation to refer to the smallest score difference among all non-zero differences, and the largest score difference, respectively:

- $\delta(\overrightarrow{s_m}) = \min(\{\alpha_i - \alpha_{i-1} \mid 2 \leqslant i \leqslant m \text{ and } \alpha_i - \alpha_{i-1} > 0\})$
- $\Delta(\overrightarrow{s_m}) = \max(\{\alpha_i - \alpha_{i-1} \mid 2 \leqslant i \leqslant m\})$

Note that for every non-trivial normalized score vector $\overrightarrow{s_m}$, $\Delta(\overrightarrow{s_m})$ is always non-zero. We now proceed to defining the notion of smooth scoring rules. Consider a score vector $\overrightarrow{s_m} = (\alpha_m, \alpha_{m-1}, \ldots, \alpha_1)$. For $0 \leqslant i \leqslant m$, we say that $\overrightarrow{s_{m+1}}$ is obtained from s_m by inserting α just before position i from the right if:

$$\overrightarrow{s_{m+1}} = (\alpha_m, \alpha_{m-1}, \ldots, \alpha_{i+1}, \alpha, \alpha_i, \ldots, \alpha_2, \alpha_1)\,.$$

Note that if $i = 0$, we have $\overrightarrow{s_{m+1}} = (\alpha_m, \alpha_{m-1}, \ldots, \alpha_1, \alpha)$, and if $i = m$, then we have $\overrightarrow{s_{m+1}} = (\alpha, \alpha_m, \alpha_{m-1}, \ldots, \alpha_1)$. For $0 \leqslant i \leqslant m$, we say that the position i is *admissible* if $i = 0$, or $i = m$, or $\alpha_{i+1} = \alpha_i$.

▶ **Definition 3** (Smooth scoring rules)**.** We say that a scoring rule s is smooth if there exists some constant $m_0 \in \mathbb{N}^+$ such that for all $m \geqslant m_0$, the score vector $\overrightarrow{s_m}$ can be obtained from $\overrightarrow{s_{m-1}}$ by inserting an additional score value at any position i that is admissible.

Observe that the additional score value is forced to be equal to an existing score value unless it is inserted at one of the endpoints. Intuitively speaking, a smooth scoring rule is one where the score vector for m candidates can be obtained by either extending the one for $(m-1)$ candidates at one of the ends, or by inserting a score between an adjacent pair of ambivalent locations (i.e, consecutive scores in the score vector with the same value). Although at a first glance it may seem that the class of smooth scoring rules involves an evolution from a limited set of operations, we note that all of the common scoring rules, such as plurality, veto, k-approval, Borda, and scoring rules of the form $(2, 1, \ldots, 1, 0)$, are smooth. We now turn to some definitions that will help describe the cases that appear in our classification result.

▶ **Definition 4.** Let $s = (\overrightarrow{s_m})_{m \in \mathbb{N}^+}$ be a scoring rule.

- We say that s is a *Borda-like scoring rule* if there exists some $m_0 \in \mathbb{N}^+$ for which we have that $\Delta(\overrightarrow{s_m}) = \delta(\overrightarrow{s_m})$ for every $m > m_0$.
- Any rule that is not Borda-like is called a *differentiating scoring rule*.
- For any vector t with ℓ coordinates, we say that s is t-difference-free if there exists some $n_0 \in \mathbb{N}^+$ such that for every $m \geqslant n_0$, the vector t does not occur in $d(\overrightarrow{s_m})$. In other words, the vector $\langle d(\overrightarrow{s_m})[i], \ldots, d(\overrightarrow{s_m})[i+\ell-1]\rangle \neq t$ for any $1 \leqslant i \leqslant m - \ell$.
- For any vector t, we say that s is t-contaminated if it is not t-difference-free. We also say that s is t-contaminated at m if the vector t occurs in $d(\overrightarrow{s_m})$.

We will frequently be dealing with Borda-like score vectors. To this end, the following easy observation will be useful.

▶ **Observation 5.** *If $s = (\overrightarrow{s_m})_{m \in \mathbb{N}^+}$ is a Borda-like scoring rule in its normalized form, then there exists $n_0 \in \mathbb{N}^+$ such that all the coordinates of $d(\overrightarrow{s_m})$ are either zero or one for all $m > n_0$.*

It turns out that if a scoring rule is smooth, then its behavior with respect to some of the properties above is fairly monotone. For instance, we have the following easy proposition. For the interest of space, we omit proofs of some of our results including all our polynomial time algorithms. For a few proofs, we only provide a sketch of the proof deferring the complete proof to the appendix. We mark these results with $\star$. All our polynomial time algorithms are based on reduction to the maximum flow problem in a graph. All the complete proofs can be found here [12].

[$\star$] Let $s = (\overrightarrow{s_m})_{m \in \mathbb{N}^+}$ be a smooth scoring rule that is not Borda-like. Then there exists some $n_0 \in \mathbb{N}^+$ such that $\Delta(\overrightarrow{s_m}) \neq \delta(\overrightarrow{s_m})$ for every $m > n_0$.

We are now ready to state the first classification result of this section, for the scenario where every vote has at most one missing pair. We use $(3, B2)$–SAT to prove some of our hardness results. The $(3, B2)$–SAT problem is the 3-SAT problem restricted to formulas in which each clause contains exactly three literals, and each variable occurs exactly twice positively and twice negatively. We know that $(3, B2)$–SAT is NP-complete [3]. Let us first present a structural result for scoring rules which we will use subsequently.

Suppose we have a set $\mathcal{C} = \{c_1, \ldots, c_{m-1}, g\}$ of m candidates including a "dummy" candidate g. Then it is well known [1, 16, 14, 13, 15, 17, 11], that for a score vector $(\alpha_m, \ldots, \alpha_1)$ and integers $\{k_i^j\}_{i\in[m-1],j\in[m-1]}$, we can add votes polynomially many in $\sum_{i\in[m-1],j\in[m-1]} k_i^j$ so that the score of the candidate c_i is $\lambda + \sum_{j\in[m-1]} k_i^j(\alpha_j - \alpha_{j+1})$ for some λ and the score of g is less than λ. Since the greatest common divisor of non-zero differences of the consecutive entries in a normalized score vector is one, we have the following.

▶ **Lemma 6.** *Let $\mathcal{C} = \{c_1, \ldots, c_m\} \cup D, (|D| > 0)$ be a set of candidates, and $\vec{\alpha}$ a normalized score vector of length $|\mathcal{C}|$. Then for every $\mathbf{X} = (X_1, \ldots, X_m) \in \mathbb{Z}^m$, there exists $\lambda \in \mathbb{N}$ and a voting profile $\mathcal{V}$ such that the $\vec{\alpha}$-score of c_i is $\lambda + X_i$ for all $1 \leqslant i \leqslant m$, and the score of candidates $d \in D$ is less than λ. Moreover, the number of votes in $\mathcal{V}$ is $O(poly(|\mathcal{C}| \cdot \sum_{i=1}^{m} |X_i|))$.*

▶ **Theorem 7.** *[⋆] Let s be a smooth scoring rule. If s is differentiating, then the* Possible Winner *problem is* NP*-complete, even if every vote has at most one undetermined pair of candidates. Otherwise, that is, when s is Borda-like, the* Possible Winner *problem for s is in* P *if every vote has at most one undetermined pair of candidates.*

Proof. *(Outline.)* For the hardness result, we reduce from an instance of $(3, B2)$-SAT. Let $\mathcal{I}$ be an instance of $(3, B2)$-SAT, over the variables $\mathcal{V} = \{x_1, \ldots, x_n\}$ and with clauses $\mathcal{T} = \{c_1, \ldots, c_t\}$. To construct the reduced instance $\mathcal{I}'$, we introduce two candidates for every variable, and one candidate for every clause, one special candidate w, and a dummy candidate g to achieve desirable score differences. Notationally, we will use b_i (corresponding to x_i) and b_i' (corresponding to $\bar{x}_i$) to refer to the candidates based on the variable x_i and e_j to refer to the candidate based on the clause c_j. To recap, the set of candidates are given by:

$$\mathcal{C} = \{b_i, b_i' \mid x_i \in \mathcal{V}\} \cup \{e_j \mid c_j \in \mathcal{T}\} \cup \{w, g\}.$$

Consider an arbitrary but fixed ordering over $\mathcal{C}$, such as the lexicographic order. In this proof, the notation $\overrightarrow{\mathcal{C}'}$ for any $\mathcal{C}' \subseteq \mathcal{C}$ will be used to denote the lexicographic ordering restricted to the subset $\mathcal{C}'$. Let m denote $|\mathcal{C}| = 2n + t + 2$, and let $\overrightarrow{s_m} = (\alpha_m, \alpha_{m-1}, \ldots, \alpha_1) \in \mathbb{N}^m$. Since s is a smooth differentiating scoring rule, we have that there exist $1 \leqslant p, q \leqslant m$ such that $|p - q| > 1$ and $\alpha_p - \alpha_{p-1} > \alpha_q - \alpha_{q-1} \geqslant 1$.

We use D to refer to the larger of the two differences above, namely $\alpha_p - \alpha_{p-1}$ and d to refer to $\alpha_q - \alpha_{q-1}$. We now turn to a description of the votes. Fix an arbitrary subset $\mathcal{C}_1$ of $(m - p)$ candidates. For every variable $x_i \in \mathcal{V}$, we introduce the following complete and partial votes.

$$\mathfrak{p}_i := \overrightarrow{\mathcal{C}_1} \succ b_i \succ b_i' \succ \overrightarrow{\mathcal{C} \setminus \mathcal{C}_1} \text{ and } \mathfrak{p}_i' := \mathfrak{p}_i \setminus \{(b_i, b_i')\}$$

We next fix an arbitrary subset $\mathcal{C}_2 \subset \mathcal{C}$ of $(m - q)$ candidates. Consider a literal ℓ corresponding to the variable x_i. We use $\ell^\star$ to refer to the candidate b_j if the literal is positive and to refer to the candidate b_j' if the literal is negated. For every clause $c_j \in \mathcal{T}$ given by $c_j = \{\ell_1, \ell_2, \ell_3\}$, we introduce the following complete and partial votes.

$$\mathfrak{q}_{j,1} := \overrightarrow{\mathcal{C}_2} \succ e_j \succ \ell_1^\star \succ \overrightarrow{\mathcal{C} \setminus \mathcal{C}_2} \text{ and } \mathfrak{q}_{j,1}' := \mathfrak{q}_{j,1} \setminus \{(e_j, \ell_1^\star)\}$$

Table 2 Score of candidates from $\mathcal{P} \cup \mathcal{W}$.

$s^+(e_j) = s^+(w) + d \; \forall \; 1 \leqslant j \leqslant t$ $s^+(b'_i) = s^+(w) + 1 - d - D \; \forall \; 1 \leqslant i \leqslant n$	$s^+(b_i) = s^+(w) + 1 - d \; \forall \; 1 \leqslant i \leqslant n$ $s^+(g) < s^+(w)$

$\mathfrak{q}_{j,2} := \overrightarrow{\mathcal{C}_2} \succ e_j \succ \ell_2^\star \succ \overrightarrow{\mathcal{C} \setminus \mathcal{C}_2}$ and $\mathfrak{q}'_{j,2} := \mathfrak{q}_{j,2} \setminus \{(e_j, \ell_2^\star)\}$

$\mathfrak{q}_{j,3} := \overrightarrow{\mathcal{C}_2} \succ e_j \succ \ell_3^\star \succ \overrightarrow{\mathcal{C} \setminus \mathcal{C}_2}$ and $\mathfrak{q}'_{j,3} := \mathfrak{q}_{j,3} \setminus \{(e_j, \ell_3^\star)\}$

Let us define the following sets of votes:

$$\mathcal{P} = \left(\bigcup_{i=1}^{n} \mathfrak{p}_i\right) \cup \left(\bigcup_{1 \leqslant j \leqslant t, 1 \leqslant b \leqslant 3} \mathfrak{q}_{j,b}\right) \text{ and } \mathcal{P}' = \left(\bigcup_{i=1}^{n} \mathfrak{p}'_i\right) \cup \left(\bigcup_{1 \leqslant j \leqslant t, 1 \leqslant b \leqslant 3} \mathfrak{q}'_{j,b}\right)$$

There exists a set of complete votes $\mathcal{W}$ of size polynomial in m with the following properties due to Lemma 6. Let $s^+ : \mathcal{C} \longrightarrow \mathbb{N}$ be a function mapping candidates to their scores from the set of votes $\mathcal{P} \cup \mathcal{W}$. Then $\mathcal{W}$ can be constructed to ensure the scores as in Table 2. We now define the instance $\mathcal{I}'$ of Possible Winner to be $(\mathcal{C}, \mathcal{P}' \cup \mathcal{W}, w)$. This completes the description of the reduction. We now turn to a proof of the equivalence. Before we begin making our arguments, observe that since w does not participate in any undetermined pairs of the votes in $\mathcal{P}'$, it follows that the score of w continues to be $s^+(w)$ in any completion of $\mathcal{P}'$. The intuition for the construction, described informally, is as follows. The score of every "clause candidate" needs to decrease by d, which can be achieved by pushing it down against its literal partner in the $\mathfrak{q}_j$-votes. However, this comes at the cost of increasing the score of the literals by $2d$ (since every literal appears in at most two clauses). It turns out that this can be compensated appropriately by ensuring that the candidate corresponding to the literal appears in the $(p-1)^{th}$ position among the $\mathfrak{p}$-votes, which will adjust for this increase. Therefore, the setting of the (b'_i, b_i) pairs in a successful completion of $\mathfrak{p}_i$ can be read off as a signal for how the corresponding variable should be set by a satisfying assignment. We defer the formal proof of equivalence of the two instances and the polynomial time solvable case to the appendix. ◀

We make a couple of quick remarks before moving on to our next result. Observe that any hardness result that holds for instances where every vote has at most k undetermined pairs also holds for instances where every vote has at most k' undetermined pairs with $k' > k$, by a standard special case argument. Therefore, the next question for us to address is that of whether the Possible Winner problem is in P for all Borda-like scoring rules when the number of undetermined pairs in every vote is at most two. We show that the complexity of the Possible Winner problem for the Borda-like scoring rules crucially depends on the presence (or absence) some particular patterns in the score vector. We begin by stating a hardness result which uses a reduction from the well-known Three Dimensional Matching problem [19].

To help us deal with the nature of the score vectors considered, we will use the following proposition, which again reflects the monotonicity property alluded to earlier.

[⋆] Let s be a normalized smooth scoring rule that is not $\langle 1, 1 \rangle$-difference-free. Then there exists some $n_0 \in \mathbb{N}^+$ such that for every $m \geqslant n_0$, s is $\langle 1, 1 \rangle$-contaminated at m.

We are now ready to state our next result, which shows that if there are at most 2 undetermined pairs of candidates in every vote, and we are dealing with a smooth Borda-like scoring rule s, then the Possible Winner problem is NP-complete if s is $\langle 1, 1 \rangle$-contaminated, and solvable in polynomial time otherwise.

▶ **Theorem 8.** *[⋆] Let s be a smooth, Borda-like scoring rule. If s is $\langle 1,1\rangle$-contaminated, the* Possible Winner *problem is* NP*-complete, even if every vote has at most* 2 *undetermined pairs of candidates. On the other hand, if s is $\langle 1,1\rangle$-difference-free, then the* Possible Winner *problem for s is in* P *if every vote has at most* 2 *undetermined pairs of candidates.*

We now address the case involving at most three undetermined pairs in every vote. The interesting scoring rules here are smooth Borda-like scoring rules that are $\langle 1,1\rangle$-difference-free. It turns out that here, if the scoring rule is further $\langle 1,0,1\rangle$-difference-free, then the problem again admits a maxflow formulation. On the other hand, s is $\langle 1,0,1\rangle$-contaminated at $m \geqslant N_0$ for some constant N_0, then the Possible Winner problem is NP-complete even with 3 undetermined pairs of candidates per vote.

▶ **Theorem 9.** *[⋆] Let s be a smooth, Borda-like, $\langle 1,1\rangle$-difference-free scoring rule. If there exists a constant $N_0 \in \mathbb{N}^+$ such that s is $\langle 1,0,1\rangle$-contaminated for all $m \geqslant n_0$, then the* Possible Winner *problem is* NP*-complete, even if every vote has at most* 3 *undetermined pairs. On the other hand, if s is $\langle 1,0,1\rangle$-difference-free, then the* Possible Winner *problem for s is in* P *if every vote has at most* 3 *undetermined pairs.*

▶ Remark. Note that unlike the previous two results, this statement is not a complete classification, because we don't have an appropriate analog of Propositions 3.1 and 3.1. Having said that, our result holds for a more general class of scoring rules: those where s is $\langle 1,0,1\rangle$-contaminated at m "sufficiently" often, that is to say that if $\overrightarrow{s_m}$ is $\langle 1,0,1\rangle$-contaminated and $m' > m$ is the smallest natural number for which $\overrightarrow{s_m}$ is $\langle 1,0,1\rangle$-contaminated, then $m' - m$ is bounded by some polynomial function of m, by inserting appropriately many dummy candidates using standard techniques.

We now turn to our final result for scoring rules. Let s be a smooth, Borda-like scoring rule that is $\langle 1,1\rangle$-difference-free. Then we have the following. If s is $\langle 0,1,0\rangle$-contaminated, then the Possible Winner problem for s is NP-complete even when every vote has at most 4 undetermined pairs of candidates. If s is $\langle 0,1,0\rangle$-difference-free, then notice that $d(\overrightarrow{s_m})$ for any suitably large $m \in \mathbb{N}^+$ can contain at most two ones (since s is also $\langle 1,1\rangle$-difference-free). If the number of ones in $d(\overrightarrow{s_m})$ is one, then $d(\overrightarrow{s_m})$ either has a one on the first or the last coordinate (recall that s is $\langle 0,1,0\rangle$-difference-free), corresponding to the plurality and veto voting rules, respectively. On the other hand, if the number of ones is two, $d(\overrightarrow{s_m}) = \langle 1,0,\ldots,0,1\rangle$, which is equivalent (in normal form) to the scoring rule $(2,1,\ldots,1,0)$. The Possible Winner problem is polynomial time solvable for plurality and veto voting rules, and we show here that it is also polynomially solvable for the scoring rule $(2,1,\ldots,1,0)$ as long as the number of undetermined pairs of candidates in any vote is at most $m-1$. We note that the status for the Possible Winner problem for this rule was left unresolved in [6] and was later resolved in [2]. If we allow for m or more undetermined pairs of candidates in every vote, then we show that the Possible Winner problem is NP-complete. As before, we will need the following property of $\langle 1,0,1\rangle$-contaminated vectors.

[⋆] Let s be a normalized smooth scoring rule that is not $\langle 1,0,1\rangle$-difference-free. Then there exists some $n_0 \in \mathbb{N}^+$ such that s is $\langle 0,1,0\rangle$-contaminated at m for every $m > n_0$.

We now state the final result in this section. It is easily checked that the result accounts for all smooth, Borda-like scoring rules that are $\langle 1,1\rangle$-difference-free.

▶ **Theorem 10.** *[⋆] Let s be a smooth, Borda-like scoring rule that is $\langle 1,1\rangle$-difference-free. Then we have the following.*

1. *If s is $\langle 0,1,0\rangle$-contaminated, then the* Possible Winner *problem for s is* NP*-complete even when every vote has at most* 4 *undetermined pairs of candidates.*

2. *If s is equivalent to $(2,1,\ldots,1,0)$, then* Possible Winner *is* NP*-complete even when the number of undetermined pairs of candidates in every vote is at most $m-1$.*
3. *If s is equivalent to $(2,1,\ldots,1,0)$ and the number of undetermined pairs of candidates is strictly less than $m-1$, then* Possible Winner *is in* P*.*
4. *If s is neither $\langle 0,1,0\rangle$-contaminated nor equivalent to $(2,1,\ldots,1,0)$, then s is equivalent to either the plurality or veto scoring rules and* Possible Winner *is in* P *for these cases.*

3.2 Copeland$^\alpha$ Voting Rule

We now turn to the Copeland$^\alpha$ voting rule. We show in Theorem 11 below that the Possible Winner problem is NP-complete for the Copeland$^\alpha$ voting rule even when every vote has at most 2 undetermined pairs of candidates for every $\alpha \in [0,1]$.

▶ **Theorem 11.** *[⋆] The* Possible Winner *problem is* NP*-complete for the Copeland$^\alpha$ voting rule even if the number of undetermined pairs of candidates in every vote is at most* 2 *for every $\alpha \in [0,1]$.*

We prove in Theorem 12 that the number of undetermined pairs of candidates in Theorem 11 is tight for the Copeland^0 and Copeland^1 voting rules.

▶ **Theorem 12.** *[⋆] The* Possible Winner *problem is in* P *for the Copeland^0 and Copeland^1 voting rules if the number of undetermined pairs of candidates in every vote is at most* 1*.*

We show next that the Possible Winner problem is NP-complete for the Copeland$^\alpha$ voting rule even if the number of undetermined pairs of candidates in every vote is at most 1 for $\alpha \in (0,1)$. We break the proof into two parts — Lemma 13 proves the result for every $\alpha \in (0, 1/2]$ and Lemma 14 proves for every $\alpha \in [1/2, 1)$.

▶ **Lemma 13.** *[⋆] The* Possible Winner *problem is* NP*-complete for the Copeland$^\alpha$ voting rule even if the number of undetermined pairs in every vote is at most* 1 *for every $\alpha \in (0, 1/2]$.*

Proof. The Possible Winner problem for the Copeland$^\alpha$ voting rule is clearly in NP. To prove NP-hardness of Possible Winner, we reduce Possible Winner from $(3,B2)$–SAT. Let $\mathcal{I}$ be an instance of $(3,B2)$–SAT, over the variables $\mathcal{V} = \{x_1,\ldots,x_n\}$ and with clauses $\mathcal{T} = \{c_1,\ldots,c_m\}$. We construct an instance $\mathcal{I}'$ of Possible Winner from $\mathcal{I}$ as follows.

Set of candidates: $\mathcal{C} = \{x_i, \bar{x}_i, d_i : i \in [n]\} \cup \{c_i : i \in [m]\} \cup \{c\} \cup \mathcal{G}$, where $\mathcal{G} = \{g_1,\ldots,g_{mn}\}$.

For every $i \in [n]$, let us define $\mathfrak{p}^1_{x_i}, \mathfrak{p}^2_{x_i} : x_i \succ d_i \succ$ others and $\mathfrak{p}^1_{\bar{x}_i}, \mathfrak{p}^2_{\bar{x}_i} : \bar{x}_i \succ d_i \succ$ others. Using $\mathfrak{p}^1_{x_i}, \mathfrak{p}^2_{x_i}, \mathfrak{p}^1_{\bar{x}_i}, \mathfrak{p}^2_{\bar{x}_i}$, we define the partial votes $\mathfrak{p}^{1\prime}_{x_i}, \mathfrak{p}^{2\prime}_{x_i}, \mathfrak{p}^{1\prime}_{\bar{x}_i}, \mathfrak{p}^{2\prime}_{\bar{x}_i}$ as follows.

$$\mathfrak{p}^{1\prime}_{x_i}, \mathfrak{p}^{2\prime}_{x_i} : \mathfrak{p}^1_{x_i} \setminus \{(x_i, d_i)\} \ , \ \mathfrak{p}^{1\prime}_{\bar{x}_i}, \mathfrak{p}^{2\prime}_{\bar{x}_i} : \mathfrak{p}^1_{\bar{x}_i} \setminus \{(\bar{x}_i, d_i)\}$$

Let a clause c_j involves the literals $\ell^1_j, \ell^2_j, \ell^3_j$. For every $j \in [m]$, let us consider the following votes $\mathfrak{q}_j(\ell^1_j), \mathfrak{q}_j(\ell^2_j), \mathfrak{q}_j(\ell^3_j)$.

$$\mathfrak{q}_j(\ell^k_j) : c_j \succ \ell^k_j \succ \text{others}, \forall k \in \{1,2,3\}$$

Using $\mathfrak{q}_j(\ell^1_j), \mathfrak{q}_j(\ell^2_j), \mathfrak{q}_j(\ell^3_j)$, we define the partial votes $\mathfrak{q}'_j(\ell^1_j), \mathfrak{q}'_j(\ell^2_j), \mathfrak{q}'_j(\ell^3_j)$ as follows.

$$\mathfrak{q}'_j(\ell^k_j) : \mathfrak{q}_j(\ell^k_j) \setminus \{(c_j, \ell^k_j)\}, \forall k \in \{1,2,3\}$$

Let us define

$$\mathcal{P} = \cup_{i\in[n]} \{\mathfrak{p}^1_{x_i}, \mathfrak{p}^2_{x_i}, \mathfrak{p}^1_{\bar{x}_i}, \mathfrak{p}^2_{\bar{x}_i}\} \cup_{j\in[m]} \{\mathfrak{q}_j(\ell^1_j), \mathfrak{q}_j(\ell^2_j), \mathfrak{q}_j(\ell^3_j)\}$$

Table 3 Summary of Copeland$^\alpha$ scores of the candidates from $\mathcal{P} \cup \mathcal{Q}$. All the wins and defeats in the table are by a margin of 2.

Candidates	Copeland$^\alpha$ score	Winning against	Losing against	Tie with
c	$(2n+m)\alpha$ $+n+{}^{3mn}/_{4}$	$G' \subset \mathcal{G}, \lvert G'\rvert = n+{}^{3mn}/_{4}$	$\mathcal{G} \setminus G', \lvert G'\rvert = n+{}^{3mn}/_{4}$ $d_i, \forall i \in [n]$	$x_i, \bar{x}_i \forall i \in [n]$ $c_j \forall j \in [m]$
$x_i, \forall i \in [n]$	$(2n+m)\alpha$ $+n+{}^{3mn}/_{4}$	$G'' \subset \mathcal{G}, \lvert G''\rvert = {}^{3mn}/_{4}$ $d_i \forall i \in [n]$	$\mathcal{G} \setminus (G' \cup G'')$	$c, G' \subset \mathcal{G}, \lvert G'\rvert = m$ $x_j, \forall j \in [n] \setminus \{i\}$ $\bar{x}_j \forall j \in [n]$
$\bar{x}_i, \forall i \in [n]$	$(2n+m)\alpha$ $+n+{}^{3mn}/_{4}$	$G'' \subset \mathcal{G}, \lvert G''\rvert = {}^{3mn}/_{4}$ $d_i \forall i \in [n]$	$\mathcal{G} \setminus (G' \cup G'')$	$c, G' \subset \mathcal{G}, \lvert G'\rvert = m$ $\bar{x}_j, \forall j \in [n] \setminus \{i\}$ $x_j \forall j \in [n]$
$c_j, \forall j \in [m]$	$(2n+m-1)\alpha$ $+n+{}^{3mn}/_{4}+1$	$x_i, \bar{x}_i \forall i \in [n]$ $G' \subset \mathcal{G}, \lvert G'\rvert = {}^{3mn}/_{4}-n+1$	$\mathcal{G} \setminus (G' \cup G'')$ $d_i, \forall i \in [n]$	c $c_j \forall j \in [m] \setminus \{i\}$ $G'' \subset \mathcal{G}, \lvert G''\rvert = 2n-1$
$d_i, i \in [n]$	$(2n+m)\alpha$ $+n+{}^{3mn}/_{4}-1$	$c, c_j, \forall j \in [m]$ $G'' \subset \mathcal{G}, \lvert G''\rvert = {}^{3mn}/_{4}-m+n-2$	$x_i, \bar{x}_i \forall i \in [n]$ $\mathcal{G} \setminus (G' \cup G'')$	$G' \subset \mathcal{G}, \lvert G'\rvert = 2n+m$
$g_i, \forall i \in [mn]$	$< {}^{3mn}/_{4}$		$\forall j \in \{i+k : k \in [\lfloor {}^{(mn-1)}/_{2} \rfloor]$	

and

$$\mathcal{P}' = \cup_{i\in[n]}\{\mathfrak{p}^{1\prime}_{x_i}, \mathfrak{p}^{2\prime}_{x_i}, \mathfrak{p}^{1\prime}_{\bar{x}_i}, \mathfrak{p}^{2\prime}_{\bar{x}_i}\} \cup_{j\in[m]} \{\mathfrak{q}'_j(\ell^1_j), \mathfrak{q}'_j(\ell^2_j), \mathfrak{q}'_j(\ell^3_j)\}.$$

There exists a set of complete votes $\mathcal{Q}$ of size polynomial in n and m which realizes Table 3 [24]. All the wins and defeats in Table 3 are by a margin of 2. We now define the instance $\mathcal{I}'$ of Possible Winner to be $(\mathcal{C}, \mathcal{P}' \cup \mathcal{Q}, c)$. Notice that the number of undetermined pairs of candidates in every vote in $\mathcal{I}'$ is at most 1. This finishes the description of the Possible Winner instance. We defer the formal proof of equivalence of the two instances and the polynomial time solvable case to the appendix. ◀

Next we present Lemma 14 which resolves the complexity of the Possible Winner problem for the Copeland$^\alpha$ voting rule for every $\alpha \in [1/2, 1)$ when every partial vote has at most one undetermined pair of candidates.

▶ **Lemma 14.** *[⋆] The* Possible Winner *problem is* NP*-complete for the Copeland$^\alpha$ voting rule even if the number of undetermined pairs in every vote is at most* 1 *for every* $\alpha \in [1/2, 1)$.

We get the following result for the Copeland$^\alpha$ voting rule from Theorem 13 and 14.

▶ **Theorem 15.** *The* Possible Winner *problem is* NP*-complete for the Copeland$^\alpha$ voting rule even if the number of undetermined pairs of candidates in every vote is at most* 1 *for every* $\alpha \in (0, 1)$.

3.3 Maximin and Bucklin Voting Rules

To prove our hardness result for the maximin voting rule, we reduce the Possible Winner problem from the d–Multicolored Independent Set problem which is defined as below. d–Multicolored Independent Set is known to be NP-complete (for example, see this [10]). We denote arbitrary instance of d–Multicolored Independent Set by $\left(\mathcal{V} = \uplus_{i=1}^{k} \mathcal{V}_k, \mathcal{E}\right)$.

▶ **Definition 16** (d–Multicolored Independent Set)**.** Given a d-regular graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an integer k, and a partition of the set of vertices $\mathcal{V}$ into k independent sets $\mathcal{V}_1, \ldots, \mathcal{V}_k$, that is $\mathcal{V} = \cup_{i\in[k]}\mathcal{V}_i$ and $\mathcal{V}_i$ is an independent set for every $i \in [k]$, does there exists an independent set $\mathcal{S} \subset \mathcal{V}$ in $\mathcal{G}$ such that $|\mathcal{S} \cap \mathcal{V}_i| = 1$ for every $i \in [k]$.

Table 4 Pairwise margins of candidates from $\mathcal{P} \cup \mathcal{Q}$.

$\forall e \in \mathcal{E}, \mathcal{D}_{\mathcal{P}\cup\mathcal{Q}}(e, c) = \lambda$	$\forall i \in [k], \forall u \in \mathcal{V}_i, \mathcal{D}_{\mathcal{P}\cup\mathcal{Q}}(u, g_i) = \lambda - 2d$
$\forall i \in [k], \forall u \in \mathcal{V}_i, \mathcal{D}_{\mathcal{P}\cup\mathcal{Q}}(g'_i, u) = \lambda + 2d$	$\forall i \in [k], e \in \mathcal{E}, \mathcal{D}_{\mathcal{P}\cup\mathcal{Q}}(e, g'_i) = \lambda$
$\forall e = (u_i, u_j) \in \mathcal{E}, \mathcal{D}_{\mathcal{P}\cup\mathcal{Q}}(u_i, e) = \mathcal{D}_{\mathcal{P}\cup\mathcal{Q}}(u_j, e) = \lambda - 2$	

Table 5 Summary of initial Copeland scores of the candidates.

Candidate	maximin score	worst against	Candidate	maximin score	worst against
c	$-\lambda$	$e \in \mathcal{E}$	$(u_i, u_j) \in \mathcal{E}$	$-(\lambda - 2)$	u_i, u_j
$u \in \mathcal{V}_i$	$-(\lambda + 2d)$	g'_i	g_i	$-(\lambda - 2d)$	$u \in \mathcal{V}_i$
g'_i	$-\lambda$	$e \in \mathcal{E}$			

Now we prove our hardness result for the Possible Winner problem for the maximin voting rule in Theorem 17.

▶ **Theorem 17.** *The* Possible Winner *problem is* NP*-complete for the maximin voting rule even if the number of undetermined pairs of candidates in every vote is at most* 2*.*

Proof. The Possible Winner problem for the maximin voting rule is clearly in NP. To prove NP-hardness of Possible Winner, we reduce Possible Winner from d–Multicolored Independent Set. Let $\mathcal{I} = \left(\mathcal{V} = \cup_{i=1}^{k} \mathcal{V}_k, \mathcal{E}\right)$ be an arbitrary instance of d–Multicolored Independent Set. We construct an instance $\mathcal{I}'$ of Possible Winner from $\mathcal{I}$ as follows.

Set of candidates: $\mathcal{C} = \mathcal{V} \cup \mathcal{E} \cup \{c\} \cup \{g_i, g'_i : i \in [k]\}$

For every $u \in \mathcal{V}_i$ and $\ell \in [d]$, let us consider the following vote $\mathfrak{p}_u$.

$\mathfrak{p}_u^\ell = \overrightarrow{(\mathcal{C} \setminus \{u, g_i, g'_i\})}_u \succ g_i \succ g'_i \succ u$, where $\overrightarrow{(\mathcal{C} \setminus \{u, g_i, g'_i\})}_u$ is any fixed ordering of $\mathcal{C} \setminus \{u, g_i, g'_i\}$

Using $\mathfrak{p}_u^\ell$, we define a partial vote $\mathfrak{p}_u'^\ell$ as $\mathfrak{p}_u'^\ell = \mathfrak{p}_u^\ell \setminus \{(g_i, u), (g'_i, u)\}$. For every edge $e = (u_i, u_j)$ where $u_i \in \mathcal{V}_i$ and $u_j \in \mathcal{V}_j$, let us consider the following votes $\mathfrak{p}_{e,u_i}$ and $\mathfrak{p}_{e,u_j}$.

$$\mathfrak{p}_{e,u_i} = \overrightarrow{(\mathcal{C} \setminus \{u_i, g'_i, e\})} \succ e \succ g'_i \succ u_i \ , \ \mathfrak{p}_{e,u_j} = \overrightarrow{(\mathcal{C} \setminus \{u_j, g'_j, e\})} \succ e \succ g'_j \succ u_j$$

Using $\mathfrak{p}_{e,u_i}$ and $\mathfrak{p}_{e,u_j}$, we define the partial votes $\mathfrak{p}'_{e,u_i}$ and $\mathfrak{p}'_{e,u_j}$ as follows.

$$\mathfrak{p}'_{e,u_i} = \mathfrak{p}_{e,u_i} \setminus \{(e, u_i), (g'_i, u_i)\} \ , \ \mathfrak{p}'_{e,u_j} = \mathfrak{p}_{e,u_j} \setminus \{(e, u_j), (g'_j, u_j)\}$$

Let us call $\mathfrak{p}_e = \{\mathfrak{p}_{e,u_i}, \mathfrak{p}_{e,u_j}\}$ and $\mathfrak{p}'_e = \{\mathfrak{p}'_{e,u_i}, \mathfrak{p}'_{e,u_j}\}$. Let us define $\mathcal{P} = \cup_{u\in\mathcal{V},\ell\in[d]} \mathfrak{p}_u^\ell \cup_{e\in\mathcal{E}} \mathfrak{p}_e$ and $\mathcal{P}' = \cup_{u\in\mathcal{V},\ell\in[d]} \mathfrak{p}_u'^\ell \cup_{e\in\mathcal{E}} \mathfrak{p}'_e$. There exists a set of complete votes $\mathcal{Q}$ of size polynomial in $|\mathcal{V}|$ and $|\mathcal{E}|$ with the pairwise margins as in Table 4 [24]. Let $\lambda > 3d$ be any positive even integer.

For every pair of candidates $(c_i, c_j) \in \mathcal{C} \times \mathcal{C}$ whose pairwise margin is not defined above, we define $\mathcal{D}_{\mathcal{P}\cup\mathcal{Q}}(c_i, c_j) = 0$. We summarize the maximin score of every candidate in $\mathcal{P} \cup \mathcal{Q}$ in Table 5. We now define the instance $\mathcal{I}'$ of Possible Winner to be $(\mathcal{C}, \mathcal{P}' \cup \mathcal{Q}, c)$. Notice that the number of undetermined pairs of candidates in every vote in $\mathcal{I}'$ is at most 2. This finishes the description of the Possible Winner instance. We claim that $\mathcal{I}$ and $\mathcal{I}'$ are equivalent.

In the forward direction, suppose that $\mathcal{I}$ be a YES instance of d–MULTICOLORED INDEPENDENT SET. Then there exists $u_i \in \mathcal{V}_i$ for every $i \in [k]$ such that $\mathcal{U} = \{u_i : i \in [k]\}$ forms an independent set. We extend the partial vote $\mathfrak{p}'^{\ell}_u$ for every $u \in \mathcal{V}_i, i \in [k], \ell \in [d]$ to $\bar{\mathfrak{p}}^{\ell}_u$ as follows.

$$\bar{\mathfrak{p}}^{\ell}_u = \begin{cases} \overrightarrow{(\mathcal{C} \setminus \{u, g_i, g'_i\})}_u \succ u \succ g_i \succ g'_i & u \in \mathcal{U} \\ \overrightarrow{(\mathcal{C} \setminus \{u, g_i, g'_i\})}_u \succ g_i \succ g'_i \succ u & u \notin \mathcal{U} \end{cases}$$

For every $e = (u_i, u_j)$, we extend $\mathfrak{p}'_{e,u_i}$ and $\mathfrak{p}'_{e,u_j}$ to $\bar{\mathfrak{p}}_{e,u_i}$ and $\bar{\mathfrak{p}}_{e,u_j}$. Since $\mathcal{U}$ is an independent set, at least one of u_i and u_j does not belong to $\mathcal{U}$. Without loss of generality, let us assume $u_i \notin \mathcal{U}$.

$$\bar{\mathfrak{p}}_{e,u_i} = \overrightarrow{(\mathcal{C} \setminus \{u_i, g'_i, e\})} \succ u_i \succ e \succ g'_i \ , \ \bar{\mathfrak{p}}_{e,u_j} = \overrightarrow{(\mathcal{C} \setminus \{u_j, g'_j, e\})} \succ e \succ g'_j \succ u_j$$

Let us call $\bar{\mathfrak{p}}_e = \{\bar{\mathfrak{p}}_{e,u_i}, \bar{\mathfrak{p}}_{e,u_j}\}$. We consider the extension of $\mathcal{P}$ to $\bar{\mathcal{P}} = \cup_{u \in \mathcal{V}, \ell \in [d]} \bar{\mathfrak{p}}^{\ell}_u \cup_{e \in \mathcal{E}} \bar{\mathfrak{p}}_e$. We claim that c is a co-winner in the profile $\bar{\mathcal{P}} \cup \mathcal{Q}$ since the maximin score of c, g_i, g'_i for every $i \in [k]$, $u \in \mathcal{V}$, and $e \in \mathcal{E}$ in $\bar{\mathcal{P}} \cup \mathcal{Q}$ is $-\lambda$.

In the reverse direction suppose the POSSIBLE WINNER instance $\mathcal{I}'$ be a YES instance. Then there exists an extension of the set of partial votes $\mathcal{P}'$ to a set of complete votes $\bar{\mathcal{P}}$ such that c is a co-winner in $\bar{\mathcal{P}} \cup \mathcal{Q}$. Let us call the extension of $\mathfrak{p}'^{\ell}_u$ in $\bar{\mathcal{P}}$ $\bar{\mathfrak{p}}^{\ell}_u$, $\mathfrak{p}'_{e,u_i}$ and $\mathfrak{p}'_{e,u_j}$ in $\bar{\mathcal{P}}$ $\bar{\mathfrak{p}}_{e,u_i}$ and $\bar{\mathfrak{p}}_{e,u_j}$ respectively. First we notice that the maximin score of c in $\bar{\mathcal{P}} \cup \mathcal{Q}$ is $-\lambda$ since the relative ordering of c with respect to every other candidate is already fixed in $\mathcal{P}' \cup \mathcal{Q}$. Now we observe that, in $\mathcal{P} \cup \mathcal{Q}$, the maximin score of g_i for every $i \in [k]$ is $-(\lambda - 2d)$. Hence, for c to co-win, there must exists at least one $u^*_i \in \mathcal{V}_i$ for every $i \in [k]$ such that $u^*_i \succ g_i \succ g'_i$ in $\bar{\mathfrak{p}}^{\ell}_{u^*_i}$ for every $\ell \in [d]$. We claim that $\mathcal{U} = \{u^*_i : i \in [k]\}$ is an independent set in $\mathcal{I}$. If not, then suppose there exists an edge e between u^*_i and u^*_j for some $i, j \in [k]$. Now notice that, for c to co-win either $u^*_i \succ e \succ g'_i$ in $\bar{\mathfrak{p}}_{e,u^*_i}$ or $u^*_j \succ e \succ g'_j$ in $\bar{\mathfrak{p}}_{e,u^*_j}$. However, this makes the maximin score of either u^*_i or u^*_j strictly more than $-\lambda$ contradicting our assumption that c co-wins the election. Hence, $\mathcal{U}$ forms an independent set in $\mathcal{I}$. ◀

We next prove in Theorem 18 that the maximum number of undetermined pairs of candidates in Theorem 17 is tight.

▶ **Theorem 18.** *[⋆] The* POSSIBLE WINNER *problem is in* P *for the maximin voting rule if the number of undetermined pairs of candidates in every vote is at most* 1.

Finally, we state our results for the Bucklin voting rule.

▶ **Theorem 19.** *[⋆] The* POSSIBLE WINNER *problem is* NP*-complete for the Bucklin voting rule even if the number of undetermined pairs of candidates in every vote is at most* 2*, and is in* P *if the number of undetermined pairs of candidates in every vote is at most* 1.

4 Conclusion

We have demonstrated the exact minimum number of undetermined pairs allowed per vote which keeps the POSSIBLE WINNER winner problem NP-complete, and we were able to address a large class of scoring rules, Copeland$^{\alpha}$, maximin, and Bucklin voting rules. Our results generalize many of the known hardness results in the literature, and show that for many voting rules, we need a surprisingly small number of undetermined pairs (often just one or two) for the POSSIBLE WINNER problem to be NP-complete. In the context of scoring rules, it would be interesting to extend these tight results to the class of pure scoring rules, and to extend Theorem 9 to account for all smooth scoring rules.

References

1 Dorothea Baumeister, Magnus Roos, and Jörg Rothe. Computational complexity of two variants of the possible winner problem. In *Proc. International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 853–860, 2011.

2 Dorothea Baumeister and Jörg Rothe. Taking the final step to a full dichotomy of the possible winner problem in pure scoring rules. *Inf. Process. Lett.*, 112(5):186–190, 2012. doi:10.1016/j.ipl.2011.11.016.

3 Piotr Berman, Marek Karpinski, and Alex D. Scott. Approximation hardness and satisfiability of bounded occurrence instances of SAT. *Electronic Colloquium on Computational Complexity (ECCC)*, 10(022), 2003. URL: http://eccc.hpi-web.de/eccc-reports/2003/TR03-022/index.html.

4 Nadja Betzler, Robert Bredereck, and Rolf Niedermeier. Partial kernelization for rank aggregation: theory and experiments. In *Proc. 5th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 26–37. Springer, 2010.

5 Nadja Betzler, Robert Bredereck, and Rolf Niedermeier. Theoretical and empirical evaluation of data reduction for exact kemeny rank aggregation. *Autonomous Agents and Multi-Agent Systems*, 28(5):721–748, 2014. doi:10.1007/s10458-013-9236-y.

6 Nadja Betzler and Britta Dorn. Towards a dichotomy of finding possible winners in elections based on scoring rules. In *Proc. 34th Mathematical Foundations of Computer Science (MFCS)*, pages 124–136. Springer, 2009.

7 Nadja Betzler, Susanne Hemmann, and Rolf Niedermeier. A Multivariate Complexity Analysis of Determining Possible Winners given Incomplete Votes. In *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI)*, volume 9, pages 53–58, 2009.

8 Yann Chevaleyre, Jérôme Lang, Nicolas Maudet, and Jérôme Monnot. Possible winners when new candidates are added: The case of scoring rules. In *Proc. 24th International Conference on Artificial Intelligence (AAAI)*, 2010.

9 William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *J. Artif. Int. Res.*, 10(1):243–270, May 1999. URL: http://dl.acm.org/citation.cfm?id=1622859.1622867.

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

11 Palash Dey. Resolving the complexity of some fundamental problems in computational social choice. *CoRR*, abs/1703.08041, 2017. URL: http://arxiv.org/abs/1703.08041.

12 Palash Dey and Neeldhara Misra. On the exact amount of missing information that makes finding possible winners hard. *CoRR*, abs/1610.08407, 2016. URL: http://arxiv.org/abs/1610.08407.

13 Palash Dey, Neeldhara Misra, and Y. Narahari. Kernelization complexity of possible winner and coalitional manipulation problems in voting. In *Proc. 14th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pages 87–96, 2015. URL: http://dl.acm.org/citation.cfm?id=2772894.

14 Palash Dey, Neeldhara Misra, and Y. Narahari. Complexity of manipulation with partial information in voting. In *Proc. 25th International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, USA*, pages 229–235, 2016. URL: http://www.ijcai.org/Abstract/16/040.

15 Palash Dey, Neeldhara Misra, and Y. Narahari. Frugal bribery in voting. In *Proc. 30th AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 2466–2472, 2016. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12133.

16 Palash Dey, Neeldhara Misra, and Y. Narahari. Kernelization complexity of possible winner and coalitional manipulation problems in voting. *Theor. Comput. Sci.*, 616:111–125, 2016. doi:10.1016/j.tcs.2015.12.023.

17 Palash Dey, Neeldhara Misra, and Y. Narahari. Frugal bribery in voting. *Theor. Comput. Sci.*, 676:15–32, 2017. doi:10.1016/j.tcs.2017.02.031.

18 Piotr Faliszewski, Yannick Reisch, Jörg Rothe, and Lena Schend. Complexity of manipulation, bribery, and campaign management in bucklin and fallback voting. In *Proc. 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1357–1358. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

19 Michael R Garey and David S Johnson. *Computers and Intractability*, volume 174. freeman New York, 1979.

20 Benjamin G. Jackson, Patrick S. Schnable, and Srinivas Aluru. Consensus genetic maps as median orders from inconsistent sources. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 5(2):161–171, 2008. doi:10.1145/1371585.1371586.

21 Kathrin Konczak and Jérôme Lang. Voting procedures with incomplete preferences. In *Proc. 19th International Joint Conference on Artificial Intelligence-05 Multidisciplinary Workshop on Advances in Preference Handling*, volume 20, 2005.

22 Jérôme Lang, Maria Silvia Pini, Francesca Rossi, Domenico Salvagnin, Kristen Brent Venable, and Toby Walsh. Winner determination in voting trees with incomplete preferences and weighted votes. *Auton. Agent Multi Agent Syst.*, 25(1):130–157, 2012.

23 Jérôme Lang, Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Winner determination in sequential majority voting. In *Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 1372–1377, 2007.

24 David C McGarvey. A theorem on the construction of voting paradoxes. *Econometrica*, pages 608–610, 1953.

25 Hervé Moulin, Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D Procaccia. *Handbook of Computational Social Choice*. Cambridge University Press, 2016.

26 David M. Pennock, Eric Horvitz, and C. Lee Giles. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proc. 17th National Conference on Artificial Intelligence and 12th Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 729–734, 2000. URL: http://www.aaai.org/Library/AAAI/2000/aaai00-112.php.

27 Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Incompleteness and incomparability in preference aggregation. In *Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 1464–1469, 2007.

28 Toby Walsh. Uncertainty in preference elicitation and aggregation. In *Proc. 22nd International Conference on Artificial Intelligence (AAAI)*, volume 22, page 3, 2007.

29 Lirong Xia and Vincent Conitzer. Determining possible and necessary winners under common voting rules given partial orders. *J. Artif. Intell. Res.*, 41(2):25–67, 2011.

Fractal Intersections and Products via Algorithmic Dimension

Neil Lutz*

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA
neillutz@gmail.com

Abstract

Algorithmic dimensions quantify the algorithmic information density of individual points and may be defined in terms of Kolmogorov complexity. This work uses these dimensions to bound the classical Hausdorff and packing dimensions of intersections and Cartesian products of fractals in Euclidean spaces. This approach shows that a known intersection formula for Borel sets holds for arbitrary sets, and it significantly simplifies the proof of a known product formula. Both of these formulas are prominent, fundamental results in fractal geometry that are taught in typical undergraduate courses on the subject.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases algorithmic randomness, geometric measure theory, Hausdorff dimension, Kolmogorov complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.58

1 Introduction

Classical fractal dimensions, among which *Hausdorff dimension* [12] is the most important, refine notions of measure to quantitatively classify sets of measure 0. In 2000, J. Lutz [15] showed that Hausdorff dimension can be simply characterized using betting strategies called *gales*, and that this characterization can be *effectivized* in order to quantitatively classify non-random infinite data objects. This *effective Hausdorff dimension* and other, related *algorithmic dimensions* have been applied to multiple areas of computer science and have proven especially useful in algorithmic information theory [25].

The connection between algorithmic and classical dimensions has more recently been exploited in the other direction, i.e., to apply algorithmic information theoretic methods and intuition to classical fractal geometry (e.g., [29, 2]). A *point-to-set principle* of J. Lutz and N. Lutz [16], stated here as Theorem 6, characterizes the classical Hausdorff dimension of any set in $\mathbb{R}^n$ in terms of the algorithmic dimensions of its individual points.

In the same work, J. Lutz and N. Lutz showed that the point-to-set principle gives rise to a new, pointwise technique for dimensional lower bounds, and, as a proof of concept, used this technique to give an algorithmic information theoretic proof of Davies's 1971 [7] theorem stating that every Kakeya set in $\mathbb{R}^2$ has Hausdorff dimension 2. This bounding technique has since been used by N. Lutz and Stull [18] to make new progress on a problem in classical fractal geometry by deriving an improved lower bound on the Hausdorff dimension of generalized Furstenberg sets, as defined by Molter and Rela [26].

* This research was done at Rutgers University and supported in part by National Science Foundation Grant 1445755.

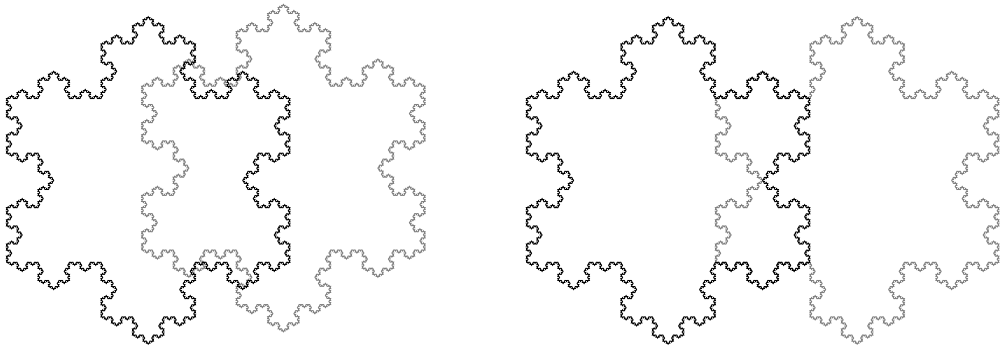


Figure 1 Let E and F both be Koch snowflakes, which have Hausdorff dimension $\log_3 4 \approx 1.26$. *Left:* Theorem 1 states that, for almost all translation parameters $z \in \mathbb{R}^2$, the Hausdorff dimension of the intersection $E \cap (F + z)$ is at most $2 \log_3 4 - 2 \approx 0.52$. *Right:* For a measure zero set of translations, the intersection may have Hausdorff dimension as large as that of the original sets. Note that Koch curves are Borel sets, so the new generality introduced by Theorem 1 is not required for this example.

The same algorithmic dimensional technique is applied here to bound the dimensions of intersections and products of fractals. Most significantly, we extend the following intersection formula, previously shown to hold when E and F are Borel sets [11], to arbitrary sets E and F.[1] This formula is illustrated in Figure 1.

▶ **Theorem 1.** *For all $E, F \subseteq \mathbb{R}^n$, and for (Lebesgue) almost all $z \in \mathbb{R}^n$,*

$$\dim_H(E \cap (F + z)) \leq \max\{0, \dim_H(E \times F) - n\},$$

where $F + z = \{x + z : x \in F\}$.

Our approach also yields a simplified proof of the following known product formula for general sets.

▶ **Theorem 2** (Marstrand [19])**.** *For all $E \subseteq \mathbb{R}^m$ and $F \subseteq \mathbb{R}^n$,*

$$\dim_H(E) + \dim_H(F) \leq \dim_H(E \times F).$$

We use symmetric arguments to derive the known corresponding statements about *packing dimension* [37, 10], a formulation of fractal dimension that was developed independently by Tricot [37] and Sullivan [36] and is dual to Hausdorff dimension. These results are included here to showcase the versatility of this technique and its ability to capture the exact duality between Hausdorff and packing dimensions.

2 Classical Fractal Dimensions

We begin by stating classical, measure-theoretic definitions of the two most well-studied notions of fractal dimension, Hausdorff dimension and packing dimension. These definitions are included here for completeness but are not used directly in the remainder of this work; we will instead apply equivalent characterizations in terms of algorithmic information, as described in Section 3.

[1] This result is closely related to the Marstrand Slicing Theorem, as stated in the excellent recent book by Bishop and Peres [4]. The proof given there assumes that a set is Borel, but this assumption was inadvertently omitted from the theorem statement [3].

▶ **Definition 2.1** (Hausdorff [12]). For $E \subseteq \mathbb{R}^n$, let $\mathcal{U}_\delta(E)$ be the collection of all countable covers of E by sets of positive diameter at most δ, where the *diameter* of any set $U \subseteq \mathbb{R}^n$ is given by

$$\operatorname{diam}(U) = \sup_{x,y \in U} |x - y| .$$

For all $s \geq 0$, let

$$H^s_\delta(E) = \inf \left\{ \sum_{i \in \mathbb{N}} \operatorname{diam}(U_i)^s \ : \ \{U_i\}_{i \in \mathbb{N}} \in \mathcal{U}_\delta(E) \right\} .$$

The *s-dimensional Hausdorff (outer) measure of* E is

$$H^s(E) = \lim_{\delta \to 0^+} H^s_\delta(E) ,$$

and the *Hausdorff dimension of* E is

$$\dim_H(E) = \inf \{s > 0 : H^s(E) = 0\} = \sup \{s : H^s(E) = \infty\} .$$

Three desirable properties have made $\dim_H$ the most standard notion of fractal dimension since it was introduction by Hausdorff in 1919. First, it is defined on every set in $\mathbb{R}^n$. Second, it is *monotone*: if $E \subseteq F$, then $\dim_H(E) \leq \dim_H(F)$. Third, it is *countably stable*: if $E = \bigcup_{i \in \mathbb{N}} E_i$, then $\dim_H(E) = \sup_{i \in \mathbb{N}} \dim_H(E_i)$. These three properties also hold for packing dimension, which was defined much later, independently by Tricot [37] and by Sullivan [36].

▶ **Definition 2.2** (Tricot [37], Sullivan [36]). For all $x \in \mathbb{R}^n$ and $\rho > 0$, let $B_\rho(x)$ denote the open ball of radius ρ and center x. For all $E \subseteq \mathbb{R}^n$, let $\mathcal{V}_\delta(E)$ be the class of all countable collections of pairwise disjoint open balls with centers in E and diameters at most δ. That is, for every $i \in \mathbb{N}$, we have $V_i = B_{\rho_i}(x_i)$ for some $x_i \in E$ and $\rho_i \in [0, \delta/2]$, and for every $j \neq i$, $V_i \cap V_j = \emptyset$. For all $s \geq 0$, define

$$P^s_\delta(E) = \sup \left\{ \sum_{i \in \mathbb{N}} \operatorname{diam}(V_i)^s \ : \ \{V_i\}_{i \in \mathbb{N}} \in \mathcal{V}_\delta(E) \right\} ,$$

and let

$$P^s_0(E) = \lim_{\delta \to 0^+} P^s_\delta(E) .$$

The *s-dimensional packing (outer) measure of* E is

$$P^s(E) = \inf \left\{ \sum_{i \in \mathbb{N}} P^s_0(E_i) \ : \ E \subseteq \bigcup_{i \in \mathbb{N}} E_i \right\} ,$$

and the *packing dimension of* E is

$$\dim_P(E) = \inf \{s : P^s(E) = 0\} = \sup \{s > 0 : P^s(E) = \infty\} .$$

Notice that defining packing dimension in this way requires an extra step of optimization compared to Hausdorff dimension. More properties and details about classical fractal dimensions may be found in standard references such as [23, 11, 35].

3 Algorithmic Fractal Dimensions

This section defines the effective Hausdorff and packing dimensions in terms of algorithmic information, i.e., Kolmogorov complexity. We also define conditional dimensions and discuss some properties of these dimensions, including their relationships to classical Hausdorff and packing dimensions.

3.1 Kolmogorov Complexity

Kolmogorov complexity quantifies the *incompressibility* of finite data objects. It is most often defined in the space $\{0,1\}^*$ of binary strings, but it is readily extended to other discrete domains. For the purposes of this work, the complexity of rational points is most relevant. Hence, fix some standard binary encoding for n-tuples of rationals. The *Kolmogorov complexity of* p is the length of the shortest binary program that outputs p. Formally, it is

$$K(p) = \min_{\pi\in\{0,1\}^*} \{|\pi| : U(\pi) = p\},$$

where U is a fixed universal prefix-free Turing machine and $|\pi|$ is the length of π. This quantity is also called the *algorithmic information content* of p. The *conditional Kolmogorov complexity of* p *given* $q \in \mathbb{Q}^n$ is the length of the shortest binary program that outputs p when given q as an input:

$$K(p|q) = \min_{\pi\in\{0,1\}^*} \{|\pi| : U(\pi, q) = p\}.$$

The *algorithmic mutual information between* $p \in \mathbb{Q}^m$ *and* $q \in \mathbb{Q}^n$ measures, informally, the amount that knowledge of q helps in the task of compressing p. Formally, it is

$$I(p : q) = K(p) - K(p|q).$$

The quantities $K(p)$, $K(p|q)$, and $I(p : q)$ may be considered algorithmic versions of *entropy* $H(X)$, *conditional entropy* $H(X|Y)$, and *mutual information* $I(X;Y)$, from classical (Shannon) information theory. See references [14, 27, 8] for more details on algorithmic information and the connections between algorithmic and classical theories of information.

3.2 Effective Dimensions

Using approximation by rationals, Kolmogorov complexity may be further extended to Euclidean spaces [17]. For every $E \subseteq \mathbb{R}^n$, define

$$K(E) = \min\{K(p) : p \in E \cap \mathbb{Q}^n\},$$

where the minimum is understood to be infinite if $E \cap \mathbb{Q}^n$ is empty. This is the length of the shortest program that outputs some rational point in E. The *Kolmogorov complexity of* $x \in \mathbb{R}^n$ *at precision* $r \in \mathbb{N}$ is given by

$$K_r(x) = K(B_{2^{-r}}(x)),$$

the length of the shortest program that outputs any *precision-r* rational approximation of x. $K_r(x)$ may also be described as the *algorithmic information content* of x at precision r, and similarly, $K_r(x)/r$ is the *algorithmic information density* of x at precision r. This ratio does not necessarily converge as $r \to \infty$, but it does have limiting bounds in $[0, n]$. These limits are used to define effective dimensions.

▶ **Definition 3.1** ([15, 24, 1, 17]). Let $x \in \mathbb{R}^n$.

1. The *effective Hausdorff dimension of* x is
$$\dim(x) = \liminf_{r\to\infty} \frac{K_r(x)}{r}\,.$$
2. The *effective packing dimension of* x is
$$\mathrm{Dim}(x) = \limsup_{r\to\infty} \frac{K_r(x)}{r}\,.$$

These dimensions were originally defined by J. Lutz [15] and Athreya, Hitchcock, J. Lutz, and Mayordomo [1], respectively. The original definitions were in Cantor space and used *gales*, which are betting strategies that generalize martingales, emphasizing the *unpredictability* of a sequence instead of its incompressibility. The Kolmogorov complexity characterizations and translation to Euclidean spaces are due to Mayordomo [24] and J. Lutz and Mayordomo [17]. Relationships between Hausdorff dimension and Kolmogorov complexity were also studied earlier by Ryabko [30, 31, 32], Staiger [33, 34], and Cai and Hartmanis [5]; see Section 6 of [15] for a detailed discussion of this history.

We will use the fact that these dimensions are preserved by sufficiently well-behaved functions, namely bi-Lipschitz computable bijections.

▶ **Lemma 3** (Reimann [28], Case and J. Lutz [6]). *If* $f : \mathbb{R}^m \to \mathbb{R}^n$ *is computable and bi-Lipschitz, then* $\dim(x) = \dim(f(x))$ *and* $\mathrm{Dim}(x) = \mathrm{Dim}(f(x))$ *for all* $x \in \mathbb{R}^m$.

3.3 Conditional Dimensions

The information theoretic nature of Definition 3.1 has led to the development of algorithmic dimensional quantities corresponding to the other algorithmic information theoretic quantities defined above. As analogues to mutual information and conditional information, Case and J. Lutz defined *mutual dimensions* [6], and J. Lutz and N. Lutz defined *conditional dimensions*. This work will use the latter, which we now describe.

Given $E \subseteq \mathbb{R}^m$ and $F \subseteq \mathbb{R}^n$, define
$$K(E|F) = \max\left\{\min\{K(p|q) : p \in E \cap \mathbb{Q}^m\} : q \in F \cap \mathbb{Q}^n\right\}.$$
Then the *conditional Kolmogorov complexity of* $x \in \mathbb{R}^m$ *at precision* $r \in \mathbb{N}$ *given* $y \in \mathbb{R}^n$ *at precision* $s \in \mathbb{N}$ is given by
$$K_{r,s}(x|y) = K(B_{2^{-r}}(x)|B_{2^{-s}}(y))\,.$$

▶ **Definition 3.2** (J. Lutz and N. Lutz [16]). Let $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$.

1. The *lower conditional dimension of* x *given* y is
$$\dim(x : y) = \liminf_{r\to\infty} \frac{K_{r,r}(x|y)}{r}\,.$$
2. The *upper conditional dimension of* x *given* y is
$$\mathrm{Dim}(x : y) = \limsup_{r\to\infty} \frac{K_{r,r}(x|y)}{r}\,.$$

That work also showed that the *symmetry of algorithmic information* holds in Euclidean space, in the form
$$K_r(x, y) = K_r(x) + K_{r,r}(y|x) + o(r)\,.$$
This fact and elementary properties of limits inferior and superior immediately imply the following *chain rule for effective dimensions*.

▶ **Theorem 4** (J. Lutz and N. Lutz [16]). *For all $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$,*

$$\begin{aligned}\dim(x) + \dim(y|x) &\leq \dim(x,y)\\ &\leq \dim(x) + \mathrm{Dim}(y|x)\\ &\leq \mathrm{Dim}(x,y)\\ &\leq \mathrm{Dim}(x) + \mathrm{Dim}(y|x)\,.\end{aligned}$$

3.4 Oracles and Relative Dimensions

By making the fixed universal machine U an oracle machine, the algorithmic information quantities above may be defined relative to any oracle $A \subseteq \mathbb{N}$. The definitions of $K^A(\sigma|\tau)$, $K^A(\sigma)$, $K^A_r(x)$, $K^A_r(x|y)$, $\dim^A(x)$, $\mathrm{Dim}^A(x)$, $\dim^A(x|y)$ and $\mathrm{Dim}^A(x|y)$ all exactly mirror their unrelativized versions, except that U is permitted to query membership in A as a computational step.

For $y \in \mathbb{R}^n$, we write $\dim^y(x)$ as shorthand for $\dim^{A_y}(x)$, where $A_y \subseteq \mathbb{N}$ encodes the binary expansions of y's coordinates in some standard way, and similarly for $\mathrm{Dim}^y(x)$. Since this kind of oracle access to y is at least as informative as any finite-precision estimate for y (ignoring the small amount of information given by the precision parameter itself), these relative dimensions are bounded above by conditional dimensions.

▶ **Lemma 5** (J. Lutz and N. Lutz [16]). *For all $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$,*

1. $\dim^y(x) \leq \dim(x|y)$,
2. $\mathrm{Dim}^y(x) \leq \mathrm{Dim}(x|y)$.

3.5 Point-to-Set Principle

Effective Hausdorff dimension and effective packing dimension were conceived as constructive versions of classical Hausdorff dimension and packing dimension [15, 1]. The following *point-to-set principle* uses relativization to precisely characterize their relationships to their non-algorithmic precursors.

▶ **Theorem 6** (J. Lutz and N. Lutz [16]). *For every $E \subseteq \mathbb{R}^n$, the Hausdorff dimension and packing dimension of E are*

1. $\dim_H(E) = \min_{A \subseteq \mathbb{N}} \sup_{x \in E} \dim^A(x)\,,$
2. $\dim_P(E) = \min_{A \subseteq \mathbb{N}} \sup_{x \in E} \mathrm{Dim}^A(x)\,.$

Notice that, unlike the definitions of $\dim_H(E)$ and $\dim_P(E)$ given in Section 2, the above characterizations are completely symmetrical.

Theorem 6 allows us to prove lower bounds on classical dimensions in a pointwise way. To show a statement of the form $\dim_H(E) \geq \alpha$, it suffices to show, for a given oracle A and every $\varepsilon > 0$, that there exists an $x \in E$ satisfying $\dim^A(x) \geq \alpha - \varepsilon$. Unlike previous applications of this bounding technique [16, 18], the proofs in Sections 4 and 5 do not directly invoke Kolmogorov complexity; the only tools needed are Lemma 3, Theorem 4, Lemma 5, and Theorem 6.

4 Intersections of Fractals

In this section we prove Theorem 1. We then use a symmetric argument to prove the corresponding statement for packing dimension, which is known [10]. For the case where

$E, F \subseteq \mathbb{R}^n$ are Borel sets, Theorem 1 was shown in its present form by Falconer [11]. Closely related results, which also place restrictions on E and F, were proven earlier by Mattila [21, 22] and Kahane [13].

▶ **Theorem 1.** *For all $E, F \subseteq \mathbb{R}^n$, and for (Lebesgue) almost all $z \in \mathbb{R}^n$,*

$$\dim_H(E \cap (F + z)) \leq \max\{0, \dim_H(E \times F) - n\}\,, \tag{1}$$

where $F + z = \{x + z : x \in F\}$.

Proof. Let $E, F \subseteq \mathbb{R}^n$ and $z \in \mathbb{R}^n$. If $E \cap (F + z) = \emptyset$, then (1) holds trivially, so assume that the intersection is nonempty. Theorem 6 guarantees that there is some oracle set $A \subseteq \mathbb{N}$ satisfying

$$\dim_H(E \times F) = \sup_{(x,y) \in E \times F} \dim^A(x, y)\,. \tag{2}$$

It also guarantees, given any $\varepsilon > 0$, that there is an $x \in E \cap (F + z)$ such that

$$\dim^{A,z}(x) \geq \dim_H(E \cap (F + z)) - \varepsilon\,. \tag{3}$$

Since $(x, x - z) \in E \times F$, we have

$$\begin{aligned}
\dim_H(E \times F) &\geq \dim^A(x, x - z) \\
&= \dim^A(x, z) \\
&\geq \dim^A(z) + \dim^A(x|z) \\
&\geq \dim^A(z) + \dim^{A,z}(x) \\
&\geq \dim^A(z) + \dim_H(E \cap (F + z)) - \varepsilon\,.
\end{aligned}$$

The above lines follow from (2), Lemma 3, Theorem 4, Lemma 5, and (3), respectively. Letting $\varepsilon \to 0$, we have

$$\dim_H(E \cap (F + z)) \leq \dim_H(E \times F) - \dim^A(z)\,.$$

Thus, (a) holds whenever $\dim^A(z) = n$. In particular, it holds when z is Martin-Löf random relative to A, i.e., for Lebesgue almost all $z \in \mathbb{R}^n$ [14, 20]. ◀

For the case that E and F are Borel sets, Falconer [11] notes that the intersection formula is readily extended to rigid motions and similarities. The same argument applies in the general case, so Theorem 1 has the following corollary.

▶ **Corollary 7.** *Let $E, F \subseteq \mathbb{R}^n$. Let G be the group of rigid motions or the group of similarities on $\mathbb{R}^n$. Then, for almost all $\sigma \in G$,*

$$\dim_H(E \cap \sigma(F)) \leq \max\{0, \dim_H(E \times F) - n\}\,. \tag{4}$$

Proof (Following Falconer [11]). For all rotations (and all scalings) of F, Theorem 1 tells us that (4) holds for almost all translations. Thus, (4) holds for almost all rigid motions and almost all similarities. ◀

A corresponding intersection formula for packing dimension has been shown for arbitrary $E, F \subseteq \mathbb{R}^n$ by Falconer [10]. That proof is not difficult or long, but an algorithmic dimensional proof is presented here as an instance where this technique applies symmetrically to both Hausdorff and packing dimension.

▶ **Theorem 8** (Falconer [10]). *For all $E, F \subseteq \mathbb{R}^n$, and for (Lebesgue) almost all $z \in \mathbb{R}^n$,*

$$\dim_P(E \cap (F+z)) \leq \max\{0, \dim_P(E \times F) - n\}\,.$$

Proof. As in Theorem 1, we may assume that the intersection is nonempty. Apply Theorem 6 to choose an oracle set $B \subseteq \mathbb{N}$ such that

$$\dim_P(E \times F) = \sup_{(x,y)\in E\times F} \mathrm{Dim}^B(x,y) \tag{5}$$

and, given $\varepsilon > 0$, a point $y \in E \cap (F+z)$ satisfying

$$\mathrm{Dim}^{B,z}(y) \geq \dim_P(E \cap (F+z)) - \varepsilon\,. \tag{6}$$

Then $(y, y-z) \in E \times F$, and we may proceed much as before:

$$\begin{aligned}
\dim_P(E \times F) &\geq \mathrm{Dim}^B(y, y-z) \\
&= \mathrm{Dim}^B(y,z) \\
&\geq \dim^B(z) + \mathrm{Dim}^B(y|z) \\
&\geq \dim^B(z) + \mathrm{Dim}^{B,z}(y) \\
&\geq \dim^B(z) + \dim_P(E \cap (F+z)) - \varepsilon\,.
\end{aligned}$$

These lines follow from (5), Lemma 3, Theorem 4, Lemma 5, and (6). Again, $\dim^B(z) = n$ for almost every $z \in \mathbb{R}^n$, so this completes the proof. ◀

5 Products of Fractals

In this section we prove four known product inequalities for fractal dimensions. Inequality (7), which was stated in the introduction as Theorem 2, is due to Marstrand [19]. When E and F are Borel sets, it is simple to prove (7) by using Frostman's Lemma, but the argument for general sets using net measures is considerably more difficult [23, 9]. The other three inequalities are due to Tricot [37]. Reference [23] gives a more detailed account of this history.

▶ **Theorem 9** (Marstrand [19], Tricot [37]). *For all $E \subseteq \mathbb{R}^m$ and $F \subseteq \mathbb{R}^n$,*

$$\begin{aligned}
\dim_H(E) + \dim_H(F) &\leq \dim_H(E \times F) && (7)\\
&\leq \dim_H(E) + \dim_P(F) && (8)\\
&\leq \dim_P(E \times F) && (9)\\
&\leq \dim_P(E) + \dim_P(F)\,. && (10)
\end{aligned}$$

Notice the superficial resemblance of this theorem to Theorem 4. This similarity is not a coincidence; each inequality in Theorem 9 follows from the corresponding line in Theorem 4. The arguments given here for (7–10) are each similar in length to the proof of (7) for Borel sets. That is, they are quite short.

Proof. Theorem 6 guarantees, for every $\varepsilon > 0$, that there exist an oracle set $A \subseteq \mathbb{N}$ and points $x \in E$ and $y \in F$ such that

$$\begin{aligned}
\dim_H(E \times F) &= \sup_{z\in E\times F} \dim^A(z)\,, && (11)\\
\dim^A(x) &\geq \dim_H(E) - \varepsilon\,,\\
\dim^{A,x}(y) &\geq \dim_H(F) - \varepsilon\,.
\end{aligned}$$

Then by (11), Theorem 4 relative to A, and Lemma 5 relative to A, we have

$$\begin{aligned}\dim_H(E \times F) &\geq \dim^A(x, y) \\ &\geq \dim^A(x) + \dim^A(y|x) \\ &\geq \dim^A(x) + \dim^{A,x}(y) \\ &\geq \dim_H(E) + \dim_H(F) - 2\varepsilon\,,\end{aligned}$$

by our choice of x and y. Since $\varepsilon > 0$ was arbitrary, we conclude that (7) holds.

For (8), let $\varepsilon > 0$ and use both parts of Theorem 6 to find $B, C \subseteq \mathbb{N}$, $u \in E$, and $v \in F$ such that

$$\begin{aligned}\dim_H(E) &= \sup_{x \in E} \dim^B(x)\,, \\ \dim_P(F) &= \sup_{y \in E} \mathrm{Dim}^C(y)\,, \\ \dim^{B,C}(u, v) &\geq \dim_H(E \times F) - \varepsilon\,.\end{aligned}$$

Since B and C minimize their respective expressions, we also have

$$\begin{aligned}\dim_H(E) &= \sup_{x \in E} \dim^{B,C}(x)\,, \\ \dim_P(F) &= \sup_{y \in E} \mathrm{Dim}^{B,C}(y)\,.\end{aligned}$$

Thus, we can apply Theorem 4 relative to B, C, after first noticing that conditioning on another point never increases dimension.

$$\begin{aligned}\dim_H(E) + \dim_P(F) &\geq \dim^{B,C}(u) + \mathrm{Dim}^{B,C}(v) \\ &\geq \dim^{B,C}(u|v) + \mathrm{Dim}^{B,C}(v) \\ &\geq \dim^{B,C}(u, v) \\ &\geq \dim_H(E \times F) - \varepsilon\,.\end{aligned}$$

Again, ε was arbitrary, so (8) holds.

For (9) and (10), we use essentially the same arguments as above. By Theorem 6, there are $A', B' \subseteq \mathbb{N}$, $x', u' \in E$, $y', v' \in F$, and $\varepsilon > 0$ that satisfy

$$\begin{aligned}\dim_P(E \times F) &= \sup_{z \in E \times F} \mathrm{Dim}^{A'}(z)\,, \\ \dim_H(E) &= \sup_{z \in E} \mathrm{Dim}^{B'}(z)\,, \\ \dim^{A'}(x') &\geq \dim_H(E) - \varepsilon\,, \\ \mathrm{Dim}^{A',x'}(y') &\geq \dim_P(F) - \varepsilon\,, \\ \mathrm{Dim}^{B',C}(u', v') &\geq \dim_P(E \times F) - \varepsilon\,,\end{aligned}$$

where x and C are as above. We once again apply relativized versions of Theorem 4 and

Lemma 5:

$$\begin{aligned}
\dim_P(E) + \dim_P(F) &\geq \mathrm{Dim}^{B',C}(u') + \mathrm{Dim}^{B',C}(v') \\
&\geq \mathrm{Dim}^{B',C}(u'|v') + \mathrm{Dim}^{B',C}(v') \\
&\geq \mathrm{Dim}^{B',C}(u', v') \\
&\geq \dim_P(E \times F) - \varepsilon \\
&\geq \mathrm{Dim}^{A'}(x', y') - \varepsilon \\
&\geq \dim^{A'}(x') + \mathrm{Dim}^{A'}(y'|x') - \varepsilon \\
&\geq \dim^{A'}(x') + \mathrm{Dim}^{A',x'}(y') - \varepsilon \\
&\geq \dim_H(E) + \dim_P(F) - 3\varepsilon\,.
\end{aligned}$$

Letting $\varepsilon \to 0$ completes the proof. ◀

6 Conclusion

The applications of theoretical computer science to pure mathematics in this paper yielded a significant extension to a basic theorem on Hausdorff dimension, as well as a much simpler argument for another such theorem. Understanding classical fractal dimensions as pointwise, algorithmic information theoretic quantities enables reasoning about them in a way that is both fine-grained and intuitive, and the proofs in this work are further evidence of the power and versatility of bounding techniques using Theorem 6. In particular, Theorem 1 demonstrates that this approach can be used to strengthen the foundations of fractal geometry. Therefore, in addition to further applications of these techniques, developing more refined results on the relationship between classical geometric measure theory and Kolmogorov complexity is an appealing direction for future investigations.

References

1 Krishna B. Athreya, John M. Hitchcock, Jack H. Lutz, and Elvira Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM Journal of Computing*, 37(3):671–705, 2007. doi:10.1137/s0097539703446912.

2 Verónica Becher, Jan Reimann, and Theodore A. Slaman. Irrationality exponent, Hausdorff dimension and effectivization. arXiv:1601.00153 [math.NT], 2016.

3 Christopher J. Bishop. Personal communication, April 27, 2017.

4 Christopher J. Bishop and Yuval Peres. *Fractals in Probability and Analysis*. Cambridge University Press, 2017. doi:10.1017/9781316460238.

5 Jin-Yi Cai and Juris Hartmanis. On Hausdorff and topological dimensions of the Kolmogorov complexity of the real line. *Journal of Computer and System Sciences*, 49(3):605–619, 1994. doi:10.1016/S0022-0000(05)80073-X.

6 Adam Case and Jack H. Lutz. Mutual dimension. *ACM Transactions on Computation Theory*, 7(3):12, 2015. doi:10.1145/2786566.

7 R. O. Davies. Some remarks on the Kakeya problem. *Proceedings of the Cambridge Philosophical Society*, 69:417–421, 1971. doi:10.1017/s0305004100046867.

8 Rod Downey and Denis Hirschfeldt. *Algorithmic Randomness and Complexity*. Springer-Verlag, 2010. doi:10.1007/978-0-387-68441-3.

9 Kenneth J. Falconer. *The Geometry of Fractal Sets*. Cambridge University Press, 1985. doi:10.1017/cbo9780511623738.

10 Kenneth J. Falconer. Sets with large intersection properties. *Journal of the London Mathematical Society*, 49(2):267–280, 1994. doi:10.1112/jlms/49.2.267.

11 Kenneth J. Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. Wiley, third edition, 2014. doi:10.1002/0470013850.

12 Felix Hausdorff. Dimension und äusseres Mass. *Mathematische Annalen*, 79:157–179, 1919. doi:10.1007/978-3-642-59483-0_2.

13 Jean-Pierre Kahane. Sur la dimension des intersections. In Jorge Alberto Barroso, editor, *Aspects of mathematics and its applications*, North-Holland Mathematical Library, 34, pages 419–430. Elsevier, 1986. doi:10.1016/s0924-6509(09)70272-7.

14 Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, third edition, 2008. doi:10.1007/978-0-387-49820-1.

15 Jack H. Lutz. The dimensions of individual strings and sequences. *Information and Computation*, 187(1):49–79, 2003. doi:10.1016/s0890-5401(03)00187-1.

16 Jack H. Lutz and Neil Lutz. Algorithmic information, plane Kakeya sets, and conditional dimension. In *Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8–11, 2017, Hannover, Germany*, pages 53:1–53:13, 2017. doi:10.4230/LIPIcs.STACS.2017.53.

17 Jack H. Lutz and Elvira Mayordomo. Dimensions of points in self-similar fractals. *SIAM Journal of Computing*, 38(3):1080–1112, 2008. doi:10.1007/978-3-540-69733-6_22.

18 Neil Lutz and D. M. Stull. Bounding the dimension of points on a line. In TV Gopal, Gerhard Jaeger, and Silvia Steila, editors, *Theory and Applications of Models of Computation: 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, pages 425–439, 2017. doi:10.1007/978-3-319-55911-7_31.

19 John M. Marstrand. Some fundamental geometrical properties of plane sets of fractional dimensions. *Proceedings of the London Mathematical Society*, 4(3):257–302, 1954. doi:10.1112/plms/s3-4.1.257.

20 Per Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602–619, 1966. doi:10.1016/s0019-9958(66)80018-9.

21 Pertti Mattila. Hausdorff dimension and capacities of intersections of sets in n-space. *Acta Mathematica*, 152:77–105, 1984. doi:10.1007/bf02392192.

22 Pertti Mattila. On the Hausdorff dimension and capacities of intersections. *Mathematika*, 32:213–217, 1985. doi:10.1112/s0025579300011001.

23 Pertti Mattila. *Geometry of sets and measures in Euclidean spaces: fractals and rectifiability*. Cambridge University Press, 1995. doi:10.1017/cbo9780511623813.

24 Elvira Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Inf. Process. Lett.*, 84(1):1–3, 2002. doi:10.1016/s0020-0190(02)00343-5.

25 Elvira Mayordomo. Effective fractal dimension in algorithmic information theory. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *New Computational Paradigms: Changing Conceptions of What is Computable*, pages 259–285. Springer New York, 2008. doi:10.1007/978-0-387-68546-5_12.

26 Ursula Molter and Ezequiel Rela. Furstenberg sets for a fractal set of directions. *Proceedings of the American Mathematical Society*, 140:2753–2765, 2012. doi:10.1090/s0002-9939-2011-11111-0.

27 Andre Nies. *Computability and Randomness*. Oxford University Press, Inc., New York, NY, USA, 2009. doi:10.1093/acprof:oso/9780199230761.001.0001.

28 Jan Reimann. *Computability and fractal dimension*. PhD thesis, Heidelberg University, 2004.

29 Jan Reimann. Effectively closed sets of measures and randomness. *Annals of Pure and Applied Logic*, 156(1):170–182, 2008. doi:10.1016/j.apal.2008.06.015.

30 Boris Ryabko. Noiseless coding of combinatorial sources. *Problems of Information Transmission*, 22:170–179, 1986.

31 Boris Ryabko. Algorithmic approach to the prediction problem. *Problems of Information Transmission*, 29:186–193, 1993.

32 Boris Ryabko. The complexity and effectiveness of prediction algorithms. *Journal of Complexity*, 10(3):281–295, 1994. doi:10.1006/jcom.1994.1015.

33 Ludwig Staiger. Kolmogorov complexity and Hausdorff dimension. *Information and Computation*, 103:159–194, 1989. doi:10.1007/3-540-51498-8_42.

34 Ludwig Staiger. A tight upper bound on Kolmogorov complexity and uniformly optimal prediction. *Theory of Computing Systems*, 31:215–229, 1998. doi:10.1007/s002240000086.

35 Elias M. Stein and Rami Shakarchi. *Real Analysis: Measure Theory, Integration, and Hilbert Spaces*. Princeton Lectures in Analysis. Princeton University Press, 2005.

36 Dennis Sullivan. Entropy, Hausdorff measures old and new, and limit sets of geometrically finite Kleinian groups. *Acta Mathematica*, 153(1):259–277, 1984. doi:10.1007/bf02392379.

37 Claude Tricot. Two definitions of fractional dimension. *Mathematical Proceedings of the Cambridge Philosophical Society*, 91(1):57–74, 1982. doi:10.1017/s0305004100059119.

Domains for Higher-Order Games*†

Matthew Hague[1], Roland Meyer[‡2], and Sebastian Muskalla[3]

1 **Royal Holloway University of London, United Kingdom**
`matthew.hague@rhul.ac.uk`
2 **TU Braunschweig, Germany**
`roland.meyer@tu-braunschweig.de`
3 **TU Braunschweig, Germany**
`s.muskalla@tu-braunschweig.de`

Abstract

We study two-player inclusion games played over word-generating higher-order recursion schemes. While inclusion checks are known to capture verification problems, two-player games generalize this relationship to program synthesis. In such games, non-terminals of the grammar are controlled by opposing players. The goal of the existential player is to avoid producing a word that lies outside of a regular language of safe words.

We contribute a new domain that provides a representation of the winning region of such games. Our domain is based on (functions over) potentially infinite Boolean formulas with words as atomic propositions. We develop an abstract interpretation framework that we instantiate to abstract this domain into a domain where the propositions are replaced by states of a finite automaton. This second domain is therefore finite and we obtain, via standard fixed-point techniques, a direct algorithm for the analysis of two-player inclusion games. We show, via a second instantiation of the framework, that our finite domain can be optimized, leading to a $(k+1)$EXP algorithm for order-k recursion schemes. We give a matching lower bound, showing that our approach is optimal. Since our approach is based on standard Kleene iteration, existing techniques and tools for fixed-point computations can be applied.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Higher-order recursion schemes, games, semantics, abstract interpretation, fixed points.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.59

1 Introduction

Inclusion checking has recently received considerable attention [53, 22, 1, 2, 36]. One of the reasons is a new verification loop, which invokes inclusion as a subroutine in an iterative fashion. The loop has been proposed by Podelski et al. for the safety verification of recursive programs [32], and then been generalized to parallel and parameterized programs [42, 20, 18] and to liveness [19]. The idea of Podelski's loop is to iteratively approximate unsound data flow in the program of interest, and add the approximations to the specification. Consider a program with control-flow language CF that is supposed to satisfy a safety specification

* The full version is available as technical report [28].
† This work was supported by the Engineering and Physical Sciences Research Council [EP/K009907/1]. The work instigated while some of the authors were visiting the Institute for Mathematical Sciences, National University of Singapore in 2016. The visit was partially supported by the Institute.
‡ A part of the work was carried out when the author was at Aalto University.

given by a regular language R. If the check $CF \subseteq R$ succeeds, then the program is correct as the data flow only restricts the set of computations. If a computation $w \in CF$ is found that lies outside R, then it depends on the data flow whether the program is correct. If data is handled correctly, w is a counterexample to R. Otherwise, w is generalized to a regular language S of infeasible computations. We set $R = R \cup S$ and repeat the procedure.

Podelski's loop has also been generalized to synthesis [35, 44]. In that setting, the program is assumed to have two kinds of non-determinism. Some of the non-deterministic transitions are understood to be controlled by the environment. They provide inputs that the system has to react to, and are also referred to as demonic non-determinism. In contrast, the so-called angelic non-determinism are the alternatives of the system to react to an input. The synthesis problem is to devise a controller that resolves the angelic non-determinism in a way that a given safety specification is met. Technically, the synthesis problem corresponds to a two-player perfect information game, and the controller implements a winning strategy for the system player. When generalizing Podelski's loop to the synthesis problem, the inclusion check thus amounts to solving a strategy-synthesis problem.

Our motivation is to synthesize functional programs with Podelski's loop. We assume the program to be given as a non-deterministic higher-order recursion scheme where the non-terminals are assigned to two players. One player is the system player who tries to enforce the derivation of words that belong to a given regular language. The other player is the environment, trying to derive a word outside the language. The use of the corresponding strategy-synthesis algorithm in Podelski's loop comes with three characteristics: (1) The algorithm is invoked iteratively, (2) the program is large and the specification is small, and (3) the specification is non-deterministic. The first point means that the strategy synthesis should not rely on costly precomputation. Moreover, it should have the chance to terminate early. The second says that the cost of the computation should depend on the size of the specification, not on the size of the program. Computations on the program, in particular iterative ones, should be avoided. Together with the third characteristic, these two consequences rule out reductions to reachability games. The required determinization would mean a costly precomputation, and the reduction to reachability would mean a product with the program. This discussion in particular forbids a reduction of the strategy-synthesis problem to higher-order model checking [45], which indeed can be achieved (see the full version [28] for a comparison to intersection types [41]). Instead, we need a strategy synthesis that can directly deal with non-deterministic specifications.

We show that the winning region of a higher-order inclusion game wrt. a non-deterministic right-hand side can be computed with a standard fixed-point iteration. Our contribution is a domain suitable for this computation. The key idea is to use Boolean formulas whose atomic propositions are the states of the targeted finite automaton. While a formula-based domain has recently been proposed for context-free inclusion games [35] (and generalized to infinite words [44]), the generalization to higher-order is new. Consider a non-terminal that is ground and for which we have computed a formula. The Boolean structure reflects the alternation among the players in the plays that start from this non-terminal. The words generated along the plays are abstracted to sets of states from which these words can be accepted. Determining the winner of the game is done by evaluating the formula when sets of states containing the initial state are assigned the value true. To our surprise, the above domain did not give the optimal complexity. Instead, it was possible to further optimize it by resolving the determinization information. Intuitively, the existential player can also resolve the non-determinism captured by a set. Crucially, our approach handles the non-determinism of the specification inside the analysis, without preprocessing.

Besides offering the characteristics that are needed for Podelski's loop, our development also contributes to the research program of *effective denotational semantics*, as recently proposed by Salvati and Walukiewicz [51] as well as Grellois and Melliès [24, 24], with [5, 48] being early works in this field. The idea is to solve verification problems by computing the semantics of a program in a suitable domain. Salvati and Walukiewicz studied the expressiveness of greatest fixed-point semantics and their correspondence to automata [51], and constructions of enriched Scott models for parity conditions [50, 49]. A similar line of investigation has been followed in recent work by Grellois and Melliès [25, 26]. Hofmann and Chen considered the verification of more restricted ω-path properties with a focus on the domain [33]. They show that explicit automata constructions can be avoided and give a domain that directly captures subsets (so-called patches) of the ω-language. The work has been generalized to higher order [34]. Our contribution is related in that we focus on the domain (suitable for capturing plays).

Besides the domain, the correctness proof may be of interest. We employ an exact fixed-point transfer result as known from abstract interpretation. First, we give a semantic characterization showing that the winning region can be captured by an infinite model (a greatest fixed point). This domain has as elements (potentially infinite) sets of (finite) Boolean formulas. The formulas capture plays (up to a certain depth) and the atomic propositions are terminal words. The infinite set structure is to avoid infinite syntax. Then we employ the exact fixed-point transfer result to replace the terminals by states and get rid of the sets. The final step is another exact fixed-point transfer that justifies the optimization. We give a matching lower bound. The problem is $(k+1)\mathsf{EXP}$-complete for order-k schemes.

Related Work. The relationship between recursion schemes and extensions of pushdown automata has been well studied [16, 17, 37, 29]. This means algorithms for recursion schemes can be transferred to extensions of pushdown automata and vice versa. In the sequel, we will use *pushdown automata* to refer to pushdown automata and their family of extensions.

The decidability of Monadic Second Order Logic (MSO) over trees generated by recursion schemes was first settled in the restricted case of *safe* schemes by Knapik *et al.* [37] and independently by Caucal [14]. This result was generalized to all schemes by Ong [45]. Both of these results consider *deterministic* schemes only.

Related results have also been obtained in the consideration of games played over the configuration graphs of pushdown automata [52, 13, 38, 29]. Of particular interest are *saturation* methods for pushdown games [7, 21, 12, 8, 30, 31, 9]. In these works, automata representing sets of winning configurations are constructed using fixed-point computations.

A related approach pioneered by Kobayashi et al. operating directly on schemes is that of *intersection types* [40, 41], where types embedding a property automaton are assigned to terms of a scheme. Recently, saturation techniques were transferred to intersection types by Broadbent and Kobayashi [10]. The typing algorithm is then a least fixed-point computation analogous to an optimized version of our Kleene iteration, restricted to deterministic schemes. This has led to one of the most competitive model-checking tools for schemes [39].

One may reduce our language inclusion problems to many of the above works. E.g. from an inclusion game for schemes, we may build a game over an equivalent kind of pushdown automaton and take the product with a determinization of the NFA. This obtains a reachability game over a pushdown automaton that can be solved by any of the above methods. However, such constructions are undesirable for iterative invocations as in Podelski's loop.

We already discussed the relationship to model-theoretic verification algorithms. Abstract interpretation has also been used by Ramsay [47], Salvati and Walukiewicz [50, 49], and

Grellois and Melliès [24, 23] for verification. The former used a Galois connection between safety properties (concrete) and equivalence classes of intersection types (abstract) to recreate decidability results known in the literature. The latter two strands gives a semantics capable of computing properties expressed in MSO. Indeed, abstract interpretation has long been used for static analysis of higher-order programs [4].

2 Preliminaries

Complete Partial Orders. Let $(D, \leq)$ be a *partial order* with set D and (partial) ordering $\leq$ on D. We call $(D, \leq)$ *pointed* if there is a greatest element, called the *top element* and denoted by $\top \in D$. A *descending chain* in D is a sequence $(d_i)_{i\in\mathbb{N}}$ of elements in D with $d_i \geq d_{i+1}$. We call $(D, \leq)$ *ω-complete* if every descending chain has a greatest lower bound, called the *meet* or the *infimum*, and denoted by $\bigsqcap_{i\in\mathbb{N}} d_i$. If $(D, \leq)$ is pointed and ω-complete, we call it a *pointed ω-complete partial order (cppo)*. In the following, we will only consider partial orders that are cppos. Note, cppo is usually used to refer to the dual concept, i.e. partial orders with a least element and least upper bounds for ascending chains.

A function $f : D \to D$ is *$\sqcap$-continuous* if for all descending chains $(d_i)_{i\in\mathbb{N}}$ we have $f(\bigsqcap_{i\in\mathbb{N}} d_i) = \bigsqcap_{i\in\mathbb{N}} f(d_i)$. We call a function $f : D \to D$ *monotonic* if for all $d, d' \in D$, $d \leq d'$ implies $f(d) \leq f(d')$. Any function that is $\sqcap$-continuous is also monotonic. For a monotonic function, $\top \geq f(\top) \geq f^2(\top) = f(f(\top)) \geq f^3(\top) \geq \dots$ is a descending chain.

If the function is $\sqcap$-continuous, then $\bigsqcap_{i\in\mathbb{N}} f^i(\top)$ is by Kleene's theorem the greatest fixed point of f, i.e. $f(\bigsqcap_{i\in\mathbb{N}} f^i(\top)) = \bigsqcap_{i\in\mathbb{N}} f^i(\top)$ and $\bigsqcap_{i\in\mathbb{N}} f^i(\top)$ is larger than any other element d with $f(d) = d$. We also say $\bigsqcap_{i\in\mathbb{N}} f^i(\top)$ is the greatest solution to the equation $x = f(x)$.

A lattice satisfies the *descending chain condition (DCC)* if every descending chain has to be stationary at some point. In this case $\bigsqcap_{i\in\mathbb{N}} f^i(\top) = \bigsqcap_{i=0}^{i_0} f^i(\top)$ for some index i_0 in $\mathbb{N}$. With this, we can compute the greatest fixed point: Starting with $\top$, we iteratively apply f until the result does not change. This process is called *Kleene iteration*. Note that finite cppos, i.e. with finitely many elements in D, trivially satisfy the descending chain condition.

Finite Automata. A *non-deterministic finite automaton (NFA)* is a tuple $A = (Q_{NFA}, \Gamma, \delta, q_0, Q_f)$ where Q_{NFA} is a finite set of states, Γ is a finite alphabet, $\delta \subseteq Q_{NFA} \times \Gamma \times Q_{NFA}$ is a (non-deterministic) transition relation, $q_0 \in Q_{NFA}$ is the initial state, and $Q_f \subseteq Q_{NFA}$ is a set of final states. We write $q \xrightarrow{a} q'$ to denote $(q, a, q') \in \delta$. Moreover, given a word $w = a_1 \cdots a_\ell$, we write $q \xrightarrow{w} q'$ whenever there is a sequence of transitions, also called *run*, $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \cdots \xrightarrow{a_\ell} q_{\ell+1}$ with $q_1 = q$ and $q_{\ell+1} = q'$. The run is accepting if $q = q_0$ and $q' \in Q_f$. The language of A is $\mathcal{L}(A) = \{w \mid q_0 \xrightarrow{w} q \in Q_f\}$.

3 Higher-Order Recursion Schemes

We introduce higher-order recursion schemes, *schemes* for short, following the presentation in [27]. Schemes can be understood as grammars generating the computation trees of programs in a functional language. As is common in functional languages, we need a typing discipline. To avoid confusion with type-based approaches to higher-order model checking [40, 46, 41], we refer to types as *kinds*. Kinds define the functionality of terms, without specifying the data domain. Technically, the only data domain is the ground kind o, from which (potentially higher-order) function kinds are derived by composition:

$$\kappa \ ::= \ o \ \mid \ (\kappa_1 \to \kappa_2) \ .$$

We usually omit the brackets and assume that the arrow associates to the right. The number of arguments to a kind is called the *arity*. The *order* defines the functionality of the arguments: A first-order kind defines functions that act on values, a second-order kind functions that expect functions as parameters. Formally, we have

$$\mathsf{arity}(o) = 0, \qquad \mathsf{order}(o) = 0,$$
$$\mathsf{arity}(\kappa_1 \to \kappa_2) = \mathsf{arity}(\kappa_2) + 1, \qquad \mathsf{order}(\kappa_1 \to \kappa_2) = \max(\mathsf{order}(\kappa_1) + 1, \mathsf{order}(\kappa_2)) \ .$$

Let K be the set of all kinds. Higher-order recursion schemes assign kinds to symbols from different alphabets, namely non-terminals, terminals, and variables. Let Γ be a set of such *kinded symbols*. For each kind κ, we denote by Γ^κ the restriction of Γ to the symbols with kind κ. The *terms* $\mathcal{T}^\kappa(\Gamma)$ of kind κ over Γ are defined by simultaneous induction over all kinds. They form the smallest set satisfying

1. $\Gamma^\kappa \subseteq \mathcal{T}^\kappa(\Gamma)$,
2. $\bigcup_{\kappa_1} \{t\ v \mid t \in \mathcal{T}^{\kappa_1 \to \kappa_2}(\Gamma), v \in \mathcal{T}^{\kappa_1}(\Gamma)\} \subseteq \mathcal{T}^{\kappa_2}(\Gamma)$, and
3. $\{\lambda x.t \mid x \in \mathcal{T}^{\kappa_1}(\Gamma), t \in \mathcal{T}^{\kappa_2}(\Gamma)\} \subseteq \mathcal{T}^{\kappa_1 \to \kappa_2}(\Gamma)$.

If term t is of kind κ, we also write $t\colon \kappa$. We use $\mathcal{T}(\Gamma)$ for the set of all terms over Γ. We say a term is λ-*free* if it contains no sub-term of the form $\lambda x.t$. A term is *variable-closed* if all occurring variables are bound by a preceding λ-expression.

▶ **Definition 1.** A *higher-order recursion scheme*, (*scheme* for short), is a tuple $G = (V, N, T, R, S)$, where V is a finite set of kinded symbols called *variables*, T is a finite set of kinded symbols called *terminals*, and N is a finite set of kinded symbols called *non-terminals* with $S \in N$ the *initial symbol*. The sets V, T, and N are pairwise disjoint. The finite set R consists of *rewriting rules* of the form $F = \lambda x_1 \dots \lambda x_n.e$, where $F \in N$ is a non-terminal of kind $\kappa_1 \to \dots \kappa_n \to o$, $x_1, \dots, x_n \in V$ are variables of the required kinds, and e is a λ-free, variable-closed term of ground kind from $\mathcal{T}^o(T \uplus N \uplus \{x_1\colon \kappa_1, \dots, x_n\colon \kappa_n\})$.

The semantics of G is defined by rewriting subterms according to the rules in R. A *context* is a term $C[\bullet] \in \mathcal{T}(\Gamma \uplus \{\bullet\colon o\})$ in which $\bullet$ occurs exactly once. Given a context $C[\bullet]$ and a term $t : o$, we obtain $C[t]$ by replacing the unique occurrence of $\bullet$ in $C[\bullet]$ by t. With this, $t \Rightarrow_G t'$ if there is a context $C[\bullet]$, a rule $F = \lambda x_1 \dots \lambda x_n.e$, and a term $F\ t_1\ \dots\ t_n : o$ such that $t = C[F\ t_1\ \dots\ t_n]$ and $t' = C[e[x_1 \mapsto t_1, \dots, x_n \mapsto t_n]]$. In other words, we replace one occurrence of F in t by a right-hand side of a rewriting rule, while properly instantiating the variables. We call such a replaceable $F\ t_1\ \dots\ t_n$ a *reducible expression (redex)*. The rewriting step is *outermost to innermost (OI)* if there is no redex that contains the rewritten one as a proper subterm. The OI-language $\mathcal{L}(G)$ of G is the set of all (finite, ranked, labeled) trees T over the terminal symbols that can be created from the initial symbol S via OI-rewriting steps. We will restrict the rewriting relation to OI-rewritings in the rest of this paper. Note, all words derivable by IO-rewriting are also derivable with OI-rewriting.

Word-Generating Schemes. We consider *word-generating schemes*, i.e. schemes with terminals $T \uplus \{\$: o\}$ where exactly one terminal symbol \$ has kind o and all others are of kind $o \to o$. The generated trees have the shape $a_1\ (a_2\ (\cdots\ (a_k\ \$)))$, which we understand as the finite word $a_1 a_2 \dots a_k \in T^*$. We also see $\mathcal{L}(G)$ as a language of finite words.

Determinism. The above schemes are non-deterministic in that several rules may rewrite a non-terminal. We associate with a non-deterministic scheme $G = (V, N, T, R, S)$ a deterministic scheme G^{det} with exactly one rule per non-terminal. Intuitively, G^{det} makes the non-determinism explicit with new terminal symbols.

Formally, let $F : \kappa$ be a non-terminal with rules $F = t_1$ to $F = t_\ell$. We may assume each $t_i = \lambda x_1 \dots \lambda x_k . e_i$, where e_i is λ-free. We introduce a new terminal symbol $op_F : o \to o \to \dots \to o$ of arity ℓ. Let the set of all these terminals be $T^{det} = \{ op_F \mid F \in N \}$. The set of rules R^{det} now consists of a single rule for each non-terminal, namely $F = \lambda x_1 \dots \lambda x_k . op_F\ e_1 \cdots e_\ell$. The original rules in R are removed. This yields $G^{det} = (V, N, T \cup T^{det}, R^{det}, S)$. The advantage of resolving the non-determinism explicitly is that we can give a semantics to non-deterministic choices that depends on the non-terminal instead of having to treat non-determinism uniformly.

Semantics. Let $G = (V, N, T, R, S)$ be a deterministic scheme. A *model* of G is a pair $\mathcal{M} = (\mathcal{D}, \mathcal{I})$, where $\mathcal{D}$ is a family of domains $(\mathcal{D}(\kappa))_{\kappa \in K}$ that satisfies the following: $\mathcal{D}(o)$ is a cppo and $\mathcal{D}(\kappa_1 \to \kappa_2) = \mathit{Cont}(\mathcal{D}(\kappa_1), \mathcal{D}(\kappa_2))$. Here, $\mathit{Cont}(A, B)$ is the set of all $\sqcap$-continuous functions from domain A to B. We comment on this cppo in a moment. The interpretation $\mathcal{I} : T \to \mathcal{D}$ assigns to each terminal $s : \kappa$ an element $\mathcal{I}(s) \in \mathcal{D}(\kappa)$.

The ordering on functions is defined component-wise, $f \leq_{\kappa_1 \to \kappa_2} g$ if $(f\ x) \leq_{\kappa_2} (g\ x)$ for all $x \in \mathcal{D}(\kappa_1)$. For each κ, we denote the top element of $\mathcal{D}(\kappa)$ by $\top_\kappa$. For the ground kind, $\top_o$ exists since $\mathcal{D}(\kappa)$ is a cppo, and $\top_{\kappa_1 \to \kappa_2}$ is the function that maps every argument to $\top_{\kappa_2}$. The meet of a descending chain of functions $(f_i)_{i \in \mathbb{N}}$ is the function defined by $(\bigsqcap_{\kappa_1 \to \kappa_2} (f_i)_{i \in \mathbb{N}})\ x = \bigsqcap_{\kappa_2} (f_i\ x)_{i \in \mathbb{N}}$. Note that the sequence on the right-hand side is a descending chain.

The *semantics of terms* defined by a model is a function

$$\mathcal{M}[\![-]\!] : \mathcal{T} \to (N \cup V \nrightarrow \mathcal{D}) \to \mathcal{D} \ .$$

that assigns to each term built over the non-terminals and terminals again a function. This function expects a valuation $\nu : N \cup V \nrightarrow \mathcal{D}$ and returns an element from the domain. A valuation is a partial function that is defined on all non-terminals and the free variables. We lift $\sqcap$ to descending chains of valuations with $(\bigsqcap_{i \in \mathbb{N}} \nu_i)(y) = \bigsqcap_{i \in \mathbb{N}} (\nu_i(y))$ for $y \in N \cup V$. We obtain that the set of such valuations is a cppo where the greatest elements are those valuations which assign the greatest elements of the appropriate domain to all arguments.

Since the right-hand sides of the rules in the scheme are variable-closed, we do not need a variable valuation for them. We need the variable valuation, however, whenever we proceed by induction on the structure of terms. The semantics is defined by such an induction:

$$\mathcal{M}[\![s]\!]\ \nu = \mathcal{I}(s) \qquad \mathcal{M}[\![F]\!]\ \nu = \nu(F) \qquad \mathcal{M}[\![t_1\ t_2]\!]\ \nu = (\mathcal{M}[\![t_1]\!]\ \nu)\ (\mathcal{M}[\![t_2]\!]\ \nu)$$
$$\mathcal{M}[\![x]\!]\ \nu = \nu(x) \qquad \mathcal{M}[\![\lambda x : \kappa . t_1]\!]\ \nu = d \in \mathcal{D}(\kappa) \mapsto \mathcal{M}[\![t_1]\!]\ \nu[x \mapsto d] \ .$$

We show that $\mathcal{M}[\![t]\!]$ is $\sqcap$-continuous for all terms t. This follows from continuity of the functions in the domain, but requires some care when handling application.

▶ **Proposition 2.** *For all t, $\mathcal{M}[\![t]\!]$ is $\sqcap$-continuous (in ν) over the respective lattice.*

Given $\mathcal{M}$, the rules $F_1 = t_1, \dots, F_k = t_k$ of the (deterministic) scheme give a function

$$\mathit{rhs}_\mathcal{M} : (N \to \mathcal{D}) \to (N \to \mathcal{D}) \ , \quad \text{where} \quad \mathit{rhs}_\mathcal{M}(\nu)(F_j) = \mathcal{M}[\![t_j]\!]\ \nu \ .$$

Since the right-hand sides are variable-closed, the $\mathcal{M}[\![t_j]\!]$ are functions in the non-terminals. Provided $\mathcal{M}[\![t_1]\!]$ to $\mathcal{M}[\![t_k]\!]$ are $\sqcap$-continuous (in the valuation of the non-terminals), the function $\mathit{rhs}_\mathcal{M}$ will be $\sqcap$-continuous. This allows us to apply Kleene iteration as follows. The initial value is the greatest element $\sigma^0_\mathcal{M}$ where $\sigma^0_\mathcal{M}(F_j) = \top_j$ with $\top_j$ the top element of $\mathcal{D}(\kappa_j)$. The $(i+1)^{\text{th}}$ approximant is computed by evaluating the right-hand side at the i^{th}

solution, $\sigma_{\mathcal{M}}^{i+1} = rhs_{\mathcal{M}}(\sigma_{\mathcal{M}}^{i})$. The greatest fixed point is the tuple $\sigma_{\mathcal{M}}$ defined below. It can be understood as the greatest solution to the equation $\nu = rhs_{\mathcal{M}}(\nu)$. We call this greatest solution $\sigma_{\mathcal{M}}$ the *semantics of the scheme* in the model.

$$\sigma_{\mathcal{M}} = \bigsqcap_{i \in \mathbb{N}} \sigma_{\mathcal{M}}^{i} = \bigsqcap_{i \in \mathbb{N}} rhs_{\mathcal{M}}^{i}(\sigma_{\mathcal{M}}^{0})$$

4 Higher-Order Inclusion Games

Our goal is to solve higher-order games, whose arena is defined by a scheme. We assume that the derivation process is controlled by two players. To this end, we divide the non-terminals of a word-generating scheme into those owned by the existential player $\Diamond$ and those owned by the universal player $\Box$. Whenever a non-terminal is to be replaced during the derivation, it is the owner who chooses which rule to apply. The winning condition is given by an automaton A, Player $\Diamond$ attempts to produce a word that is in $\mathcal{L}(A)$, while Player $\Box$ attempts to produce a word outside of $\mathcal{L}(A)$.

▶ **Definition 3.** A *higher-order game* is a triple $\mathcal{G} = (G, A, O)$ where G is a word-generating scheme, A is an NFA, $O : N \to \{\Diamond, \Box\}$ is a partitioning of the non-terminals of G.

A play of the game is a sequence of OI-rewriting steps. Since terms generate words, it is unambiguous which term forms the next redex to be rewritten. In particular, all terms are of the form $a_1(a_2(\cdots(a_k(t))))$, where t is either \$ or a redex $F\ t_1\ \cdots\ t_m$. If $O(F) = \Diamond$ then Player $\Diamond$ chooses a rule $F = \lambda x_1 \ldots \lambda x_m.e$ to apply, else Player $\Box$ chooses the rule. This moves the play to $a_1\ (a_2\ (\cdots\ (a_k\ e[x_1 \mapsto t_1, \ldots, x_m \mapsto t_m])))$.

Each play begins at the initial non-terminal S, and continues either ad infinitum or until a term $a_1\ (a_2\ (\cdots\ (a_k\ \$)))$, understood as the word $w = a_1 \ldots a_k$, is produced. Infinite plays do not produce a word and are won by Player $\Diamond$. Finite maximal plays produce such a word w. Player $\Diamond$ wins whenever $w \in \mathcal{L}(A)$, Player $\Box$ wins if $w \in \overline{\mathcal{L}(A)}$. Since the winning condition is Borel, either Player $\Diamond$ or Player $\Box$ has a winning strategy [43].

The Winner of a Higher-Order Game (HOG)	
Input:	A higher-order game $\mathcal{G}$.
Question:	Does Player $\Diamond$ win $\mathcal{G}$? If so, effectively represent Player $\Diamond$'s strategy.

Our contribution is a fixed-point algorithm to decide HOG. We derive it in three steps. First, we develop a concrete model for higher-order games whose semantics captures the above winning condition. Second, we introduce a framework that for two models and a mapping between them guarantees that the mapping of the greatest fixed point with respect to the one model is the greatest fixed point with respect to the other model. Finally, we introduce an abstract model that uses a finite ground domain. The solution of HOG can be read off from the semantics in the abstract model, which in turn can be computed via Kleene iteration. Moreover, this semantics can be used to define Player $\Diamond$'s winning strategy. We instantiate the framework for the concrete and abstract model to prove the soundness of the algorithm.

Concrete Semantics

Consider a HOG instance $\mathcal{G} = (G, A, O)$. Let G^{det} be the determinized version of G. Our goal is to define a model $\mathcal{M}^C = (\mathcal{D}^C, \mathcal{I}^C)$ such that the semantics of G^{det} in this model allows us to decide HOG. Recall that we only have to define the ground domain. For composed kinds, we use the functional lifting discussed in Section 3.

Our idea is to associate to kind o the set of positive Boolean formulas where the atomic propositions are words in T^*. To be able to reuse the definition, we define formula domains in more generality as follows.

Domains of Boolean Formulas. Given a (potentially infinite) set P of atomic propositions, the *positive Boolean formulas* $\mathsf{PBool}(P)$ over P are defined to contain true, every p from P, and compositions of formulas via conjunction and disjunction. We work up to logical equivalence, which means we treat ϕ_1 and ϕ_2 as equal as long as they are logically equivalent.

Unfortunately, if the set P is infinite, $\mathsf{PBool}(P)$ is not a cppo, because the meet of a descending chain of formulas might not be a finite formula. The idea of our domain is to have conjunctions of infinitely many formulas. As is common in logic, we represent them as infinite sets. Therefore, we consider the set of all sets of (finite) positive Boolean formulas $\mathcal{P}(\mathsf{PBool}(T^*)) \setminus \{\emptyset\}$ factorized modulo logical equivalence, denoted $(\mathcal{P}(\mathsf{PBool}(T^*)) \setminus \{\emptyset\})/_{\Leftrightarrow}$. To be precise, the sets may be finite or infinite, but they must be non-empty.

To define the factorization, let an assignment to the atomic propositions be given by a subset of $P' \subseteq P$. The atomic proposition p is true if $p \in P'$. An assignment satisfies a Boolean formula, if the formula evaluates to true in that assignment. It satisfies a set of Boolean formulas, if it satisfies all elements. Given two sets of formulas Φ_1 and Φ_2, we write $\Phi_1 \Rightarrow \Phi_2$, if every assignment that satisfies Φ_1 also satisfies Φ_2. Two sets of formulas are equivalent, denoted $\Phi_1 \Leftrightarrow \Phi_2$, if $\Phi_1 \Rightarrow \Phi_2$ and $\Phi_2 \Rightarrow \Phi_1$ holds.

The ordering on these factorized sets is implication (which by transitivity is independent of the representative). The top element is the set $\{\mathsf{true}\}$, which is implied by every set. The conjunction of two sets is union. Note that it forms the meet in the partial order, and moreover note that meets over arbitrary sets exist, in particular the domain is a cppo. We will also need an operation of disjunction, which is defined by $\Phi_1 \vee \Phi_2 = \{\phi_1 \vee \phi_2 \mid \phi_1 \in \Phi_1, \phi_2 \in \Phi_2\}$. We will also use disjunctions of higher (but finite) arity where convenient. Note that the disjunction on finite formulas is guaranteed to result in a finite formula. Therefore, the above is well-defined.

In our case, the assignment $P' \subseteq T^*$ of interest is the language of the automaton A. Player $\Diamond$ will win the game iff the concrete semantics assigns a set of formulas to S that is satisfied by $\mathcal{L}(A)$.

The Concrete Domains and Interpretation of Terminals. From a ground domain, higher-order domains are defined as continuous functions as in Section 3. Thus we only need

$$\mathcal{D}^C(o) = (\mathcal{P}(\mathsf{PBool}(T^*)) \setminus \{\emptyset\})/_{\Leftrightarrow} \ .$$

The endmarker \$ yields the set of formulas $\{\varepsilon\}$, i.e. $\mathcal{I}^C(\$) = \{\varepsilon\}$. A terminal $a : o \to o$ prepends a to a given word w. That is $\mathcal{I}^C(a) = \mathsf{prepend}_a$, where $\mathsf{prepend}_a$ distributes over conjunction and disjunction:

$$\mathsf{prepend}_a(\phi) = \begin{cases} aw & \phi = w \ , \\ \mathsf{prepend}_a(\phi_1)\ op\ \mathsf{prepend}_a(\phi_2) & \phi = \phi_1\ op\ \phi_2 \text{ and } \ op \in \{\wedge, \vee\} \ , \\ \phi & \phi = \mathsf{true} \ . \end{cases}$$

We apply $\mathsf{prepend}_a$ to sets of formulas by applying it to every element. Finally, $\mathcal{I}^C(op_F)$ where op_F has arity ℓ is an ℓ-ary conjunction (resp. disjunction) if Player $\Box$ (resp. $\Diamond$) owns F.

For $\mathcal{M}^C = (\mathcal{D}^C, \mathcal{I}^C)$ to be a model, we need our interpretation of terminals to be $\sqcap$-continuous. This follows largely by the distributivity of our definitions.

▶ **Lemma 4.** *For all non-ground terminals s, $\mathcal{I}^C(s)$ is $\sqcap$-continuous.*

▶ **Example 5.** Consider the higher-order game defined by the scheme $S = H\ a\ \$ \mid b\ \$$ and $H = \lambda f.\lambda x.f\ (f\ x) \mid \lambda f.\lambda x.H\ (H\ f)\ x$. Assume S is owned by Player $\Diamond$ and H is owned by Player $\Box$. Let the automaton accept the language $\{b\}$. Player $\Diamond$ can choose to rewrite S to $b\ \$$ and therefore has a strategy to produce a word in the language. To derive this information from the concrete semantics, we compute $\sigma_{\mathcal{M}^C}(H)$. It is the function mapping $f \in \mathit{Cont}(\mathcal{D}^C(o), \mathcal{D}^C(o))$ and $d \in \mathcal{D}^C(o)$ to $\bigcup_{k>0} f^{2k}(d)$. Note that the union is the conjunction of sets of formulas, which is the interpretation of op_H for the universal player. Moreover, note that due to non-determinism we obtain all even numbers of applications of f, not only the powers of 2. With this, the semantics of the initial symbol is

$$\sigma_{\mathcal{M}^C}(S) = \bigcup_{k>0} \mathsf{prepend}_a^{2k}(\{\varepsilon\}) \vee \mathsf{prepend}_b(\{\varepsilon\}) = \{a^{2k} \vee b \mid k > 0\}.$$

The assignment $\{b\}$ given by the language of the NFA satisfies $\{a^{2k} \vee b \mid k > 0\}$. Indeed, since b evaluates to true, every formula in the set evaluates to true.

Correctness of Semantics and Winning Strategies. We need to show that the concrete semantics matches the original semantics of the game.

▶ **Theorem 6.** *$\sigma_{\mathcal{M}^C}(S)$ is satisfied by $\mathcal{L}(A)$ iff there is a winning strategy for Player $\Diamond$.*

When $\sigma_{\mathcal{M}^C}(S)$ is satisfied by $\mathcal{L}(A)$ the concrete semantics gives a winning strategy for $\Diamond$: From a term t such that $\mathcal{M}^C[\![t]\!]\ \sigma_{\mathcal{M}^C}$ is satisfied by $\mathcal{L}(A)$, Player $\Diamond$, when able to choose, picks a rewrite rule that transforms t to t', where $\mathcal{M}^C[\![t']\!]\ \sigma_{\mathcal{M}^C}$ remains satisfied. The proof of Theorem 6 shows this is always possible, and, moreover, Player $\Box$ is unable to reach a term for which satisfaction does not hold. This does not yet give an effective strategy since we cannot compute $\mathcal{M}^C[\![t]\!]\ \sigma_{\mathcal{M}^C}$. However, the abstract semantics will be computable, and can be used in place of the concrete semantics by Player $\Diamond$ to implement the winning strategy.

The proof that $\sigma_{\mathcal{M}^C}(S)$ being unsatisfied implies a winning strategy for Player $\Box$ is more involved and requires the definition of a correctness relation between semantics and terms that is lifted to the level of functions, and shown to hold inductively.

5 Framework for Exact Fixed-Point Transfer

The concrete model $\mathcal{M}^C$ does not lead to an algorithm for solving HOG since its domains are infinite. Here, we consider an abstract model $\mathcal{M}^A$ with finite domains. The soundness of the resulting Kleene iteration relies on the two semantics being related by a precise abstraction α. Since both semantics are defined by fixed points, this requires us to prove $\alpha(\sigma_{\mathcal{M}^C}) = \sigma_{\mathcal{M}^A}$. In this section, we provide a general framework to this end.

Consider the deterministic scheme G together with two models (left and right) $\mathcal{M}_l = (\mathcal{D}_l, \mathcal{I}_l)$ and $\mathcal{M}_r = (\mathcal{D}_r, \mathcal{I}_r)$. Our goal is to relate the semantics in these models in the sense that $\sigma_{\mathcal{M}_r} = \alpha(\sigma_{\mathcal{M}_l})$. Such exact fixed-point transfer results are well-known in abstract interpretation. To generalize them to higher-order we give easy to instantiate conditions on α, $\mathcal{M}_l$, and $\mathcal{M}_r$ that yield the above equality. Interestingly, exact fixed-point transfer results seem to be rare for higher-order (e.g. [46]). Our development is inspired by Abramsky's lifting of abstraction functions to logical relations [3], which generalizes [11, 4]. These works focus on approximation and the compatibility we need for exactness is missing. Our framework is easier to apply than [15, 6], which are again concerned with approximation and do not offer (but may lead to) exact fixed-point transfer results.

For the terminology, an *abstraction* is a function $\alpha : \mathcal{D}_l(o) \to \mathcal{D}_r(o)$. To lift the abstraction to function domains, we define the notion of *being compatible with* α. Compatibility intuitively states that the function on the concrete domain is not more precise than what the abstraction function distinguishes. This allows us to define the abstraction of a function by applying the function and abstracting the result, $\alpha(f)\ \alpha(v_l) = \alpha(f\ v_l)$. Compatibility ensures the independence of the choice of v_l.

By definition, all ground elements $v_l \in \mathcal{D}_l(o)$ are compatible with α. For function domains, compatibility and the abstraction are defined as follows.

▶ **Definition 7.** Assume α and the notion of compatibility are defined on $\mathcal{D}_l(\kappa_1)$ and $\mathcal{D}_l(\kappa_2)$. Let $\top^l_\kappa$ (resp. $\top^r_\kappa$) be the greatest element of $\mathcal{D}_l(\kappa)$ (resp. $\mathcal{D}_r(\kappa)$) for each κ.

1. Function $f \in \mathcal{D}_l(\kappa_1 \to \kappa_2)$ is compatible with α, if
   a. for all compatible $v_l, v_l' \in \mathcal{D}_l(\kappa_1)$ with $\alpha(v_l) = \alpha(v_l')$ we have $\alpha(f\ v_l) = \alpha(f\ v_l')$, and
   b. for all compatible $v_l \in \mathcal{D}_l(\kappa_1)$ we have that $f\ v_l$ is compatible.
2. We define $\alpha(f) \in \mathcal{D}_r(\kappa_1 \to \kappa_2)$ as follows.
   a. If f is compatible, we set $\alpha(f)\ v_r = \alpha(f\ v_l)$, provided there is a compatible $v_l \in \mathcal{D}_l(\kappa_1)$ with $v_r = \alpha(v_l)$, and $\alpha(f)\ v_r = \top^r_{\kappa_2}$ otherwise.
   b. If f is not compatible, $\alpha(f) = \top^r_{\kappa_1 \to \kappa_2}$.

We lift α to valuations $\nu : N \uplus V \nrightarrow \mathcal{D}_l$ by $\alpha(\nu)(F) = \alpha(\nu(F))$ and similar for x. We also lift compatibility to valuations $\nu : N \uplus V \nrightarrow \mathcal{D}_l$ by requiring $\nu(F)$ to be compatible for all $F \in N$ and similar for $x \in V$.

The conditions needed for the exact fixed-point transfer are the following.

▶ **Definition 8.** Function α is *precise* for $\mathcal{M}_l$ and $\mathcal{M}_r$, if

(P1) $\alpha(\mathcal{D}_l(o)) = \mathcal{D}_r(o)$,

(P2) $\alpha : \mathcal{D}_l(o) \to \mathcal{D}_r(o)$ is $\sqcap$-continuous,

(P3) $\alpha(\top^l_o) = \top^r_o$,

(P4) $\alpha(\mathcal{I}_l(s)) = \mathcal{I}_r(s)$ for all terminals $s\colon o$, and similarly $\alpha(\mathcal{I}_l(s)\ v_l) = \mathcal{I}_r(s)\ \alpha(v_l)$ for all terminals $s : \kappa_1 \to \kappa_2$ and all compatible $v_l \in \mathcal{D}_l(\kappa_1)$,

(P5) $\mathcal{I}_l(s)\ v_l$ is compatible for all terminals $s : \kappa_1 \to \kappa_2$, and all compatible $v_l \in \mathcal{D}_l(\kappa_1)$.

(P1) is surjectivity of α. (P2) states that α is well-behaved wrt. $\sqcap$. (P3) says that the greatest element is mapped as expected. Note that (P1)-(P3) are only posed for the ground domain. One can prove that they generalize to function domains by the definition of function abstraction. (P4) is that the interpretations of terminals in $\mathcal{M}^C$ and $\mathcal{M}^A$ are suitably related. Finally (P5) is compatibility. (P4) and (P5) are generalized to terms in Lemma 9.

To prove $\alpha(\sigma_{\mathcal{M}_l}) = \sigma_{\mathcal{M}_r}$, we need that $rhs_{\mathcal{M}_r}$ is an exact abstract transformer of $rhs_{\mathcal{M}_l}$. The following lemma states this for all terms t, in particular those that occur in the equations. The generalization to product domains is immediate. Note that the result is limited to compatible valuations, but this will be sufficient for our purposes. The proof proceeds by induction on the structure of terms, while simultaneously proving $\mathcal{M}_l[\![t]\!]$ compatible with α. With this result, we obtain the required exact fixed-point transfer for precise abstractions.

▶ **Lemma 9.** *Assume* (P1)*,* (P4)*, and* (P5) *hold. For all terms t and all compatible ν, we have $\mathcal{M}_l[\![t]\!]\ \nu$ compatible and $\alpha(\mathcal{M}_l[\![t]\!]\ \nu) = \mathcal{M}_r[\![t]\!]\ \alpha(\nu)$.*

▶ **Theorem 10** (Exact Fixed-Point Transfer)**.** *Let G be a scheme with models $\mathcal{M}_l$ and $\mathcal{M}_r$. Let σ_l and σ_r be the corresponding semantics. If $\alpha : \mathcal{D}_l \to \mathcal{D}_r$ is precise, we have $\sigma_r = \alpha(\sigma_l)$.*

6 Domains for Higher-Order Games

We propose two domains, *abstract* and *optimized*, that allow us to solve HOG. The computation is a standard fixed-point iteration, and, in the optimized domain, this iteration has optimal complexity. Correctness follows by instantiating the previous framework.

Abstract Semantics. Our goal is to define an abstract model for games that (1) suitably relates to the concrete model from Section 4 and (2) is computable. By a suitable relation, we mean the two models should relate via an abstraction function. Provided the conditions on precision hold, correctness of the abstraction then follows from Theorem 10. Combined with Theorem 6, this will allow us to solve HOG. Computable in particular means the domain should be finite and the operations should be efficiently computable.

We define the $\mathcal{M}^A = (\mathcal{D}^A, \mathcal{I}^A)$ as follows. Again, we resolve the non-determinism into Boolean formulas. But rather than tracking the precise words generated by the scheme, we only track the current set of states of the automaton. To achieve the surjectivity required by precision, we restrict the powerset to those sets of states from which a word is accepted. Let $\mathsf{acc}(w) = \{q \mid q \xrightarrow{w} q_f \in Q_f\}$. For a language L we have $\mathsf{acc}(L) = \{\mathsf{acc}(w) \mid w \in L\}$. The abstract domain for terms of ground kind is $\mathcal{D}^A(o) = \mathsf{PBool}(\mathsf{acc}(T^*))$. The lifting to functions is as explained in Section 3. Satisfaction is now defined relative to a set Ω of elements of $\mathcal{P}(Q_{NFA})$ (cf. Section 4). With finitely many atomic propositions, there are only finitely many formulas (up to logical equivalence). This means we no longer need sets of formulas to represent infinite conjunctions, but can work with plain formulas. The ordering is thus the ordinary implication with the meet being conjunction and top being true.

The interpretation of ground terms is $\mathcal{I}^A(\$) = Q_f$ and $\mathcal{I}^A(a) = \mathsf{pre}_a$. Here pre_a is the predecessor computation under label a, $\mathsf{pre}_a(Q) = \{q' \in Q_{NFA} \mid q' \xrightarrow{a} q \in Q\}$. It is lifted to formulas by distributing it over conjunction and disjunction. The composition operators are again interpreted as conjunctions and disjunctions, depending on the owner of the non-terminal. Since we restrict the atomic propositions to $\mathsf{acc}(T^*)$, we have to show that the interpretations use only this restricted set. Proving $\mathcal{I}^A(s)$ is $\sqcap$-continuous is standard.

▶ **Lemma 11.** *The interpretations are defined on the abstract domain.*

▶ **Lemma 12.** *For all terminals s, $\mathcal{I}^A(s)$ is $\sqcap$-continuous over the respective lattices.*

Recall our concrete model is $\mathcal{M}^C = (\mathcal{D}^C, \mathcal{I}^C)$, where $\mathcal{D}^C = \mathcal{P}(\mathsf{PBool}(T^*))$. To relate this model to $\mathcal{M}^A$, we define the abstraction function $\alpha : \mathcal{D}^C(o) \to \mathcal{D}^A(o)$. It leaves the Boolean structure of a formula unchanged but maps every word (which is an atomic proposition) to the set of states from which this word is accepted. For a set of formulas, we take the conjunction of the abstraction of the elements. This conjunction is finite as we work over a finite domain, so there is no need to worry about infinite syntax. Technically, we define α on $\mathsf{PBool}(T^*)$ by $\alpha(\Phi) = \bigwedge_{\phi \in \Phi} \alpha(\phi)$ for a set of formulas $\Phi \in \mathcal{P}(\mathsf{PBool}(T^*))$, and

$$\alpha(\phi) = \begin{cases} \mathsf{acc}(w) & \text{if } \phi = w, \\ \alpha(\phi_1)\ op\ \alpha(\phi_2) & \text{if } \phi = \phi_1\ op\ \phi_2 \text{ and } op \in \{\wedge, \vee\}, \\ \phi & \text{if } \phi = \mathsf{true}\ . \end{cases}$$

This definition is suitable in that $\alpha(\sigma_{\mathcal{M}^C}) = \sigma_{\mathcal{M}^A}$ entails the following.

▶ **Theorem 13.** *$\sigma_{\mathcal{M}^A}(S)$ is satisfied by $\{Q \in \mathsf{acc}(T^*) \mid q_0 \in Q\}$ iff Player $\Diamond$ wins $\mathcal{G}$.*

To see that the theorem is a consequence of the exact fixed-point transfer, observe that $\{Q \in \mathsf{acc}(T^*) \mid q_0 \in Q\} = \mathsf{acc}(\mathcal{L}(A))$. Then, by $\sigma_{\mathcal{M}^A} = \alpha(\sigma_{\mathcal{M}^C})$ we have $\mathsf{acc}(\mathcal{L}(A))$ satisfies

$\sigma_{\mathcal{M}^A}(S)$ iff it also satisfies $\alpha(\sigma_{\mathcal{M}^C}(S))$. This holds iff $\mathcal{L}(A)$ satisfies $\sigma_{\mathcal{M}^C}(S)$ (a simple induction over formulas). By Theorem 6, this occurs iff Player $\Diamond$ wins the game.

It remains to establish $\alpha(\sigma_{\mathcal{M}^C}) = \sigma_{\mathcal{M}^A}$. With the framework, the exact fixed-point transfer follows from precision, Theorem 10. The proof of the following is routine.

▶ **Proposition 14.** *α is precise. Hence, $\alpha(\sigma_{\mathcal{M}^C}) = \sigma_{\mathcal{M}^A}$.*

Optimized Semantics. The above model yields a decision procedure for HOG via Kleene iteration. Unfortunately, the complexity is one exponential too high: The height of the domain for a symbol of order k in the abstract model is $(k+2)$-times exponential, where the height is the length of the longest strictly descending chain in the domain. This gives the maximum number of steps of Kleene iteration needed to reach the fixed point.

We present an optimized version of our model that is able to close the gap: In this model, the domain for an order-k symbol is only $(k+1)$-times exponentially high. The idea is to resolve the atomic propositions in $\mathcal{M}^A$, which are sets of states, into disjunctions among the states. The reader familiar with inclusion algorithms will find this decomposition surprising.

We first define $\alpha : \mathsf{PBool}(\mathsf{acc}(T^*)) \to \mathsf{PBool}(Q_{NFA})$. The optimized domain will then be based on the image of α. This guarantees surjectivity. For a set of states Q, we define $\alpha(Q) = \bigvee Q = \bigvee_{q \in Q} q$. For a formula, the abstraction function is defined to distribute over conjunction and disjunction. The optimized model is $\mathcal{M}^O = (\mathcal{D}^O, \mathcal{I}^O)$ with ground domain $\alpha(\mathsf{PBool}(\mathsf{acc}(T^*)))$. The interpretation is $\mathcal{I}^O(\$) = \bigvee Q_f$. For a, we resolve the set of predecessors into a disjunction, $\mathcal{I}^O(a)\ q = \bigvee \mathsf{pre}_a(\{q\})$. The function distributes over conjunction and disjunction. Finally, $\mathcal{I}^O(op_F)$ is conjunction or disjunction of formulas, depending on the owner of the non-terminal. Since we use a restricted domain, we have to argue that the operations do not leave the domain. It is also straightforward to prove our interpretation is $\sqcap$-continuous as required.

▶ **Lemma 15.** *The interpretations are defined on the optimized domain.*

▶ **Lemma 16.** *For all terminals s, $\mathcal{I}^O(s)$ is $\sqcap$-continuous over the respective lattices.*

We again show precision, enabling the required exact fixed-point transfer.

▶ **Proposition 17.** *α is precise. Hence, $\alpha(\sigma_{\mathcal{M}^A}) = \sigma_{\mathcal{M}^O}$.*

▶ **Theorem 18.** *$\sigma_{\mathcal{M}^O}(S)$ is satisfied by $\{q_0\}$ iff Player $\Diamond$ wins $\mathcal{G}$.*

It is sufficient to show $\sigma_{\mathcal{M}^A}(S)$ is satisfied by $\{Q \in \mathsf{acc}(T^*) \mid q_0 \in Q\}$ iff $\sigma_{\mathcal{M}^O}(S)$ is satisfied by $\{q_0\}$. Theorem 13 then yields the statement. Propositions Q in $\sigma_{\mathcal{M}^A}(S)$ are resolved into disjunctions $\bigvee Q$ in $\sigma_{\mathcal{M}^O}(S)$. For such a proposition, we have $Q \in \{Q \in \mathsf{acc}(T^*) \mid q_0 \in Q\}$ iff $\bigvee Q$ is satisfied by $\{q_0\}$. This equivalence propagates to the formulas $\sigma_{\mathcal{M}^A}(S)$ and $\sigma_{\mathcal{M}^O}(S)$ as the Boolean structure coincides. The latter follows from $\alpha(\sigma_{\mathcal{M}^A}(S)) = \sigma_{\mathcal{M}^O}(S)$.

Complexity. To solve HOG, we compute the semantics $\sigma_{\mathcal{M}^O}$ and then evaluate $\sigma_{\mathcal{M}^O}(S)$ at the assignment $\{q_0\}$. For the complexity, assume that the highest order of any non-terminal in $\mathcal{G}$ is k. We show the number of iterations needed to compute the greatest fixed point is at most $(k+1)$-times exponential. We do this via a suitable upper bound on the length of strictly descending chains in the domains assigned by $\mathcal{D}^O$.

▶ **Proposition 19.** *The semantics $\sigma_{\mathcal{M}^O}$ can be computed in $(k+1)\mathsf{EXP}$, where k is the highest order of any non-terminal in the input scheme.*

The lower bound is via a reduction from the word membership problem for alternating k-iterated pushdown automata with polynomially-bounded auxiliary work-tape. This problem was shown by Engelfriet to be $(k+1)$EXP-hard. We can reduce this problem to HOG via well-known translations between iterated stack automata and recursion schemes, using the regular language specifying the winning condition to help simulate the work-tape.

▶ **Proposition 20.** *Determining whether Player $\Diamond$ wins $\mathcal{G}$ is $(k+1)$EXP-hard for $k > 0$.*

Together, these results show the following corollary and final result.

▶ **Corollary 21.** HOG *is $(k+1)$EXP-complete for order-k schemes and $k > 0$.*

References

1. P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Simulation subsumption in Ramsey-based Büchi automata universality and inclusion testing. In *CAV*, volume 6174 of *LNCS*, pages 132–147. Springer, 2010.
2. P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Advanced Ramsey-based Büchi automata inclusion testing. In *CONCUR*, volume 6901 of *LNCS*, pages 187–202. Springer, 2011.
3. S. Abramsky. Abstract interpretation, logical relations and Kan extensions. *J. Log. Comp.*, 1(1):5–40, 1990.
4. S. Abramsky and C. Hankin. An introduction to abstract interpretation. In *Abstract Interpretation of declarative languages*, volume 1, pages 63–102. Ellis Horwood, 1987.
5. K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *LMCS*, 3(3):1–23, 2007.
6. K. Backhouse and R. C. Backhouse. Safety of abstract interpretations for free, via logical relations and Galois connections. *Sci. Comp. Prog.*, 51(1-2):153–196, 2004.
7. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.
8. A. Bouajjani and A. Meyer. Symbolic reachability analysis of higher-order context-free processes. In *FSTTCS*, volume 3328 of *LNCS*, pages 135–147. Springer, 2004.
9. C. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *ICALP*, volume 7392 of *LNCS*, pages 165–176. Springer, 2012.
10. C. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *CSL*, volume 23 of *LIPIcs*, pages 129–148. Dagstuhl, 2013.
11. G. L. Burn, C. Hankin, and S. Abramsky. Strictness analysis for higher-order functions. *Sci. Comp. Prog.*, 7(3):249–278, 1986.
12. T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP*, volume 2380 of *LNCS*, pages 704–715. Springer, 2002.
13. T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *ICALP*, volume 2719 of *LNCS*, pages 556–569. Springer, 2003.
14. D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS*, volume 2420 of *LNCS*, pages 165–176. Springer, 2002.
15. P. Cousot and R. Cousot. Higher order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection, and PER analysis. In *ICCL*, pages 95–112. IEEE, 1994.
16. W. Damm. The IO- and OI-hierarchies. *Theor. Comp. Sci.*, 20:95–207, 1982.
17. W. Damm and A. Goerdt. An automata-theoretical characterization of the OI-hierarchy. *Inf. Comp.*, 71:1–32, 1986.

18 A. Farzan, Z. Kincaid, and A. Podelski. Proof spaces for unbounded parallelism. In *POPL*, pages 407–420. ACM, 2015.

19 A. Farzan, Z. Kincaid, and A. Podelski. Proving liveness of parameterized programs. In *LICS*, pages 185–196. IEEE, 2016.

20 Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Proofs that count. In *POPL*, pages 151–164. ACM, 2014.

21 A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. *ENTCS*, 9:27–37, 1997.

22 S. Fogarty and M. Y. Vardi. Efficient Büchi universality checking. In *TACAS*, volume 6015 of *LNCS*, pages 205–220. Springer, 2010.

23 C. Grellois. *Semantics of linear logic and higher-order model-checking*. PhD thesis, Université Paris Diderot (Paris 7), 2016.

24 C. Grellois and P.-A. Melliès. Finitary semantics of linear logic and higher-order model-checking. In *MFCS*, volume 9234 of *LNCS*, pages 256–268. Springer, 2015.

25 C. Grellois and P.-A. Melliès. An infinitary model of linear logic. In *FoSSaCS*, volume 9034 of *LNCS*, pages 41–55. Springer, 2015.

26 C. Grellois and P.-A. Melliès. Relational semantics of linear logic and higher-order model checking. In *CSL*, volume 41 of *LIPIcs*, pages 260–276. Dagstuhl, 2015.

27 A. Haddad. IO vs OI in higher-order recursion schemes. In *FICS*, volume 77 of *EPTCS*, pages 23–30, 2012.

28 M. Hague, R. Meyer, and S. Muskalla. Domains for higher-order games. *CoRR*, abs/1705.00355, 2017. URL: `http://arxiv.org/abs/1705.00355`.

29 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461. IEEE, 2008.

30 M. Hague and C.-H. L. Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. In *FoSSaCS*, volume 4423 of *LNCS*, pages 213–227. Springer, 2007.

31 M. Hague and C.-H. L. Ong. Winning regions of pushdown parity games: A saturation method. In *CONCUR*, volume 5710 of *LNCS*, pages 384–398. Springer, 2009.

32 M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.

33 M. Hofmann and W. Chen. Abstract interpretation from Büchi automata. In *CSL-LICS*, pages 51:1–51:10, 2014.

34 M. Hofmann and J. Ledent. A cartesian-closed category for higher-order model checking. In *LICS*. IEEE, 2017. To appear.

35 L. Holík, R. Meyer, and S. Muskalla. Summaries for context-free games. In *FSTTCS*, volume 65 of *LIPIcs*, pages 41:1–41:16. Dagstuhl, 2016.

36 Lukás Holík and Roland Meyer. Antichains for the verification of recursive programs. In *NETYS*, volume 9466 of *LNCS*, pages 322–336. Springer, 2015.

37 T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, volume 2303 of *LNCS*, pages 205–222. Springer, 2002.

38 T. Knapik, D. Niwiński, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *ICALP*, volume 3580 of *LNCS*, pages 1450–1461. Springer, 2005.

39 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/`.

40 N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pages 416–428. ACM, 2009.

41 N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *LICS*, pages 179–188. IEEE, 2009.

42 Z. Long, G. Calin, R. Majumdar, and R. Meyer. Language-theoretic abstraction refinement. In *FASE*, volume 7212 of *LNCS*, pages 362–376. Springer, 2012.

43 D. A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975. URL: `http://www.jstor.org/stable/1971035`.

44 R. Meyer, S. Muskalla, and E. Neumann. Liveness verification and synthesis: New algorithms for recursive programs. `https://arxiv.org/abs/1701.02947`.

45 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90. IEEE, 2006.

46 S. J. Ramsay. *Intersection-Types and Higher-Order Model Checking*. PhD thesis, Oxford University, 2013.

47 S. J. Ramsay. Exact intersection type abstractions for safety checking of recursion schemes. In *PPDP*, pages 175–186. ACM, 2014.

48 S. Salvati. Recognizability in the simply typed lambda-calculus. In *WoLLIC*, volume 5514 of *LNCS*, pages 48–60. Springer, 2009.

49 S. Salvati and I. Walukiewicz. A model for behavioural properties of higher-order programs. In *CSL*, volume 41 of *LIPIcs*, pages 229–243. Dagstuhl, 2015.

50 S. Salvati and I. Walukiewicz. Typing weak MSOL properties. In *FoSSaCS*, volume 9034 of *LNCS*, pages 343–357. Springer, 2015.

51 S. Salvati and I. Walukiewicz. Using models to model-check recursive schemes. *LMCS*, 11(2):1–23, 2015.

52 I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comp.*, 164(2):234–263, 2001.

53 M. Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV*, volume 4144 of *LNCS*, pages 17–30. Springer, 2006.

Fine-Grained Complexity of Rainbow Coloring and Its Variants*

Akanksha Agrawal

University of Bergen, Norway
akanksha.agrawal@uib.no

Abstract

Consider a graph G and an edge-coloring $c_R : E(G) \to [k]$. A rainbow path between $u, v \in V(G)$ is a path P from u to v such that for all $e, e' \in E(P)$, where $e \neq e'$ we have $c_R(e) \neq c_R(e')$. In the Rainbow k-Coloring problem we are given a graph G, and the objective is to decide if there exists $c_R : E(G) \to [k]$ such that for all $u, v \in V(G)$ there is a rainbow path between u and v in G. Several variants of Rainbow k-Coloring have been studied, two of which are defined as follows. The Subset Rainbow k-Coloring takes as an input a graph G and a set $S \subseteq V(G) \times V(G)$, and the objective is to decide if there exists $c_R : E(G) \to [k]$ such that for all $(u, v) \in S$ there is a rainbow path between u and v in G. The problem Steiner Rainbow k-Coloring takes as an input a graph G and a set $S \subseteq V(G)$, and the objective is to decide if there exists $c_R : E(G) \to [k]$ such that for all $u, v \in S$ there is a rainbow path between u and v in G. In an attempt to resolve open problems posed by Kowalik et al. (ESA 2016), we obtain the following results.

- For every $k \geq 3$, Rainbow k-Coloring does not admit an algorithm running in time $2^{o(|E(G)|)}n^{\mathcal{O}(1)}$, unless ETH fails.
- For every $k \geq 3$, Steiner Rainbow k-Coloring does not admit an algorithm running in time $2^{o(|S|^2)}n^{\mathcal{O}(1)}$, unless ETH fails.
- Subset Rainbow k-Coloring admits an algorithm running in time $2^{\mathcal{O}(|S|)}n^{\mathcal{O}(1)}$. This also implies an algorithm running in time $2^{o(|S|^2)}n^{\mathcal{O}(1)}$ for Steiner Rainbow k-Coloring, which matches the lower bound we obtain.

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

Keywords and phrases Rainbow Coloring, Lower bound, ETH, Fine-grained Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.60

1 Introduction

Graph connectivity is one of the fundamental properties in graph theory. Several connectivity measures like k-vertex connectivity, k-edge connectivity, hamiltonicity, etc. have been studied for graphs. Chartrand et al. [8] defined an interesting connectivity measure, called rainbow connectivity, which is defined as follows. Let G be a graph and $c_R : E(G) \to [k]$ be an edge-coloring of G. A *rainbow path* between $u, v \in V(G)$ is a path P from u to v such that for all $e, e' \in E(P)$, where $e \neq e'$ we have $c_R(e) \neq c_R(e')$. A graph with an edge-coloring is *rainbow-connected* if for every pair of vertices there is a rainbow path between them. In the Rainbow k-Coloring problem we are given a graph G, and the objective is to decide if there exists an edge-coloring $c_R : E(G) \to [k]$ such that for all $u, v \in V(G)$, there is a rainbow

* Due to space limitations most proofs have been omitted.
The research leading to these results received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ ERC Grant Agreements no. 306992.

path between u and v in G. The problem has received attention both from graph theoretic and algorithmic point of view, the details of which can be found, for instance in [9, 23, 24].

Rainbow k-Coloring problem is notoriously hard. It was conjectured by Caro et al. [4] to be NP-complete already for $k = 2$. Indeed, by giving a polynomial time reduction from 3-SAT, this was confirmed by Chakraborty et al. [5]. Building on their results, Ananth et al. [3] later showed that Rainbow k-Coloring remains NP-complete for every $k \geq 2$. An alternate hardness proof was also given by Le and Tuza [21]. For the complexity of the problem on restricted graph classes, see e.g., [5, 6, 7, 8].

Impagliazzo et al. [16] introduced the Exponential time hypothesis (ETH), which has been used as a basis for proving qualitative lower bounds for computational problems. The ETH states that 3-SAT does not admit an algorithm running in time $2^{o(n)}n^{\mathcal{O}(1)}$, where n is the number of variables in the input 3-CNF formula. It has been shown that assuming ETH, several NP-hard problems like Independent Set, Hitting set, and Chromatic Number do not admit subexponential time algorithms (see the survey [25]).

Kowalik et al. [20] studied the fine-grained complexity of Rainbow k-Coloring and some of its variants. In particular, they showed that Rainbow k-Coloring admits neither an algorithm running in time $2^{o(|V(G)|^{3/2})}|V(G)|^{\mathcal{O}(1)}$, nor an algorithm running in time $2^{o(|E(G)|/\log|E(G)|)}|V(G)|^{\mathcal{O}(1)}$, unless ETH fails. They also studied a variant of Rainbow k-Coloring, called Subset Rainbow k-Coloring (to be defined shortly), which was introduced by Chakraborty et al. [5]. They showed that Subset Rainbow k-Coloring does not admit an algorithm running in time $2^{o(|E(G)|)}|V(G)|^{\mathcal{O}(1)}$ assuming ETH. In contrast, they designed an FPT algorithm for the problem running in time $|S|^{\mathcal{O}(|S|)}n^{\mathcal{O}(1)}$, where S is a part of the input. For $k = 2$, they obtained a faster algorithm running in time $2^{\mathcal{O}(|S|)}n^{\mathcal{O}(1)}$. Finally, they proposed yet another (parametric) variant of Rainbow k-Coloring, which they called Steiner Rainbow k-Coloring. Their lower bound result for Rainbow k-Coloring implies that Steiner Rainbow k-Coloring does not admit an algorithm running in time $2^{o|S|^{3/2}}n^{\mathcal{O}(1)}$. Moreover, their algorithm for Subset Rainbow k-Coloring gives an algorithm for Steiner Rainbow k-Coloring running in time $2^{\mathcal{O}(|S|^2\log|S|)}n^{\mathcal{O}(1)}$.

Our results. We attempt to tighten the gaps in the study of fine-grained complexity of Rainbow k-Coloring and some of its variants, initiated by Kowalik et al. [20]. We now describe our results in detail.

The first problem that we study is Steiner Rainbow k-Coloring, which is formally defined below.

Steiner Rainbow k-Coloring	**Parameter:** $\|S\|$
Input: A graph G and a vertex subset $S \subseteq V(G)$.	
Question: Does there exist an edge-coloring $c_R : E(G) \to [k]$ such that for every $u, v \in S$, there is a rainbow path between u and v in G?	

In Section 3, we show that for every $k \geq 3$, Steiner Rainbow k-Coloring does not admit an algorithm running in time $2^{o(|S|^2)}n^{\mathcal{O}(1)}$, under ETH. This resolves an open problem posed by Kowalik et al. [20]. To prove the result, we give a reduction from k-Coloring on graphs of maximum degree $2(k-1)$ which does not admit an algorithm running in time $2^{o(n)}n^{\mathcal{O}(1)}$, assuming ETH. Our reduction starts by computing a harmonious coloring of the (bounded degree) input instance of k-Coloring, which forms an essential step in the construction of S for the instance of Steiner Rainbow k-Coloring that we create. The idea of using harmonious coloring for proving lower bounds of the form $2^{o(\ell^2)}n^{\mathcal{O}(1)}$ was used by Agrawal et al. [1] to prove a lower bound for Split Contraction, when parameterized by the vertex cover number ℓ, of the input graph. Also, the idea of partitioning vertices

of the input graph based on some coloring scheme was used by Cygan et al. [10] to prove ETH-based lower bounds for Graph Homomorphism and Subgraph Isomorphism.

The next problem we study is Rainbow k-Coloring, which is formally defined below.

Rainbow k-Coloring
Input: A graph G.
Question: Does there exist an edge-coloring $c_R : E(G) \to [k]$ such that for every $u, v \in V(G)$, there is a rainbow path between u and v in G?

Kowalik et al. [20] conjectured that for every $k \geq 2$, Rainbow k-Coloring does not admit an algorithm running in time $2^{o(|E(G)|)}n^{\mathcal{O}(1)}$, unless ETH fails. In Section 4, we resolve this conjecture for every $k \geq 3$. Again, we proceed with a reduction from k-Coloring on bounded degree graphs. Although, the general scheme of reduction is the same as the one we five for Steiner Rainbow k-Coloring, in this case the reduction is more involved. Furthermore, we require to distinguish between the cases for k being odd and even in the gadget construction. Also, to keep our gadgets simpler, we separate the case for $k = 3$ and $k > 3$.

Finally, we study the complexity of Subset Rainbow k-Coloring, which is formally defined below.

Subset Rainbow k-Coloring **Parameter:** $|S|$
Input: A graph G and a subset $S \subseteq V(G) \times V(G)$.
Output: An edge-coloring $c_R : E(G) \to [k]$ such that for every $(u, v) \in S$, there is a rainbow path between u and v in G, if it exists. Otherwise, return *no*.

In Section 5 we design an FPT algorithm running in time $2^{\mathcal{O}(|S|)}n^{\mathcal{O}(1)}$ for Subset Rainbow k-Coloring, for every fixed k. This resolves the conjecture of Kowalik et al. [20] regarding the existence of an algorithm running in time $2^{\mathcal{O}(|S|)}n^{\mathcal{O}(1)}$ for Subset Rainbow k-Coloring, and is an improvement over their algorithm, which runs in time $|S|^{\mathcal{O}(|S|)}n^{\mathcal{O}(1)}$, for $k \geq 3$. Our algorithm is based on the technique of color coding, which was introduced by Alon et al. [2]. Observe that Steiner Rainbow k-Coloring is a special case of Subset Rainbow k-Coloring. Hence, as a corollary we obtain an algorithm running in time $2^{\mathcal{O}(|S|^2)}n^{\mathcal{O}(1)}$ for Steiner Rainbow k-Coloring, which matches the lower bound we prove in Section 3.

2 Preliminaries

In this section, we state some basic definitions and introduce terminology from graph theory and algorithms. We also establish some of the notation that will be used throughout.

We denote the set of natural numbers by $\mathbb{N}$. For $k \in \mathbb{N}$, by $[k]$ we denote the set $\{1, 2, \ldots, k\}$. We use standard terminology from the book of Diestel [13] for the graph related terminologies which are not explicitly defined here. We consider finite simple graphs. For a graph G, by $V(G)$ and $E(G)$ we denote the vertex and edge sets of the graph G, respectively. For $v \in V(G)$, by $N_G(v)$ we denote the set $\{u \in V(G) \mid (v, u) \in E(G)\}$. We drop the subscript G from $N_G(v)$ when the context is clear. For $C, C' \subseteq V(G)$, we say that there is an edge between C and C' in G if there exists $u \in C$ and $v \in C'$ such that $(u, v) \in E(G)$. A *path* $P = (v_1, v_2, \ldots, v_\ell)$ is a graph with vertex and edge sets as $\{v_1, v_2, \ldots, v_\ell\}$ and $\{(v_i, v_{i+1}) \mid i \in [l-1]\}$, respectively.

A *harmonious coloring* of a graph G is a vertex coloring $\varphi : V(G) \to [k]$, with color classes $C_1, C_2, \ldots, C_k$ such that for each $i \in [k]$, C_i is an independent set in G and for all $i, j \in [k]$, where $i \neq j$ there is at most one edge between C_i and C_j in G. We use the following result

for computing a harmonious coloring on bounded degree graphs.

▶ **Proposition 1** ([11, 14, 22, 26]). *Given a G with the degree of each vertex bounded by d, where d is a fixed constant. A harmonious coloring of G can be computed in time $\mathcal{O}(n^{\mathcal{O}(1)})$ using $\mathcal{O}(\sqrt{n})$ colors with each color class having at most $\mathcal{O}(\sqrt{n})$ vertices.*

3 Lower bound for Steiner Rainbow k-Coloring

In this section, we show that for every $k \geq 3$, Steiner Rainbow k-Coloring does not admit an algorithm running in time $2^{o(|S|^2)}n^{\mathcal{O}(1)}$, unless ETH fails. Towards this we give an appropriate reduction from k-Coloring on graphs of maximum degree $2(k-1)$. We note that k-Coloring does not admit an algorithm running in time $2^{o(n)}n^{\mathcal{O}(1)}$ unless ETH fails [17]. Moreover, assuming ETH, 3-Coloring does not admit an algorithm running in time $2^{o(n)}n^{\mathcal{O}(1)}$ on graphs of maximum degree 4 [18, 11]. This follows from the fact that 3-Coloring does not admit such an algorithm, and a reduction from an instance G of 3-Coloring to an equivalent instance G' of 3-Coloring, where G' is a graph with maximum degree 4 with $|V(G')| \in \mathcal{O}(|V(G)|)$ (see [15, Theorem 4.1]). In fact, we can show that k-Coloring does not admit an algorithm running in time $2^{o(n)}n^{\mathcal{O}(1)}$ on graphs of maximum degree $2(k-1)$ (folklore). This result can be obtained (inductively) by giving a reduction from an instance G of $(k-1)$-Coloring on graphs of degree at most $2(k-2)$ to an instance of k-Coloring on a graphs of bounded average degree (by adding global vertex), and then using an approach similar to that in Theorem 4.1 in [15] we can obtain an (equivalent) instance of k-Coloring where the degree of the graph is bounded by $2(k-1)$.

Given an instance G of k-Coloring on n vertices and degree bounded by $2(k-1)$, we start by computing a harmonious coloring φ of G with $t \in \mathcal{O}(\sqrt{n})$ color classes such that each color class contains at most $\mathcal{O}(\sqrt{n})$ vertices using Proposition 1. Let $C_1, C_2, \ldots, C_t$ be the color classes of φ. Recall that for $i, j \in [t]$ with $i \neq j$ there is at most one edge between C_i and C_j in G. Moreover, C_i is an independent set in G, where $i \in [t]$. We create an instance G' of k-Coloring which has a harmonious coloring φ' with color classes $C'_1, C'_2, \ldots, C'_t$ such that for all $i, j \in [t]$, $i \neq j$ we have exactly one edge between C_i and C_j. Initially, we have $G = G'$ and $C'_i = C_i$, for all $i \in [t]$. For each $i, j \in [t]$, $i \neq j$ such that there is no edge between C_i and C_j in G we add two new vertices a_{ij} and a_{ji} to $V(G')$ and add the edge (a_{ij}, a_{ji}) to $E(G')$. Furthermore, we add a_{ij} to C'_i and a_{ji} to C'_{ji}. Observe that $|V(G')| \in \mathcal{O}(n)$, $|E(G')| \in \mathcal{O}(n)$, and for each $i \in [t]$, $|C'_i| \in \mathcal{O}(\sqrt{n})$. Also, for each $i, j \in [t]$, $i \neq j$ there is exactly one edge between C'_i and C'_j in G'. It is easy to see that G is a *yes* instance of k-Coloring if and only if G' is a *yes* instance of k-Coloring.

Hereafter, we will be working with the instance G' of k-Coloring, together with its harmonious coloring φ' with color classes $C'_1, C'_2, \ldots, C'_t$. Moreover, for $i, j \in [t]$, $i \neq j$ there is exactly one edge between C'_i and C'_j in G'.

We now move to the description of creating an equivalent instance $(\tilde{G}, S)$ of Steiner Rainbow k-Coloring, where $k \geq 3$. Initially, we have $V(\tilde{G}) = V(G')$. For $(u, v) \in E(G')$ we add $k-3$ new vertices $x^{uv}_1, x^{uv}_2, \ldots, x^{uv}_{k-3}$ to $\tilde{G}$ and add all the edges in the path $(u, x^{uv}_1, \ldots, x^{uv}_{k-3}, v)$ to $E(\tilde{G})$. Note that for $k = 3$ we do not any new vertex and directly add the edge (u, v) to $\tilde{G}$. For each $i \in [t]$ we add a vertex c_i to $\tilde{G}$ and add all the edges in $\{(c_i, v) \mid v \in C'_i\}$ to $E(\tilde{G})$. Finally, we set $S = \{c_i \mid i \in [t]\}$. Notice that $|S| \in \mathcal{O}(\sqrt{n})$. In the following lemma we establish that G' is a *yes* instance of k-Coloring if and only if $(\tilde{G}, S)$ is a *yes* instance of Steiner Rainbow k-Coloring.

▶ **Lemma 2.** *G' is a* yes *instance of k-Coloring if and only if $(\tilde{G}, S)$ is a* yes *instance of* Steiner Rainbow k-Coloring.

Proof. In the forward direction, let G' be a *yes* instance of k-Coloring, and $c : V(G') \to [k]$ be one of its solution. We create a coloring $c_R : E(\tilde{G}) \to [k]$ as follows. For $i \in [t]$ and $v \in C'_i$ we set $c_R(c_i, v) = c(v)$. For $i, j \in [t]$, $i \neq j$ let u, v be the (unique) vertices in C'_i and C'_j such that $(u, v) \in E(G')$. We now describe the value of c_R for edges in the path $P = (u, x_1^{uv}, \ldots, x_{k-3}^{uv}, v)$. Notice that $|E(P)| = k - 2$, and we arbitrarily assign distinct integers in $[k] \setminus \{c_R(c_i, u), c_R(c_j, v)\}$ to $c_R(e)$, where $e \in E(P)$. Since c is a proper coloring of G', $c_R(c_i, u) = c(u) \neq c(v) = c_R(c_j, v)$. This together with the definition of c_R for edges in P implies that there is a rainbow path, namely $(c_i, u, x_1^{uv}, \ldots, x_{k-3}^{uv}, v, c_j)$ in $\tilde{G}$ between c_i and c_j. This concludes the proof in the forward direction.

In the reverse direction, let $(\tilde{G}, S)$ be a *yes* instance of Steiner Rainbow k-Coloring, and $c_R : E(\tilde{G}) \to [k]$ be one of its solution. We create a coloring $c : V(G') \to [k]$ as follows. For $i \in [t]$ and $v \in C'_i$, we let $c(v) = c_R(c_i, v)$. We show that c is a solution to k-Coloring in G'. Consider $(u, v) \in E(G')$, and let $u \in C'_i$ and $v \in C'_j$. Note that we have $i \neq j$. Let P be a rainbow path between c_i and c_j in $\tilde{G}$. By the construction of $\tilde{G}$, we have $N_{\tilde{G}}[c_i] \cap N_{\tilde{G}}[c_j] = \emptyset$. Moreover, since P is a rainbow path, it can contain at most k edges. Recall that $N_{\tilde{G}}(c_i) = C'_i$, $N_{\tilde{G}}(c_j) = C'_j$, and there is exactly one path with at most $k - 2$ edges between a vertex in C'_i and a vertex in C'_j, namely $(c_i, u, x_1^{uv}, \ldots, x_{k-3}^{uv}, v, c_j)$. This together with the construction of c implies that $c(u) \neq c(v)$. This concludes the proof. ◀

▶ **Theorem 3.** Steiner Rainbow k-Coloring *does not admit an algorithm running in time* $2^{o(|S|^2)}n^{\mathcal{O}(1)}$*, unless* ETH *fails. Here,* n *is the number of vertices in the input graph.*

4 Lower bound for Rainbow k-Coloring

In this section, we show that for every $k \geq 3$, Rainbow k-Coloring does not admit an algorithm running in time $2^{o(|E(G)|)}n^{\mathcal{O}(1)}$, unless ETH fails. We give different reductions for the case when $k = 3$ (Section 4.1), k is an even number greater than 3 (Section 4.2), and k is an odd number greater than 4 (Section 4.3). We note that although the approach used for the proving lower bound for Rainbow 3-Coloring is extensible to Rainbow k-Coloring when k is odd, it unnecessarily adds to complexity of the reduction. Moreover, the approach we follow for showing the lower bound result for $k > 3$, where k is an odd number, introduces some technical issues when we try to extend it for $k = 3$.

Towards proving our lower bound result, we give an appropriate reduction from k-Coloring on graphs of maximum degree $2(k - 1)$, which does not admit an algorithm running in time $2^{o(n)}n^{\mathcal{O}(1)}$ unless ETH fails. The key idea behind the reduction is the same as that presented in Section 3, but for this case it is more involved. Before moving on to the description of the reductions we define a graph that will be useful in our reductions.

A *clique sequence* $\mathbb{Z}_{n,t} = (Z_1, Z_2, \ldots Z_t)$ of order (n, t) is a graph defined as follows. We have $V(\mathbb{Z}_{k,t}) = \uplus_{i \in [t]} Z_i$, where $|Z_i| = n$ for all $i \in [t]$. For each $i \in [t]$, all the edges in $\{(z, z') \mid z, z' \in Z_i\}$ are present in $E(\mathbb{Z}_{n,t})$, *i.e.* Z_i is a clique. Furthermore, for all $i \in [t - 1]$ all the edges in $\{(z, z') \mid z, \in Z_i, x' \in Z_{i+1}\}$ are present in $E(\mathbb{Z}_{n,t})$.

4.1 Lower bound for Rainbow 3-Coloring

In this section, we show that Rainbow 3-Coloring does not admit an algorithm running in time $2^{o(|E(G)|)}n^{\mathcal{O}(1)}$, where n is the number of vertices in the input graph G.

Let G be an instance of 3-Coloring on n vertices with maximum degree bounded by 4. We start by computing (in polynomial time) a harmonious coloring φ of G with $t \in \mathcal{O}(\sqrt{n})$ color classes such that each color class contains at most $\mathcal{O}(\sqrt{n})$ vertices using Proposition 1.

Let $C_1, C_2, \ldots, C_t$ be the color classes of φ. From the discussion in Section 3, we assume that for $i, j \in [t]$, $i \neq j$ there is exactly one edge between C_i and C_j in G. We construct an instance G' of Rainbow 3-Coloring as follows.

- *Color class gadget.* Consider $i \in [t]$. The color class gadget $\mathcal{C}_i$ comprises of the set C_i, two vertices c_i, b_i, and a clique U_i on 3 vertices with vertex set $\{u_1^i, u_2^i, u_3^i\}$. We add all the edges in $\{(v, c_i), (v, b_i), (v, u_1^i), (v, u_2^i), (v, u_3^i) \mid v \in C_i\}$ to $E(\mathcal{C}_i)$. Also, we add the edge (b_i, c_i) to $E(\mathcal{C}_i)$.
- *Connection between color class gadgets.* Consider $i, j \in [t]$ such that $i \neq j$. We add all the edges in $\{(b_i, u_\ell^j) \mid \ell \in [3]\}$ to $E(G')$. Furthermore, we add all the edges $\{(u_\ell^i, u_{\ell'}^j) \mid \ell, \ell' \in [3]\}$ to $E(G')$. Note that $\{u_\ell^{i'} \mid i' \in [t], \ell \in [3]\}$ induces a clique in G'.
- *Encoding edges.* For $i, j \in [t]$, $i \neq j$ we add the unique edge (u, v) between C_i and C_j with $u \in C_i$ and $v \in C_j$ to G'. Note that this is same as adding all the edges in $E(G)$ to $E(G')$.

This finishes the description of the instance G' of Rainbow 3-Coloring. We note that some of the edges in G' are not necessary for the correctness of the reduction. However, they are added to reduce the number of pairs for which we need to argue about the existence of a rainbow path. Before moving on to the proof of equivalence between these instances, we create an edge-coloring $c_R : E(G') \to [3]$. Here, we create c_R based on a solution c to 3-Coloring in G, assuming that G is a *yes* instance of 3-Coloring. We will follow computation modulo k, and therefore color 0 is same as color k.

▶ **Definition 4.** Given a solution c to 3-Coloring in G, we construct $c_R : E(G') \to [3]$ as follows.

1. For $i \in [t]$, and $v \in C_i$ set $c_R(v, c_i) = c(v)$, $c_R(v, b_i) = c(v)$, and for $\ell \in [3]$, $c_R(v, u_\ell^i) = \ell$.
2. For $i, j \in [t]$, $i \neq j$ let (u, v) be the unique edge between C_i and C_j. We set $c_R(u, v)$ to be the unique integer in $[3] \setminus \{c(u), c(v)\}$. Here, the uniqueness is guaranteed by the fact that c is a proper 3-coloring of G, promising that $c(u) \neq c(v)$.
3. For $i \in [t]$ set $c_R(b_i, c_i) = 3$, $c_R(u_1^i, u_2^i) = 3$, $c_R(u_2^i, u_3^i) = 2$, and $c_R(u_3^i, u_1^i) = 1$.
4. For $i, j \in [t]$, $i \neq j$ and $\ell \in [3]$ set $c_R(b_i, u_\ell^j) = \ell - 1$.
5. For $i, j \in [t]$, $i \neq j$ and $\ell \in [3]$ set $c_R(u_\ell^i, u_\ell^j) = \ell$. Furthermore, for $\ell' \in [3] \setminus \{\ell\}$ we set $c_R(u_\ell^i, u_{\ell'}^j) = \hat{\ell}$, where $\hat{\ell}$ is the unique integer in $[3] \setminus \{\ell, \ell'\}$.

Next, we prove some lemmata that will be useful in establishing the equivalence between the instance G of 3-Coloring and the instance G' of Rainbow 3-Coloring.

▶ **Lemma 5.** *For $i, j \in [t]$, where $i \neq j$, let (u^*, v^*) be the unique edge between C_i and C_j with $u^* \in C_i$ and $v^* \in C_j$. There is exactly one path, namely (c_i, u^*, v^*, c_j) in G', between c_i and c_j that has at most 3 edges.*

▶ **Lemma 6.** *Let G be a yes instance of 3-Coloring, and c be one of its solution. Furthermore, let $c_R : E(G') \to [3]$ be the coloring given by Definition 4 for the coloring c of G. Then for all $i \in [t]$, and $u, v \in \mathcal{C}_i$ there is a rainbow path between u and v in G'.*

▶ **Lemma 7.** *Let G be a yes instance of 3-Coloring, and c be one of its solution. Furthermore, let $c_R : E(G') \to [3]$ be the coloring given by Definition 4 for the coloring c of G. Then for all $i, j \in [t]$, $i \neq j$ for all $u \in \mathcal{C}_i$ and $v \in \mathcal{C}_j$ there is a rainbow path between u and v in G'.*

We now establish equivalence between the instance G of 3-Coloring and the instance G' of Rainbow 3-Coloring.

▶ **Lemma 8.** *G is a yes instance of* 3-Coloring *if and only if G' is a yes instance of* Rainbow 3-Coloring.

Proof. In the forward direction, let G be a *yes* instance of 3-Coloring, and $c : V(G) \to [3]$ be one of its solution. Let $c_R : E(G') \to [3]$ be the coloring given by Definition 4 for the given coloring c of G. From Lemma 6 and 7 it follows that c_R is a solution to Rainbow 3-Coloring in G'.

In the reverse direction, let G' be a *yes* instance of Rainbow 3-Coloring, and $c_R : E(G') \to [3]$ be one of its solution. We create a coloring $c : V(G) \to [3]$ as follows. For $i \in [t]$ and $v \in C_i$, we let $c(v) = c_R(c_i, v)$. We show that c is a valid solution to 3-Coloring in G. Consider $(u, v) \in E(G)$, and let $u \in C_i$ and $v \in C_j$. Note that we have $i \neq j$. Let P be a rainbow path between c_i and c_j in G'. Note that P can have at most 3 edges. By Lemma 5 we know that $P = (c_i, u, v, c_j)$, therefore by construction of c, we have $c_R(c_i, u) = c(u) \neq c(v) = c_R(c_i, v)$. This concludes the proof. ◀

▶ **Theorem 9.** Rainbow 3-Coloring *does not admit an algorithm running in time $2^{o(|E(G)|)}n^{\mathcal{O}(1)}$, unless* ETH *fails. Here, n is the number of vertices in the input graph.*

4.2 Lower Bound for Rainbow k-Coloring, $k > 3$ and even

In this section, we show that Rainbow k-Coloring does not admit an algorithm running in time $2^{o(|E(G)|)}n^{\mathcal{O}(1)}$, for every even k where $k > 3$. Here, n is the number of vertices in the input graph.

Let G be an instance of k-Coloring on n vertices with maximum degree bounded by $2(k-1)$. Here, $k > 3$ and k is an even number. We start by computing (in polynomial time) a harmonious coloring φ of G with $t \in \mathcal{O}(\sqrt{n})$ color classes such that each color class contains at most $\mathcal{O}(\sqrt{n})$ vertices using Proposition 1. Let $C_1, C_2, \ldots, C_t$ be the color classes of φ with exactly one edge between C_i and C_j in G, where $i, j \in [t]$. We modify the graph G and its harmonious coloring φ, to obtain a more structured instance, which will be useful later. For each $i \in [t]$, we add k new vertices $v^*_{i1}, v^*_{i2}, \ldots, v^*_{ik}$ to $V(G)$, and add them to C_i. We continue to call the modified graph as G and its harmonious coloring as φ with color classes $C_1, C_2, \ldots, C_t$. We note that $\{v^*_{ij} \mid i \in [t], j \in [k]\}$ induce an independent set in G. The purpose of adding these k new vertices is to ensure that if G is a *yes* instance of k-Coloring then there is a k-coloring c of G, such that for each $i \in [t]$ and $j \in [k]$, we have $c^{-1}(j) \cap C_i \neq \emptyset$. This will be helpful in simplifying some of the arguments later. Observe that the original instance is a *yes* instance of a k-Coloring is and only if the modified instance is a *yes* instance of k-Coloring. Moreover, given a k-coloring of G (modified graph), in polynomial time we can obtain another k-coloring c' of G such that for all $i \in [t], j \in [k]$ we have $c(v^*_{ij}) = j$. Also, we have $|V(G)| \in \mathcal{O}(n)$, and $|E(G)| \in \mathcal{O}(n)$, where n is the number of vertices in the original instance. Hereafter, whenever we talk about a solution c to k-Coloring in G (if it exists) we will assume (without explicitly mentioning) that for all $i \in [t]$ and $p \in [k]$ we have $C_i \cap c^{-1}(p) \neq \emptyset$. We now move to the description of the reduction.

We proceed by describing color class gadget $\mathcal{C}_i$, corresponding to the color class C_i, where $i \in [t]$, and gadgets to encode edges in G. Then we state the connection between various color class gadgets and edge gadgets. We let $k = 2\ell$, where $\ell \in \mathbb{N}$ and $\ell > 1$. We create an instance G' of Rainbow k-Coloring as described below.

- *Color class gadget.* Consider $i \in [t]$. The color class gadget $\mathcal{C}_i$ comprises of the set C_i, a vertex c_i, and a clique sequence $\mathbb{Z}_i = (U^i_1 \cup D^i_1, \ldots, U^i_{\ell-1} \cup D^i_{\ell-1})$ of order $(2k, \ell - 1)$. Here, for each $i \in [\ell - 1]$ we have $|U_i| = |D_i| = k$. For $r \in [\ell - 1]$ we let $U^i_r = \{u^i_{rp} \mid p \in [k]\}$,

and $D^i_r = \{d^i_{rp} \mid p \in [k]\}$. We add all the edges in $\{(c_i, v) \mid v \in C_i\}$ to $E(\mathcal{C}_i)$. Also, we add all the edges in $\{(v, w) \mid v \in C_i, w \in U^i_1 \cup D^i_1\}$ to $E(\mathcal{C}_i)$.

- *Connection between color class gadgets.* For each $i, j \in [t]$ where $i \neq j$, we add all the edges in $\{(w, w') \mid w \in U^i_{\ell-1} \cup D^i_{\ell-1}, w' \in U^j_{\ell-1} \cup D^j_{\ell-1}\}$ to $E(G')$.
- *Edge gadget.* Consider $i, j \in [t]$ with $i < j$. Recall that there is exactly one edge between C_i and C_j. Corresponding to this edge we create a path $P = (x^{ij}_1, \ldots, x^{ij}_{\ell-2}, z_{ij}, x^{ji}_{\ell-2}, \ldots, x^{ji}_1)$ on $k-3$ vertices, and add it to G'. We note that whenever we say vertex z_{ji} it refers to the vertex z_{ij} *i.e.* z_{ij} and z_{ji} denotes the same vertex.
- *Connection between color class gadgets and edge gadgets.* Consider $i, j \in [t]$, where $i < j$. Let (u^*_i, v^*_j) be the unique edge between C_i and C_j with $u^*_i \in C_i$ and $v^*_j \in C_i$. We add the edges $(u^*_i, x^{ij}_1), (x^{ji}_1, v^*_j)$ to $E(G')$. Notice that when $\ell = 2$ x^{ij}_1 does not exists. In this case, we add the edges $(u^*_i, z), (z, v^*_j)$ to $E(G')$. For each $r \in [\ell - 2]$ we add all the edges in $\{(x^{ij}_r, w) \mid w \in U^i_r \cup D^i_r\}$ to $E(G')$. Similarly, we add all the edges in $\{(x^{ji}_r, w) \mid w \in U^j_r \cup D^j_r\}$ to $E(G')$. Also, we add all the edges in $\{(z_{ij}, u) \mid u \in U^i_{\ell-1} \cup D^i_{\ell-1} \cup U^j_{\ell-1} \cup D^j_{\ell-1}\}$ to $E(G')$.

This finishes the construction of instance G' of Rainbow k-Coloring for the given instance G of k-Coloring. Before moving on to proving the equivalence between these instances, we create an edge-coloring $c_R : E(G') \to [k]$. Here, we create c_R based on a solution c to k-Coloring in G, assuming that is G a *yes* instance of k-Coloring. We will follow computation modulo k (color 0 is same as color k).

▶ **Definition 10.** Given a solution c to k-Coloring in G, we construct $c_R : E(G') \to [k]$ as follows.

1. For $i \in [t]$, and $v \in C_i$ we set $c_R(v, c_i) = c(v)$.
2. For $i, j \in [t]$, $i < j$ let (u^*_i, v^*_j) be the unique edge between C_i and C_j. Consider the path $P = (u^*_i, x^{ij}_1, \ldots x^{ij}_{\ell-2}, z_{ij}, x^{ji}_{\ell-2}, \ldots x^{ji}_1, v^*_j)$. We arbitrarily assign unique integers in $[k] \setminus \{c(u^*_i), c(v^*_j)\}$ to $c_R(e)$, for each $e \in E(P)$.
3. For $i \in [t]$, a vertex $v \in C_i$, and $p \in [k]$ we set $c_R(v, u^i_{1p}) = p - 1$, and $c_R(v, d^i_{1p}) = p$.
4. For $i \in [t]$, $r \in [\ell - 1]$, and $p, q \in [k]$ we set $c_R(d^i_{rp}, u^i_{rq}) = p$.
5. For $i, j \in [t]$, where $i \neq j$, $r \in [\ell - 1]$, and $p \in [k]$ we set $c_R(x^{ij}_r, u^i_{rp}) = p$, and $c_R(x^{ij}_r, d^i_{rp}) = p + 1$.
6. For $i \in [t]$, $r \in [\ell - 2]$, $p, q \in [k]$ we set $c_R(d^i_{(r+1)p}, d^i_{rq}) = p$, and $c_R(u^i_{rp}, u^i_{(r+1)q}) = p$.
7. For $i, j \in [t]$ where $i \neq j$, $p, q \in [k]$ we set $c_R(u^i_{(\ell-1)p}, d^j_{(\ell-1)q}) = p$, $c_R(u^i_{(\ell-1)p}, z_{ij}) = p$, and $c_R(d^i_{(\ell-1)p}, z_{ij}) = p + 1$.
8. For $i \in [t]$, $r \in [\ell - 2]$, $p, q \in [k]$ we set $c_R(u^i_{rp}, d^i_{(r+1)q}) = q$ and $c_R(u^i_{(r+1)p}, d^i_{rq}) = p$.
9. For all $i \in [t]$, $r \in [\ell - 1]$, $p, q \in [k]$, where $p \neq q$ we set $c_R(u^i_{rp}, u^i_{rq}) = k$.
10. For all the remaining edges in $E(G')$, c_R assigns it an integer in $[k]$ arbitrarily.

Next, we prove some lemmata that will be useful in establishing equivalence between the instance G of k-Coloring and the instance G' of Rainbow k-Coloring.

▶ **Lemma 11.** *For $i, j \in [t]$, where $i \neq j$, let P be a path between c_i and c_j with at most k edges in G'. If $\ell > 2$ then P contains the edge $(x^{ij}_{\ell-2}, z_{ij})$. Otherwise, P contains the edge (u, z_{ij}), where u is the unique vertex in C_i that in adjacent to a vertex in C_j.*

▶ **Lemma 12.** *For $i, j \in [t]$, where $i \neq j$ let (u^*, v^*) be the unique edge between C_i and C_j with $u^* \in C_i$ and $v^* \in C_j$. There is exactly one path, namely $(c_i, u^*, x^{ij}_1, \ldots, x^{ij}_{\ell-2}, z_{ij}, x^{ji}_{\ell-2}, \ldots, x^{ji}_1, v^*, c_j)$ in G' between c_i and c_j that has at most k edges.*

▶ **Lemma 13.** *Let G be a yes instance of k-Coloring, and c be one of its solution. Furthermore, let $c_R : E(G') \to [k]$ be the coloring given by Definition 10 for the coloring c of G. For all $i \in [t]$, and $u, v \in V(\mathcal{C}_i) \cup \{z_{ij} \mid j \in [k] \setminus \{i\}\} \cup \{x_r^{ij} \mid j \in [t] \setminus \{i\}, r \in [\ell - 2]\}$ there is a rainbow path between u and v in G'.*

▶ **Lemma 14.** *Let G be a yes instance of k-Coloring, and c be one of its solution. Furthermore, let $c_R : E(G') \to [k]$ be the coloring given by Definition 10 for the coloring c of G. For all $i, j \in [t]$ where $i \neq j$, $u \in V(\mathcal{C}_i) \cup \{z_{ij'} \mid j' \in [k] \setminus \{i\}\} \cup \{x_r^{ij'} \mid j' \in [t] \setminus \{i\}, r \in [\ell - 2]\}$ and $v \in V(\mathcal{C}_j) \cup \{z_{ji'} \mid i' \in [k] \setminus \{j\}\} \cup \{x_r^{ji'} \mid i' \in [t] \setminus \{j\}, r \in [\ell - 2]\}$ there is a rainbow path between u and v in G'.*

We now establish equivalence between the instance G of k-Coloring and the instance G' of Rainbow k-Coloring.

▶ **Lemma 15.** *G' is a yes instance of k-Coloring if and only if G' is a yes instance of* Rainbow k-Coloring.

▶ **Theorem 16.** Rainbow k-Coloring *does not admit an algorithm running in time $2^{o(|E(G)|)} n^{\mathcal{O}(1)}$, unless* ETH *fails. Here, n is the number of vertices in the input graph, and k is an even number greater than 3.*

4.3 Lower Bound for Rainbow k-Coloring, $k > 3$ and odd

In this section, we show that Rainbow k-Coloring does not admit an algorithm running in time $2^{o(|E(G)|)} n^{\mathcal{O}(1)}$, for every odd k where $k > 3$. Here, n is the number of vertices in the input graph.

Let G be an instance of k-Coloring on n vertices with maximum degree bounded by $2(k-1)$. Here, $k > 3$ and k is an odd number. We start by computing (in polynomial time) a harmonious coloring φ of G with $t \in \mathcal{O}(\sqrt{n})$ color classes such that each color class contains at most $\mathcal{O}(\sqrt{n})$ vertices using Proposition refprop:compute-harmonious-coloring. Let $C_1, C_2, \ldots, C_t$ be the color classes of φ. From the discussion in Section 3, we assume that for $i, j \in [t]$, $i \neq j$ there is exactly one edge between C_i and C_j in G. As discussed in Section 4.2, we modify the graph G and its harmonious coloring φ, to obtain a more structured (equivalent) instance of k-Coloring. This is achieved by adding k new vertices $v_{i1}^*, v_{i2}^*, \ldots, v_{ik}^*$ to C_i (and G) for each $i \in [t]$. The purpose of adding these k new vertices is to ensure that if G is a *yes* instance of k-Coloring then there is a k-coloring c of G, such that for each $i \in [t]$ and $j \in [k]$, we have $c^{-1}(j) \cap C_i \neq \emptyset$. Hereafter, whenever we talk about a solution c to k-Coloring in G (if it exists) we will assume (without explicitly mentioning) that for all $i \in [t]$ and $p \in [k]$ we have $C_i \cap c^{-1}(p) \neq \emptyset$.

We move to the description of the reduction. We first describe the color class gadget $\mathcal{C}_i$, corresponding to each color class C_i, where $i \in [t]$, and gadgets to encode edges in G. We also have a link vertex which is connected to all color class gadgets (but not all vertices). After this, we state connections between color class gadgets and edge gadgets. We let $k = 2\ell + 1$, where $\ell \in \mathbb{N}$ and $\ell \geq 2$. We create an instance G' of Rainbow k-Coloring as follows.

- *Color class gadget.* Consider $i \in [t]$. The color class gadget $\mathcal{C}_i$ comprises of the set C_i, a vertex c_i, and a clique sequence $\mathbb{Z}_i = (U_1^i \cup D_1^i, \ldots, U_{\ell-1}^i \cup D_{\ell-1}^i)$ of order $(2k, \ell - 1)$. Here, for each $i \in [\ell - 1]$ we have $|U_i| = |D_i| = k$. For $r \in [\ell - 1]$ we let $U_r^i = \{u_{rp}^i \mid p \in [k]\}$ and $D_r^i = \{d_{rp}^i \mid p \in [k]\}$. We add all the edges in $\{(c_i, v) \mid v \in C_i\}$ to $E(\mathcal{C}_i)$. Also, we add all the edges in $\{(v, w) \mid v \in C_i, w \in U_1^i \cup D_1^i\}$ to $E(\mathcal{C}_i)$.
- *Link vertex and its connection to color class gadgets.* We add a vertex z to G'. For each $i \in [t]$, we add all the edges in $\{(z, w) \mid w \in U_{\ell-1}^i \cup D_{\ell-1}^i\}$ to $E(G')$.

- *Edge gadget.* Consider $i, j \in [t]$ with $i \neq j$. Recall that there is exactly one edge between C_i and C_j. Corresponding to this edge we create a path $P = (x_1^{ij}, \ldots, x_{\ell-1}^{ij}, x_{\ell-1}^{ji}, \ldots, x_1^{ji})$ on $k-3$ vertices, and add it to G'.
- *Connection between color class gadgets and edge gadgets.* Consider $i, j \in [t]$, where $i \neq j$. Let (u_i^*, v_j^*) be the unique edge between C_i and C_j with $u_i^* \in C_i$ and $v_j^* \in C_i$. We add the edges $(u_i^*, x_1^{ij}), (x_1^{ji}, v_j^*)$ to $E(G')$. For each $r \in [\ell - 1]$ we add all the edges in $\{(x_r^{ij}, w) \mid w \in U_r^i \cup D_r^i\}$ to $E(G')$. Similarly, we add all the edges in $\{(x_r^{ji}, w) \mid w \in U_r^j \cup D_r^j\}$ to $E(G')$.

This finishes the construction of the instance G' of Rainbow k-Coloring for the given instance G of k-Coloring. Before moving on to proving the equivalence between these instances, we create an edge-coloring $c_R : E(G') \to [k]$. Here, we create c_R based on a solution c to k-Coloring in G, assuming that is G a *yes* instance of k-Coloring. We will follow computation modulo k (color 0 is same as color k).

▶ **Definition 17.** Given a solution c to k-Coloring in G, we construct $c_R : E(G') \to [k]$ as follows.

1. For $i \in [t]$, and $v \in C_i$ we set $c_R(v, c_i) = c(v)$.
2. For $i, j \in [t]$, $i \neq j$ let (u_i^*, v_j^*) be the unique edge between C_i and C_j. Consider the path $P = (u_i^*, x_1^{ij}, \ldots x_{\ell-1}^{ij}, x_{\ell-1}^{ji}, \ldots x_1^{ji}, v_j^*)$. We arbitrarily assign unique integers in $[k] \setminus \{c(u_i^*), c(v_j^*)\}$ to $c_R(e)$, for each $e \in E(P)$.
3. For $i \in [t]$, a vertex $v \in C_i \cup \{x_1^{ij} \mid j \in [t] \setminus \{i\}\}$, and $p \in [k]$ we set $c_R(v, u_{1p}^i) = p - 1$, and $c_R(v, d_{1p}^i) = p$.
4. For $i \in [t]$, $r \in [\ell - 1]$, and $p, q \in [k]$ we set $c_R(d_{rp}^i, u_{rq}^i) = p$.
5. For $i, j \in [t]$, where $i \neq j$, $r \in [\ell - 1]$, and $p \in [k]$ we set $c_R(x_r^{ij}, u_{rp}^i) = p$, and $c_R(x_r^{ij}, d_{rp}^i) = p + 1$.
6. For $i \in [t]$, $r \in [\ell - 2]$, $p, q \in [k]$ we set $c_R(d_{(r+1)p}^i, d_{rq}^i) = p$, and $c_R(u_{rp}^i, u_{(r+1)q}^i) = p$.
7. For $i \in [t]$, $p \in [k]$ we set $c_R(u_{(\ell-1)p}^i, z) = p$, and $c_R(d_{(\ell-1)p}^i, z) = p - 1$.
8. For $i \in [t]$, $r \in [\ell - 2]$, $p, q \in [k]$ we set $c_R(u_{rp}^i, d_{(r+1)q}^i) = q$ and $c_R(u_{(r+1)p}^i, d_{rq}^i) = p$.
9. For all $i \in [t]$, $r \in [\ell]$, $p, q \in [k]$, where $p \neq q$ we set $c_R(u_{rp}^i, u_{rq}^i) = k$.
10. For all the remaining edges in $E(G')$, c_R assigns it an integer in $[k]$ arbitrarily.

Next, we prove some lemmata that will be useful in establishing the equivalence between the instance G of k-Coloring and the instance G' of Rainbow k-Coloring.

▶ **Lemma 18.** *For $i, j \in [t]$, where $i \neq j$, let P be a path between c_i and c_j with at most k edges in G'. Then $(x_{\ell-1}^{ij}, x_{\ell-1}^{ij}) \in E(P)$.*

▶ **Lemma 19.** *For $i, j \in [t]$, where $i \neq j$ let (u_i^*, v_j^*) be the unique edge between C_i and C_j with $u_i^* \in C_i$ and $v_j^* \in C_j$. There is exactly one path, namely $(c_i, u_i^*, x_1^{ij}, \ldots, x_{\ell-1}^{ij}, x_{\ell-1}^{ji}, \ldots, x_1^{ji}, v_j^*, c_j)$ in G' between c_i and c_j that has at most k edges.*

▶ **Lemma 20.** *Let G be a yes instance of k-Coloring, and c be one of its solution. Furthermore, let $c_R : E(G') \to [k]$ be the coloring given by Definition 17 for the coloring c of G. For all $i \in [t]$, and $u, v \in V(\mathcal{C}_i) \cup \{x_r^{ij} \mid j \in [t] \setminus \{i\}, r \in [\ell - 1]\} \cup \{z\}$ there is a rainbow path between u and v in G'.*

▶ **Lemma 21.** *Let G be a yes instance of k-Coloring, and c be one of its solution. Furthermore, let $c_R : E(G') \to [k]$ be the coloring given by Definition 17 for the coloring c of G. For all $i, j \in [t]$ where $i \neq j$, $u \in V(\mathcal{C}_i) \cup \{x_r^{ij'} \mid j' \in [t] \setminus \{i\}, r \in [\ell - 1]\}$ and $v \in \mathcal{C}_j \cup \{x_r^{ji'} \mid i' \in [t] \setminus \{j\}, r \in [\ell - 1]\}$ there is a rainbow path between u and v in G'.*

▶ **Lemma 22.** *G' is a yes instance of k-Coloring if and only if G' is a yes instance of* Rainbow k-Coloring.

▶ **Theorem 23.** Rainbow k-Coloring *does not admit an algorithm running in time $2^{o(|E(G)|)}n^{\mathcal{O}(1)}$, unless* ETH *fails. Here, n is the number of vertices in the input graph, and k is an odd number greater than 3.*

5 FPT Algorithm for Subset Rainbow k-Coloring

In this section, we design an FPT algorithm running in time $\mathcal{O}(2^{|S|}n^{\mathcal{O}(1)})$ for Subset Rainbow k-Coloring, when parameterized by $|S|$. Our algorithm is based on the technique of color coding, which was first introduced by Alon et al. [2]. We first describe a randomized algorithm for Subset Rainbow k-Coloring, which we derandomize using splitters.

The intuition behind the algorithm is as follows. Let (G, S) be an instance of Subset Rainbow k-Coloring on n vertices and m edges. For a solution $c_R : E(G) \to [k]$, to Subset Rainbow k-Coloring in (G, S) the following holds. For each $(u, v) \in S$, there exist a path P from u to v in G with at most k edges such that for all $e, e' \in E(P)$, where $e \neq e'$ we have $c_R(e) \neq c_R(e')$. Therefore, at most $k|S|$ edges in G seems to be "important" for us, *i.e.* if we color at most $k|S|$ edges "nicely" then we would obtain the desired soultion. To capture this, we start by randomly coloring edges in G, hoping that with sufficiently high probability we obtain a coloring that colors the desired set of edges "nicely". Once we have obtained such a "nice" coloring, we employ the algorithm of Kowalik and Lauri [19] to check if there is a rainbow path for each $(u, v) \in S$. We note that we use the algorithm given by [19] instead of the one in [28] because the latter requires exponential space.

Algorithm Rand-SRC. Let $c : E(G) \to [k]$ be a coloring of $E(G)$, where each edge is colored with one of the colors in $[k]$ uniformly and independently at random. If for each $(u, v) \in S$, there is rainbow path between u and v in G' with edge-coloring c then the algorithm return c as a solution to Subset Rainbow k-Coloring in (G, S). Otherwise, it returns *no*. We note that for a given graph G with edge-coloring c, and vertices u and v, in time $2^k n^{\mathcal{O}(1)}$ time we can check if there is a rainbow path between u and v in G' by using the algorithm given by Corollary 5 in [19]. This completes the description of the algorithm.

We now proceed to show how we can obtain an algorithm with constant success probability.

▶ **Theorem 24.** *There is an algorithm that, given an instance (G, S) of* Subset Rainbow k-Coloring*, in time $2^{\mathcal{O}(|S|k\log k)}n^{\mathcal{O}(1)}$ either returns no or outputs a solution to* Subset Rainbow k-Coloring *in (G, S). Moreover, if the input is a yes instance of* Subset Rainbow k-Coloring*, then it returns a solution with positive constant probability.*

We start by defining some terminologies which will be useful in derandomization of our algorithm (see [12, 27]). An (n, p, ℓ)-*splitter* $\mathcal{F}$, is a family of functions from $[n]$ to ℓ such that for every $S \subseteq [n]$ of size at most p there is a function $f \in \mathcal{F}$ such that f splits S evenly. That is, for all $i, j \in [\ell]$, $|f^{-1}(i)|$ and $|f^{-1}(j)|$ differs by at most 1. Observe that when $\ell \geq p$ then for any $S \subseteq [n]$ of size at most p and a function $f \in \mathcal{F}$ that splits S, we have $|f^{-1}(i) \cap S| \leq 1$, for all $i \in [\ell]$. An (n, ℓ, ℓ)-splitter is called as an (n, ℓ)-*perfect hash family*. Moreover, for any $\ell \geq 1$, we can construct an (n, ℓ)-perfect hash family of size $e^{\ell}\ell^{\mathcal{O}(\log \ell)} \log n$ in time $e^{\ell}\ell^{\mathcal{O}(\log \ell)} n \log n$ [27].

We next move to the description of derandomization of the algorithm presented in Theorem 24. For the sake of simplicity in explanation, we associate each $e \in E(G)$ with a unique integer, say i_e in $[m]$, and whenever we refer to e as an integer, we actually refer to the integer

i_e. We start by computing an $(m, k|S|)$-perfect hash family $\mathcal{F}$ of size $e^{k|S|}(k|S|)^{\mathcal{O}(\log k|S|)} \log m$ in time $e^{k|S|}(k|S|)^{\mathcal{O}(\log k|S|)} m \log m$ using the algorithm of Naor et al. [27]. We will create a family of function $\mathcal{F}'$ from $[m]$ to $[k]$ of size $e^{k|S|}(k|S|)^{\mathcal{O}(\log k|S|)} k^{k|S|} \log m$. Towards this, consider an $f \in \mathcal{F}$ and a partition $\mathcal{P} = \{P_1, P_2, \ldots, P_{k'}\}$ of $[k|S|]$ into k' sets, where $k' \leq k$. We let $f_{\mathcal{P}}$ to be the function obtained from f as follows. For each $i \in [k']$ we have $f_{\mathcal{P}}^{-1}(i) = \cup_{x \in P_i} f^{-1}(x)$. For every such pair f and $\mathcal{P}$, we add the function $f_{\mathcal{P}}$ to the set $\mathcal{F}'$. We will call such an $\mathcal{F}'$ as $(m, k|S|, k)$-*unified perfect hash* family. Observe that $\mathcal{F}'$ has size at most $e^{k|S|}(k|S|)^{\mathcal{O}(\log k|S|)} k^{k|S|} \log m$. We now describe the derandomized algorithm SRC, which is a result of derandomization of Rand-SRC.

Algorithm SRC. Given an instance (G, S) of Subset Rainbow k-Coloring, the algorithm start by computing an $(m, k|S|, k)$-unified perfect hash family $\mathcal{F}'$. If there exists $c : E(G) \to [k]$, where $c \in \mathcal{F}'$ such that for each $(u, v) \in S$, there is rainbow path between u and v in G' with the edge-coloring c then we return c as a solution to Subset Rainbow k-Coloring in (G, S). Otherwise, we return that (G, S) is a *no* instance of Subset Rainbow k-Coloring. We note that for a given graph G with edge-coloring c, and vertices u and v, in time $2^k n^{\mathcal{O}(1)}$ time we can check if there is a rainbow path between u and v in G' by using the algorithm given by Corollary 5 in [19]. This completes the description of the algorithm.

▶ **Theorem 25.** *Given an instance (G, k) of* Subset Rainbow k-Coloring*, the algorithm* SRC *either correctly reports that (G, k) is a no instance of* Subset Rainbow k-Coloring *or returns a solution to* Subset Rainbow k-Coloring *in (G, S). Moreover,* SRC *runs in time $2^{\mathcal{O}(|S|)} n^{\mathcal{O}(1)}$, for every fixed k. Here, $n = |V(G)|$.*

▶ **Corollary 26.** Steiner Rainbow k-Coloring *admits an algorithm running in time $2^{\mathcal{O}(|S|^2)} n^{\mathcal{O}(1)}$.*

6 Conclusion

In this paper, we proved that for all $k \geq 3$, Rainbow k-Coloring does not admit an algorithm running in time $2^{o(|E(G)|)} n^{\mathcal{O}(1)}$, unless ETH fails. This (partially) resolves the conjecture of Kowalik et al. [20], which states that for every $k \geq 2$, Rainbow k-Coloring does not admit an algorithm running in time $2^{o(|E(G)|)} n^{\mathcal{O}(1)}$. It would be an interesting direction to study whether or not Rainbow k-Coloring admits an algorithm running in time $2^{o(|E(G)|)} n^{\mathcal{O}(1)}$, for $k = 2$. We also studied the problem Steiner Rainbow k-Coloring, and proved that for every $k \geq 3$ the problem does not admit an algorithm running in time $2^{o(|S|^2)} n^{\mathcal{O}(1)}$, unless ETH fails. We complemented this by designing an algorithm for Subset Rainbow k-Coloring running in time $2^{\mathcal{O}(|S|)} n^{\mathcal{O}(1)}$, which implies an algorithm running in time $2^{\mathcal{O}(|S|^2)} n^{\mathcal{O}(1)}$ for Steiner Rainbow k-Coloring. It would be interesting to study whether or not Steiner Rainbow k-Coloring admits an algorithm running in time $2^{o(|S|^2)} n^{\mathcal{O}(1)}$, for $k = 2$. Kowalik et al. [20] also conjectured that for every $k \geq 2$, Rainbow k-Coloring does not admit an algorithm running in time $2^{o(n^2)} n^{\mathcal{O}(1)}$, which is another interesting direction of research.

Acknowledgements. The author is thankful to Saket Saurabh for helpful discussions.

References

1. Akanksha Agrawal, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Split contraction: The untold story. In *34th Symposium on Theoretical Aspects of Computer Science, (STACS)*, pages 5:1–5:14, 2017.
2. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.
3. Prabhanjan Ananth, Meghana Nasre, and Kanthi K. Sarpatwar. Rainbow Connectivity: Hardness and Tractability. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 13, pages 241–251, 2011.
4. Yair Caro, Arie Lev, Yehuda Roditty, Zsolt Tuza, and Raphael Yuster. On rainbow connection. *Electronic Journal of Combinatorics*, 15(1):R57, 2008.
5. Sourav Chakraborty, Eldar Fischer, Arie Matsliah, and Raphael Yuster. Hardness and algorithms for rainbow connection. *Journal of Combinatorial Optimization*, 21(3):330–347, 2011.
6. L. Sunil Chandran and Deepak Rajendraprasad. Rainbow colouring of split and threshold graphs. In *18th Annual International Conference: Computing and Combinatorics, (COCOON)*, pages 181–192, 2012.
7. 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.
8. Gary Chartrand, Garry L Johns, Kathleen A McKeon, and Ping Zhang. Rainbow connection in graphs. *Mathematica Bohemica*, 133(1):85–98, 2008.
9. Gary Chartrand and Ping Zhang. *Chromatic graph theory*. CRC press, 2008.
10. Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight bounds for graph homomorphism and subgraph isomorphism. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 1643–1649, 2016.
11. Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight bounds for graph homomorphism and subgraph isomorphism. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1643–1649, 2016.
12. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
13. Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
14. Keith Edwards. The harmonious chromatic number and the achromatic number. *Surveys in Combinatorics*, pages 13–48, 1997.
15. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
16. Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.
17. Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
18. Christian Komusiewicz. Tight running time lower bounds for vertex deletion problems. *arXiv preprint arXiv:1511.05449*, 2015.
19. Lukasz Kowalik and Juho Lauri. On finding rainbow and colorful paths. *Theoretical Computer Science*, 628(C):110–114, 2016.
20. Lukasz Kowalik, Juho Lauri, and Arkadiusz Socala. On the fine-grained complexity of rainbow coloring. In *24th Annual European Symposium on Algorithms, (ESA)*, pages 58:1–58:16, 2016.

21 V.B. Le and Z. Tuza. *Finding Optimal Rainbow Connection is Hard*. Preprints aus dem Institut für Informatik / CS. Inst. für Informatik, 2009. URL: `https://books.google.no/books?id=0ErVPgAACAAJ`.

22 Sin-Min Lee and John Mitchem. An upper bound for the harmonious chromatic number. *Journal of Graph Theory*, 11(4):565–567, 1987.

23 Xueliang Li, Yongtang Shi, and Yuefang Sun. Rainbow connections of graphs: A survey. *Graphs and Combinatorics*, 29(1):1–38, 2013.

24 Xueliang Li and Yuefang Sun. *Rainbow connections of graphs*. Springer Science & Business Media, 2012.

25 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, pages 41–71, 2011.

26 Colin McDiarmid and Luo Xinhua. Upper bounds for harmonious coloring. *Journal of Graph Theory*, 15(6):629–636, 1991.

27 M. Naor, L. J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 182–191, 1995.

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.

Faster Monte-Carlo Algorithms for Fixation Probability of the Moran Process on Undirected Graphs*

Krishnendu Chatterjee[1], Rasmus Ibsen-Jensen[2], and Martin A. Nowak[3]

1 **IST Austria, Klosterneuburg, Austria**
krish@ist.ac.at

2 **IST Austria, Klosterneuburg, Austria**
ribsen@ist.ac.at

3 **Program for Evolutionary Dynamics, Harvard University, Cambridge, USA**
martin_nowak@harvard.edu

Abstract

Evolutionary graph theory studies the evolutionary dynamics in a population structure given as a connected graph. Each node of the graph represents an individual of the population, and edges determine how offspring are placed. We consider the classical birth-death Moran process where there are two types of individuals, namely, the residents with fitness 1 and mutants with fitness r. The fitness indicates the reproductive strength. The evolutionary dynamics happens as follows: in the initial step, in a population of all resident individuals a mutant is introduced, and then at each step, an individual is chosen proportional to the fitness of its type to reproduce, and the offspring replaces a neighbor uniformly at random. The process stops when all individuals are either residents or mutants. The probability that all individuals in the end are mutants is called the fixation probability, which is a key factor in the rate of evolution. We consider the problem of approximating the fixation probability.

The class of algorithms that is extremely relevant for approximation of the fixation probabilities is the Monte-Carlo simulation of the process. Previous results present a polynomial-time Monte-Carlo algorithm for undirected graphs when r is given in unary. First, we present a simple modification: instead of simulating each step, we discard *ineffective* steps, where no node changes type (i.e., either residents replace residents, or mutants replace mutants). Using the above simple modification and our result that the number of effective steps is concentrated around the expected number of effective steps, we present faster polynomial-time Monte-Carlo algorithms for undirected graphs. Our algorithms are always at least a factor $O(n^2/\log n)$ faster as compared to the previous algorithms, where n is the number of nodes, and is polynomial even if r is given in binary. We also present lower bounds showing that the upper bound on the expected number of effective steps we present is asymptotically tight for undirected graphs.

1998 ACM Subject Classification J.1.1 Biology and genetics, E.1.3 Graphs and networks

Keywords and phrases Graph algorithms, Evolutionary biology, Monte-Carlo algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.61

* Some proofs are missing. See the full version [2]

1 Introduction

In this work we present faster Monte-Carlo algorithms for approximation of the fixation probability of the fundamental Moran process on population structures with symmetric interactions. We start with the description of the problem.

Evolutionary dynamics. Evolutionary dynamics act on populations, where the composition of the population changes over time due to mutation and selection. Mutation generates new types and selection changes the relative abundance of different types. A fundamental concept in evolutionary dynamics is the fixation probability of a new mutant [7, 11, 13, 14]: Consider a population of n *resident* individuals, each with a fitness value 1. A single *mutant* with non-negative fitness value r is introduced in the population as the initialization step. Intuitively, the fitness represents the reproductive strength. In the classical Moran process the following *birth-death* stochastic steps are repeated: At each time step, one individual is chosen at random proportional to the fitness to reproduce and one other individual is chosen uniformly at random for death. The offspring of the reproduced individual replaces the dead individual. This stochastic process continues until either all individuals are mutants or all individuals are residents. The *fixation probability* is the probability that the mutants take over the population, which means all individuals are mutants. A standard calculation shows that the fixation probability is given by $(1-(1/r))/(1-(1/r^n))$. The correlation between the relative fitness r of the mutant and the fixation probability is a measure of the effect of natural selection. The rate of evolution, which is the rate at which subsequent mutations accumulate in the population, is proportional to the fixation probability, the mutation rate, and the population size n. Hence fixation probability is a fundamental concept in evolution.

Evolutionary graph theory. While the basic Moran process happens in well-mixed population (all individuals interact uniformly with all others), a fundamental extension is to study the process on population structures. Evolutionary graph theory studies this phenomenon. The individuals of the population occupy the nodes of a connected graph. The links (edges) determine who interacts with whom. Basically, in the birth-death step, for the death for replacement, a neighbor of the reproducing individual is chosen uniformly at random. Evolutionary graph theory describes evolutionary dynamics in spatially structured population where most interactions and competitions occur mainly among neighbors in physical space [12, 3, 8, 15]. Undirected graphs represent population structures where the interactions are symmetric, whereas directed graphs allow for asymmetric interactions. The fixation probability depends on the population structure [12, 1, 9, 4]. Thus, the fundamental computational problem in evolutionary graph theory is as follows: given a population structure (i.e., a graph), the relative fitness r, and $\epsilon > 0$, compute an ϵ-approximation of the fixation probability.

Monte-Carlo algorithms. A particularly important class of algorithms for biologists is the Monte-Carlo algorithms, because it is simple and easy to interpret. The Monte-Carlo algorithm for the Moran process basically requires to simulate the process, and from the statistics obtain an approximation of the fixation probability. Hence, the basic question we address in this work is simple Monte-Carlo algorithms for approximating the fixation probability. It was shown in [6] that simple simulation can take exponential time on directed graphs and thus we focus on undirected graphs. The main previous algorithmic result in this area [5] presents a polynomial-time Monte-Carlo algorithm for undirected graphs when r is given in unary. The main result of [5] shows that for undirected graphs it suffices to run each simulation for polynomially many steps.

Table 1 Comparison with previous work, for constant $r > 1$. We denote by n, Δ, τ, and ϵ, the number of nodes, the maximum degree, the random variable for the fixation time, and the approximation factor, respectively. The results in the column "All steps" is from [5], except that we present the dependency on Δ, which was considered as n in [5]. The results of the column "Effective steps" is the results of this paper

	All steps	Effective steps
#steps in expectation	$O(n^2\Delta^2)$	$O(n\Delta)$
Concentration bounds	$\Pr[\tau \geq \frac{n^2\Delta^2 rx}{r-1}] \leq 1/x$	$\Pr[\tau \geq \frac{6n\Delta x}{\min(r-1,1)}] \leq 2^{-x}$
Sampling a step	$O(1)$	$O(\Delta)$
Fixation algo	$O(n^6\Delta^2\epsilon^{-4})$	$O(n^2\Delta^2\epsilon^{-2}(\log n + \log \epsilon^{-1}))$

Our contributions. In this work our main contributions are as follows:

1. *Faster algorithm for undirected graphs* First, we present a simple modification: instead of simulating each step, we discard *ineffective* steps, where no node changes type (i.e., either residents replace residents, or mutants replace mutants). We then show that the number of effective steps is concentrated around the expected number of effective steps. The sampling of each effective step is more complicated though than sampling of each step. We then present an efficient algorithm for sampling of the effective steps, which requires $O(m)$ preprocessing and then $O(\Delta)$ time for sampling, where m is the number of edges and Δ is the maximum degree. Combining all our results we obtain faster polynomial-time Monte-Carlo algorithms: Our algorithms are always at least a factor $n^2/\log n$ times a constant (in most cases $n^3/\log n$ times a constant) faster as compared to the previous algorithm, and is polynomial even if r is given in binary. We present a comparison in Table 1, for constant $r > 1$ (since the previous algorithm is not in polynomial time for r in binary). For a detailed comparison see the full version [2].
2. *Lower bounds* We also present lower bounds showing that the upper bound on the expected number of effective steps we present is asymptotically tight for undirected graphs.

Related complexity result. While in this work we consider evolutionary graph theory, a related problem is evolutionary games on graphs (which studies the problem of frequency dependent selection). The approximation problem for evolutionary games on graphs is considerably harder (e.g., PSPACE-completeness results have been established) [10].

Technical contributions. Note that for the problem we consider the goal is not to design complicated efficient algorithms, but simple algorithms that are efficient. By simple, we mean something that is related to the process itself, as biologists understand and interpret the Moran process well. Our main technical contribution is a simple idea to discard ineffective steps, which is intuitive, and we show that the simple modification leads to significantly faster algorithms. We show a gain of factor $O(n\Delta)$ due to the effective steps, then lose a factor of $O(\Delta)$ due to sampling, and our other improvements are due to better concentration results. We also present an interesting family of graphs for the lower bound examples. Technical proofs omitted due to lack of space are in the full version [2].

2 Moran process on graphs

Connected graph and type function. We consider the population structure represented as a connected graph. There is a connected *graph* $G = (V, E)$, of n nodes and m edges, and two types $T = \{t_1, t_2\}$. The two types represent residents and mutants, and in the technical

exposition we refer to them as t_1 and t_2 for elegant notation. We say that a node v is a *successor* of a node u if $(u, v) \in E$. The graph is *undirected* if for all $(u, v) \in E$ we also have $(v, u) \in E$, otherwise it is *directed*. There is a *type function* f mapping each node v to a type $t \in T$. Each type t is in turn associated with a positive integer $w(t)$, the type's fitness denoting the corresponding reproductive strength. Without loss of generality, we will assume that $r = w(t_1) \geq w(t_2) = 1$, for some number r (the process we consider does not change under scaling, and r denotes relative fitness). Let $W(f) = \sum_{u \in V} w(f(u))$ be the total fitness. For a node v let $\deg v$ be the degree of v in G. Also, let $\Delta = \max_{v \in V} \deg v$ be the maximum degree of a node. For a type t and type function f, let $V_{t,f}$ be the nodes mapped to t by f. Given a type t and a node v, let $f[v \to t]$ denote the following function: $f[v \to t](u) = t$ if $u = v$ and $f(u)$ otherwise.

Moran process on graphs. We consider the following classical Moran birth-death process where a *dynamic evolution step* of the process changes a type function from f to f' as follows:

1. First a node v is picked at random with probability proportional to $w(f(v))$, i.e. each node v has probability of being picked equal to $\frac{w(f(v))}{W(f)}$.
2. Next, a successor u of v is picked uniformly at random.
3. The type of u is then changed to $f(v)$. In other words, $f' = f[u \to f(v)]$.

Fixation. A type t *fixates* in a type function f if f maps all nodes to t. Given a type function f, repeated applications of the dynamic evolution step generate a sequence of type functions $f = f_1, f_2, \ldots, f_\infty$. Note that if a type has fixated (for some type t) in f_i then it has also fixated in f_j for $i < j$. We say that a process has *fixation time* i if f_i has fixated but f_{i-1} has not. We say that an initial type function f has fixation probability p for a type t, if the probability that t eventually fixates (over the probability measure on sequences generated by repeated applications of the dynamic evolution step f)

Basic questions. We consider the following basic questions:

1. *Fixation problem* Given a type t, what is the fixation probability of t averaged over the n initial type functions with a single node mapping to t?
2. *Extinction problem* Given a type t, what is the fixation probability of t averaged over the n initial type functions with a single node *not* mapping to t?
3. *Generalized fixation problem* Given a graph, a type t and an type function f what is the fixation probability of t in G, when the initial type function is f?

▶ Remark. Note that in the *neutral* case when $r = 1$, the fixation problem has answer $1/n$ and extinction problem has answer $1 - 1/n$. Hence, in the rest of the paper we will consider $r > 1$. Also, to keep the presentation focused, in the main article, we will consider fixation and extinction of type t_1. In the full version [2] we also present another algorithm for the extinction of t_2.

Results. We will focus on undirected graphs. For undirected graphs, we will give new FPRAS (fully polynomial, randomized approximation scheme) for the fixation and the extinction problem, and a polynomial-time algorithm for an additive approximation of the generalized fixation problem. There exists previous FPRAS for the fixation and extinction problems [5]. Our upper bounds are at least a factor of $O(\frac{n^2}{\log n})$ (most cases $O(\frac{n^3}{\log n})$) better and always in $\mathrm{Poly}(n, 1/\epsilon)$, whereas the previous algorithms are not in polynomial time for r given in binary.

3 Discarding ineffective steps

We consider undirected graphs. Previous work by Diaz et al. [5] showed that the expected number of dynamic evolution steps till fixation is polynomial, and then used it to give a polynomial-time Monte-Carlo algorithm. Our goal is to improve the quite high polynomial-time complexity, while giving a Monte-Carlo algorithm. To achieve this we define the notion of effective steps.

Effective steps. A dynamic evolution step, which changes the type function from f to f', is *effective* if $f \neq f'$ (and *ineffective* otherwise). The idea is that steps in which no node changes type (because the two nodes selected in the dynamic evolution step already had the same type) can be discarded, without changing which type fixates/gets eliminated.

Two challenges. The two challenges are as follows:

1. *Number of steps* The first challenge is to establish that the expected number of effective steps is asymptotically smaller than the expected number of all steps. We will establish a factor $O(n\Delta)$ improvement (recall Δ is the maximum degree).
2. *Sampling* Sampling an effective step is harder than sampling a normal step. Thus it is not clear that considering effective steps leads to a faster algorithm. We consider the problem of efficiently sampling an effective step in a later section, see Section 5. We show that sampling an effective step can be done in $O(\Delta)$ time (after $O(m)$ preprocessing).

Notation. For a type function f, let $\Gamma_v(f)$ be the subset of successors of v, such that $u \in \Gamma_v(f)$ iff $f(v) \neq f(u)$. Also, let $W'(f) = \sum_u w(f(u)) \cdot \frac{|\Gamma_u(f)|}{\deg u}$.

Modified dynamic evolution step. Formally, we consider the following *modified dynamic evolution step* (that changes the type function from f to f' and assumes that f does not map all nodes to the same type):

1. First a node v is picked at random with probability proportional to $p(v) = w(f(v)) \cdot \frac{|\Gamma_v(f)|}{\deg v}$ i.e. each node v has probability of being picked equal to $\frac{p(v)}{W'(f)}$.
2. Next, a successor u of v is picked uniformly at random among $\Gamma_v(f)$.
3. The type of u is then changed to $f(v)$, i.e., $f' = f[u \to f(v)]$.

In the following lemma we show that the modified dynamic evolution step corresponds to the dynamic evolution step except for discarding steps in which no change was made.

▶ **Lemma 1.** *Fix any type function f such that neither type has fixated. Let f_d (resp., f_m) be the next type function under dynamic evolution step (resp., modified dynamic evolution step). Then, $\Pr[f \neq f_d] > 0$ and for all type functions f' we have: $\Pr[f' = f_d \mid f \neq f_d] = \Pr[f' = f_m]$.*

Potential function ψ. Similar to [5] we consider the *potential function* $\psi = \sum_{v \in V_{t_1,f}} \frac{1}{\deg v}$ (recall that $V_{t_1,f}$ is the set of nodes of type t_1). We now lower bound the expected difference in potential per modified evolutionary step.

▶ **Lemma 2.** *Let f be a type function such that neither type has fixated. Apply a modified dynamic evolution step on f to obtain f'. Then,*

$$\mathbb{E}[\psi(f') - \psi(f)] \geq \frac{r-1}{\Delta \cdot (r+1)} \quad .$$

Proof. Observe that f differs from f' for exactly one node u. More precisely, let v be the node picked in line 1 of the modified dynamic evolution step and let u be the node picked in line 2. Then, $f' = f[u \to f(v)]$. The probability to select v is $\frac{p(v)}{W'(f)}$. The probability to then pick u is $\frac{1}{|\Gamma_v(f)|}$.

We have that

- If $f(u) = t_2$ (and thus, since it got picked $f(v) = t_1$), then $\psi(f') - \psi(f) = \frac{1}{\deg u}$.
- If $f(u) = t_1$ (and thus, since it got picked $f(v) = t_2$), then $\psi(f') - \psi(f) = -\frac{1}{\deg u}$.

Below we use the following notations:

$$E_{12} = \{(v,u) \in E \mid f(v) = t_1 \text{ and } f(u) = t_2\}; E_{21} = \{(v,u) \in E \mid f(v) = t_2 \text{ and } f(u) = t_1\}.$$

Thus,

$$\begin{aligned}
&\mathbb{E}[\psi(f') - \psi(f)] = \\
&\sum_{(v,u)\in E_{12}} \left(\frac{p(v)}{W'(f)} \cdot \frac{1}{|\Gamma_v(f)|} \cdot \frac{1}{\deg u}\right) - \sum_{(v,u)\in E_{21}} \left(\frac{p(v)}{W'(f)} \cdot \frac{1}{|\Gamma_v(f)|} \cdot \frac{1}{\deg u}\right) \\
&= \sum_{(v,u)\in E_{12}} \left(\frac{w(f(v))}{W'(f) \cdot (\deg u) \cdot (\deg v)}\right) - \sum_{(v,u)\in E_{21}} \left(\frac{w(f(v))}{W'(f) \cdot (\deg u) \cdot (\deg v)}\right) \ .
\end{aligned}$$

Using that the graph is undirected we get,

$$\begin{aligned}
\mathbb{E}[\psi(f) - \psi(f')] &= \sum_{(v,u)\in E_{12}} \left(\frac{w(f(v)) - w(f(u))}{W'(f) \cdot (\deg u) \cdot (\deg v)}\right) \\
&= \frac{1}{W'(f)} \sum_{(v,u)\in E_{12}} \left(\frac{r-1}{\min(\deg u, \deg v) \cdot \max(\deg u, \deg v)}\right) \\
&\geq \frac{r-1}{\Delta \cdot W'(f)} \sum_{(v,u)\in E_{12}} \frac{1}{\min(\deg u, \deg v)} = \frac{r-1}{\Delta \cdot W'(f)} \cdot S \ ,
\end{aligned}$$

where $S = \sum_{(v,u)\in E_{12}} \frac{1}{\min(\deg u, \deg v)}$. Note that in the second equality we use that for two numbers a, b, their product is equal to $\min(a,b) \cdot \max(a,b)$. By definition of $W'(f)$, we have

$$\begin{aligned}
W'(f) &= \sum_u w(f(u)) \cdot \frac{|\Gamma_u(f)|}{\deg u} = \sum_u \sum_{v\in\Gamma_u(f)} \frac{w(f(u))}{\deg u} = \sum_{\substack{(v,u)\in E \\ f(u)\neq f(v)}} \frac{w(f(u))}{\deg u} \\
&= \sum_{(v,u)\in E_{12}} \left(\frac{w(f(u))}{\deg u} + \frac{w(f(v))}{\deg v}\right) \leq \sum_{(v,u)\in E_{12}} \frac{w(f(u)) + w(f(v))}{\min(\deg u, \deg v)} = (r+1) \cdot S \ .
\end{aligned}$$

Thus, we see that $\mathbb{E}[\psi(f') - \psi(f)] \geq \frac{r-1}{\Delta\cdot(r+1)}$, as desired. This completes the proof. ◀

▶ **Lemma 3.** *Let $r = x\Delta$ for some number $x > 0$. Let f be a type function such that neither type has fixated. Apply a modified dynamic evolution step on f to obtain f'. The probability that $|V_{t_1,f'}| = |V_{t_1,f}| + 1$ is at least $\frac{x}{x+1}$ (otherwise, $|V_{t_1,f'}| = |V_{t_1,f}| - 1$).*

▶ **Lemma 4.** *Consider an upper bound ℓ, for each starting type function, on the expected number of (effective) steps to fixation. Then for any starting type function the probability that fixation requires more than $2 \cdot \ell \cdot x$ (effective) steps is at most 2^{-x}.*

We now present the main theorem of this section, which we obtain using the above lemmas, and techniques from [5].

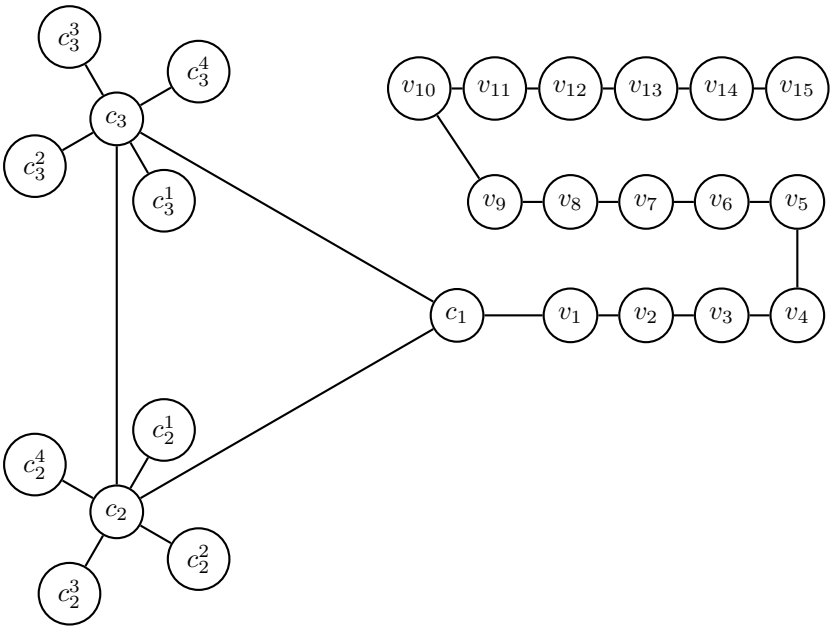


Figure 1 Example of a member of the family that attains the lower bound for undirected graphs. (Specifically, it is $G^{6,31}$)

▶ **Theorem 5.** *Let t_1 and t_2 be the two types, such that $r = w(t_1) > w(t_2) = 1$. Let Δ be the maximum degree. Let k be the number of nodes of type t_2 in the initial type function. The following assertions hold:*

- Bounds dependent on r
  1. Expected steps *The process requires at most $3k\Delta/\min(r-1,1)$ effective steps in expectation, before fixation is reached.*
  2. Probability *For any integer $x \geq 1$, after $6xn\Delta/\min(r-1,1)$ effective steps, the probability that the process has not fixated is at most 2^{-x}, irrespective of the initial type function.*
- Bounds independent on r
  1. Expected steps *The process requires at most $2nk\Delta^2$ effective steps in expectation, before fixation is reached.*
  2. Probability *For any integer $x \geq 1$, after $4xn^2\Delta^2$ effective steps, the probability that the process has not fixated is at most 2^{-x}, irrespective of the initial type function.*
- Bounds for $r \geq 2\Delta$
  1. Expected steps *The process requires at most $3k$ effective steps in expectation, before fixation is reached.*
  2. Probability *For any integer $x \geq 1$, after $6xn$ effective steps, the probability that the process has not fixated is at most 2^{-x}, irrespective of the initial type function.*

4 Lower bound for undirected graphs

In this section, we will argue that our bound on the expected number of effective steps is essentially tight, for fixed r.

We construct our lower bound graph $G^{\Delta,n}$, for given Δ, n (sufficiently large), but fixed $r > 1$, as follows. We will argue that fixation of $G^{\Delta,n}$ takes $\Omega(k\Delta)$ effective steps, if there are initially exactly k members of type t_2. For simplicity, we consider $\Delta > 2$ and $n > 4\Delta$ (it is

easy to see using similar techniques that for lines, where $\Delta = 2$, the expected fixation time is $\Omega(k)$ - basically because t_1 is going to fixate with pr. $\approx 1 - 1/r$, using a proof like Lemma 6, and converting the k nodes of type t_2 takes at least k efficient steps). There are two parts to the graph: A line of $\approx n/2$ nodes and a stars-on-a-cycle graph of $\approx n/2$. There is 1 edge from the one of the stars in the stars-on-a-cycle graph to the line. More formally, the graph is as follows: Let $x := \lfloor n/(2\Delta - 2) \rfloor$. There are nodes $V_C = \{c_1, \ldots, c_x\}$, such that c_i is connected to c_{i-1} and c_{i+1} for $1 < i < x$. Also, c_1 is connected to c_x. The nodes V_C are the centers of the stars in the stars-on-a-cycle graph. For each i, such that $2 \leq i \leq x$, the node c_i is connected to a set of leaves $V_C^i = \{c_i^1, \ldots, c_i^{\Delta-2}\}$. The set $V_C \cup \bigcup_{i=2}^{x} V_C^i$ forms the stars-on-a-cycle graph. Note that c_1 is only connected to c_2 and c_x in the stars-on-a-cycle graph. We have that the stars-on-a-cycle graph consists of $s = (x-1) \cdot (\Delta - 1) + 1 \approx n/2$ nodes. There are also nodes $V_L = \{\ell_1, \ldots, \ell_{n-s}\}$, such that node ℓ_i is connected to ℓ_{i-1} and ℓ_{i+1} for $1 < i < n/2$. The nodes V_L forms the line and consists of $n - s \geq n/2$ nodes. The node c_1 is connected to ℓ_1. There is an illustration of $G^{6,31}$ in Figure 1.

We first argue that if at least one of $V_L' = \{\ell_{\lceil n/4 \rceil}, \ldots, \ell_{n-s}\}$ is initially of type t_1, then with pr. lower bounded by a number depending only on r, type t_1 fixates (note that $|V_L'| \geq n/4$ and thus, even if there is only a single node of type t_1 initially placed uniformly at random, it is in V_L' with pr. $\geq 1/4$).

▶ **Lemma 6.** *With pr. above $\frac{1-1/r}{2}$ if at least one of V_L' is initially of type t_1, then t_1 fixates.*

The proof is based on applying the gambler's ruin twice. Once to find out that the pr. that V_L eventually becomes all t_1 is above $\frac{1-1/r}{2}$ (it is nearly $1 - 1/r$ in fact) and once to find out that if V_L is at some point all t_1, then the pr. that t_2 fixates is exponentially small with base r and exponent $n - s$. See the full version [2] for the proof.

Whenever a node of V_C^i, for some i, changes type, we say that a *leaf-step* occurred. We will next consider the pr. that an effective step is a leaf-step.

▶ **Lemma 7.** *The pr. that an effective step is a leaf-step is at most $\frac{r}{\Delta}$.*

The proof is quite direct and considers that the probability that a leaf gets selected for reproduction over a center node in the stars-on-a-cycle graph. See the full version [2] for the proof.

We are now ready for the theorem.

▶ **Theorem 8.** *Let $r > 1$ be some fixed constant. Consider $\Delta > 2$ (the maximum degree of the graph), $n > 4\Delta$ (sufficiently big), and some k such that $0 < k < n$. Then, if there are initially k members of type t_2 placed uniformly at random, the expected fixation time of $G^{\Delta,n}$ is above $\frac{k\Delta(1-1/r)}{32r}$ effective steps.*

Proof. Even if $k = n - 1$, we have that with pr. at least $\frac{1}{4}$, the lone node of type t_1 is initially in V_L'. If so, by Lemma 6, type t_1 is going to fixate with pr. at least $\frac{1-1/r}{2}$. Note that even for $\Delta = 3$, at least $\frac{n}{4}$ nodes of the graphs are in $V' := \bigcup_{i=2}^{x} V_C^i$ (i.e. the leaves of the stars-on-a-cycle graph). In expectation $\frac{k}{4}$ nodes of V' are thus initially of type t_2. For fixation for t_1 to occur, we must thus make that many leaf-steps. Any effective step is a leaf-step with pr. at most $\frac{r}{\Delta}$ by Lemma 7. Hence, with pr. $\frac{1}{4} \cdot \frac{1-1/r}{2}$ ($\frac{1}{4}$ is the probability that at least one node of type t_1 is in V_L' and $\frac{1-1/r}{2}$ is a lower bound on the fixation probability if a node of V_L' is of type t_1) we must make $\frac{k\Delta}{4r}$ effective steps before fixation in expectation, implying that the expected fixation time is at least $\frac{k\Delta(1-1/r)}{32r}$ effective steps. ◀

5 Sampling an effective step

In this section, we consider the problem of sampling an effective step. It is quite straightforward to do so in $O(m)$ time. We will present a data-structure that after $O(m)$ preprocessing can sample and update the distribution in $O(\Delta)$ time. For this result we assume that a uniformly random number can be selected between 0 and x for any number $x \leq n \cdot w(t)$ in constant time, a model that was also implicitly assumed in previous works [5][1].

▶ Remark. If we consider a weaker model, that requires constant time for each random bit, then we need $O(\log n)$ random bits in expectation and additional $O(\Delta)$ amortized time, using a similar data-structure (i.e., a total of $O(\Delta + \log n)$ amortized time in expectation). The argument for the weaker model is presented in the full version [2]. In this more restrictive model [5] would use $O(\log n)$ time per step for sampling.

Sketch of data-structure. We first sketch a list data-structure that supports (1) inserting elements; (2) removing elements; and (3) finding a random element; such that each operation takes (amortized or expected) $O(1)$ time. The idea based on dynamic arrays is as follows:

1. **Insertion** Inserting elements takes $O(1)$ amortized time in a dynamic array, using the standard construction.
2. **Deletion** Deleting elements is handled by changing the corresponding element to a null-value and then rebuilding the array, without the null-values, if more than half the elements have been deleted since the last rebuild. Again, this takes $O(1)$ amortized time.
3. **Find random element** Repeatedly pick a uniformly random entry. If it is not null, then output it. Since the array is at least half full, this takes in expectation at most 2 attempts and thus expected $O(1)$ time.

At all times we keep a doubly linked list of empty slots, to find a slot for insertion in $O(1)$ time.

Data-structure. The idea is then as follows. We have 2Δ such list data-structures, one for each pair of type and degree. We also have a weight associated to each list, which is the sum of the weight of all nodes in the list, according to the modified dynamic evolution step. When the current type function is f, we represent each node v as follows: The corresponding list data-structure contains $|\Gamma_v(f)|$ copies of v (and v keeps track of the locations in a doubly linked list). Each node v also keeps track of $\Gamma_v(f)$, using another list data-structure. It is easy to construct the initial data-structure in $O(m)$ time (note: $\sum_v |\Gamma_v(f)| \leq 2m$).

Updating the data-structure. We can then update the data-structure when the current type function f changes to $f[u \to t]$ (all updates have that form for some t and u), by removing u from the list data-structure $(f(u), \deg u)$ containing it and adding it to $(t, \deg u)$. Note that if we removed x' copies of u from $(f(u), \deg u)$ we add $\deg u - x'$ to $(t, \deg u)$. Also, we update each neighbor v of u (by deleting or adding a copy to $(f(v), \deg v)$, depending on whether $f(v) = t$). We also keep the weight corresponding to each list updated and $\Gamma_v(f)$ for all nodes v. This takes at most 4Δ data-structure insertions or deletions, and thus $O(\Delta)$ amortized time in total.

[1] The construction of [5] was to store a list for t_1 and a list for t_2 and then first decide if a t_1 or t_2 node would be selected in this step (based on r and the number of nodes of the different types) and then pick a random such node. This works when all nodes of a type has the same weight but does not generalize to the case when each node can have a distinct weight based on the nodes successors like here

Sampling an effective step. Let f be the current type function. First, pick a random list L among the 2Δ lists, proportional to their weight. Then pick a random node v from L. Then pick a node at random in $\Gamma_v(f)$. This takes $O(\Delta)$ time in expectation.

▶ Remark. Observe that picking a random list among the 2Δ lists, proportional to their weight takes $O(\Delta)$ time to do naively: E.g. consider some ordering of the lists and let w_i be the total weight of list i (we keep this updated so it can be found in constant time). Pick a random number x between 1 and the total weight of all the lists (assumed to be doable in constant time). Iterate over the lists in order and when looking at list i, check if $x < \sum_{j=1}^{i} w_j$. If so, pick list i, otherwise continue to list $i+1$. By making a binary, balanced tree over the lists (similar to what is used for the more restrictive model, see the full version [2]), the time can be brought down to $O(\log \Delta)$ for this step - however the naive approach suffices for our application, because updates requires $O(\Delta)$ time.

This leads to the following theorem.

▶ **Theorem 9.** *An effective step can be sampled in (amortized and expected) $O(\Delta)$ time after $O(m)$ preprocessing, if a uniformly random integer between 0 and x, for any $0 < x \leq n \cdot w(t)$, can be found in constant time.*

6 Algorithms for approximating fixation probability

We present the algorithms for solving the fixation, extinction, and generalized fixation problems.

The Meta-simulation algorithm. Similar to [5], the algorithms are instantiating the following meta-simulation algorithm, that takes a distribution over initial type functions $\mathcal{D}$, type t and natural numbers u and z as input:

Function MetaSimulation(t,z,u,$\mathcal{D}$)

```
Let y ← 0;
for (i ∈ {1, ..., z}) do
    Initialize a new simulation I with initial type function f picked according to D;
    Let j ← 0;
    while (I has not fixated) do
        if (j ≥ u) then
            return Simulation took too long;
        Set j ← j + 1;
        Simulate an effective step in I;
    if (t fixated in I) then
        Set y ← y + 1;
return y/z;
```

Basic principle of simulation. Note that the meta-simulation algorithm uses $O(uz\Delta)$ time (by Theorem 9). In essence, the algorithm runs z simulations of the process and terminates with "Simulation took too long" iff some simulation took over u steps. Hence, whenever the algorithm returns a number it is the mean of z binary random variables, each equal to 1 with

probability $\Pr[\mathcal{F}_t \mid \mathcal{E}_u]$, where $\mathcal{F}_t$ is the event that t fixates and $\mathcal{E}_u$ is the event that fixation happens before u effective steps, when the initial type function is picked according to $\mathcal{D}$ (we note that the conditional part was overlooked in [5], moreover, instead of steps we consider only effective steps). By ensuring that u is high enough and that the approximation is tight enough (basically, that z is high enough), we can use $\Pr[\mathcal{F}_t \mid \mathcal{E}_u]$ as an approximation of $\Pr[\mathcal{F}_t]$, as shown in the following lemma.

▶ **Lemma 10.** *Let $0 < \epsilon < 1$ be given. Let $\mathcal{X}, \mathcal{E}$ be a pair of events and x a number, such that $\Pr[\mathcal{E}] \geq 1 - \frac{\epsilon \cdot \Pr[\mathcal{X}]}{4}$ and that $x \in [(1-\epsilon/2)\Pr[\mathcal{X} \mid \mathcal{E}], (1+\epsilon/2)\Pr[\mathcal{X} \mid \mathcal{E}]]$. Then*

$$x \in [(1-\epsilon)\cdot \Pr[\mathcal{X}], (1+\epsilon)\cdot \Pr[\mathcal{X}]] \ .$$

The value of u: $u_{z,r}$. Consider some fixed value of z. The value of u is basically just picked so high that $\Pr[\mathcal{E}_u] \geq 1 - \frac{\epsilon \cdot \Pr[\mathcal{F}_t]}{4}$ (so that we can apply Lemma 10) and such that after taking union bound over the z trials, we have less than some constant probability of stopping. The right value of u is thus sensitive to r, but in all cases at most $O(n^2\Delta^2 \max(\log z, \log \epsilon^{-1}))$, because of Theorem 5. More precisely, we let

$$u_{z,r} = \begin{cases} 30n \cdot \max(\log z, \log \epsilon^{-1}) & \text{if } r \geq 2\Delta \\ \frac{30n\Delta}{\min(r-1,1)} \cdot \max(\log z, \log \epsilon^{-1}) & \text{if } 1 + \frac{1}{n\cdot\Delta} \leq r < 2\Delta \\ 20n^2\Delta^2 \cdot \max(\log z, \log \epsilon^{-1}) & \text{if } r < 1 + \frac{1}{n\cdot\Delta} \end{cases} .$$

Algorithm Algo1. We consider the fixation problem for t_1. Algorithm Algo1 is as follows:
1. Let $\mathcal{D}$ be the uniform distribution over the n type functions where exactly one node is t_1.
2. Return MetaSimulation(t_1,z,$u_{z,r}$,$\mathcal{D}$), for $z = 48 \cdot \frac{n}{\epsilon^2}$.

Algorithm Algo2. We consider the extinction problem for t_1. Algorithm Algo2 is as follows:
1. Let $\mathcal{D}$ be the uniform distribution over the n type functions where exactly one node is t_2.
2. Return MetaSimulation(t_1,z,$u_{z,r}$,$\mathcal{D}$), for $z = 24/\epsilon^2$.

Algorithm Algo3. We consider the problem of (additively) approximating the fixation probability given some type function f and type t. Algorithm Algo3 is as follows:
1. Let $\mathcal{D}$ be the distribution that assigns 1 to f.
2. Return MetaSimulation(t,z,$u_{z,r}$,$\mathcal{D}$), for $z = 6/\epsilon^2$.

▶ **Theorem 11.** *Let G be a connected undirected graph of n nodes with the highest degree Δ, divided into two types of nodes t_1, t_2, such that $r = w(t_1) > w(t_2) = 1$. Given $\frac{1}{2} > \epsilon > 0$, let $\alpha = n^2 \cdot \Delta \cdot \epsilon^{-2} \cdot \max(\log n, \log \epsilon^{-1})$ and $\beta = n \cdot \Delta \cdot \epsilon^{-2} \cdot \log \epsilon^{-1}$. Consider the running times:*

$$T(x) = \begin{cases} O(x) & \text{if } r \geq 2\Delta \\ O(\frac{x\cdot\Delta}{\min(r-1,1)}) & \text{if } 1 + \frac{1}{n\cdot\Delta} \leq r < 2\Delta \\ O(n \cdot \Delta^2 \cdot x) & \text{if } 1 < r < 1 + \frac{1}{n\cdot\Delta} \end{cases} .$$

- **Fixation (resp. Extinction) problem for t_1** *Algorithm* Algo1 *(resp.* Algo2*) is an FPRAS algorithm, with running time $T(\alpha)$ (resp. $T(\beta)$), that with probability at least $\frac{3}{4}$ outputs a number in $[(1-\epsilon)\cdot\rho, (1+\epsilon)\cdot\rho]$, where ρ is the solution of the fixation (resp. extinction) problem for t_1.*

- **Generalized fixation problem** *Given an initial type function f and a type t, there is an (additive approximation) algorithm,* Algo3*, with running time $T(\beta)$, that with probability at least $\frac{3}{4}$ outputs a number in $[\rho - \epsilon, \rho + \epsilon]$, where ρ is the solution of the generalized fixation problem given f and t.*

▶ Remark. There exists no known FPRAS for the generalized fixation problem and since the fixation probability might be exponentially small such an algorithm might not exist. (It is exponentially small for fixation of t_2, even in the Moran process (that is, when the graph is complete) when there initially is 1 node of type t_2)

Alternative algorithm for extinction for t_2. We also present an alternative algorithm for extinction for t_2 when r is big. This is completely different from the techniques of [5]. The alternative algorithm is based on the following result where we show for big r that $1/r$ is a good approximation of the extinction probability for t_2, and thus the algorithm is polynomial even for big r in binary.

▶ **Theorem 12.** *Consider an undirected graph G and consider the extinction problem for t_2 on G. If $r \geq \max(\Delta^2, n)/\epsilon$, then $\frac{1}{r} \in [(1-\epsilon)\cdot\rho, (1+\epsilon)\cdot\rho]$, where ρ is the solution of the extinction problem for t_2.*

Proof sketch. We present a proof sketch, and details are in the full version [2]. We have two cases:

- By [5, Lemma 4], we have $\rho \geq \frac{1}{n+r}$. Thus, $(1+\epsilon)\cdot\rho \geq \frac{1}{r}$, as desired, since $n/\epsilon \leq r$.
- On the other hand, the probability of fixation for t_2 in the first effective step is at most $\frac{1}{r+1} < \frac{1}{r}$ (we show this in the full version [2]). The probability that fixation happens for t_2 after the first effective step is at most ϵ/r because of the following reason: By Lemma 3, the probability of increasing the number of members of t_2 is at most $p := \frac{1}{r/\Delta+1}$ and otherwise it decrements. We then model the problem as a Markov chain M with state space corresponding to the number of members of t_1, using p as the probability to decrease the current state. In M the starting state is state 2 (after the first effective step, if fixation did not happen, then the number of members of t_1 is 2). Using that $\Delta^2/\epsilon \leq r$, we see that the probability of absorption in state 0 of M from state 2 is less than ϵ/r. Hence, ρ is at most $(1+\epsilon)/r$ and $(1-\epsilon)\rho$ is thus less than $1/r$. ◀

▶ Remark. While Theorem 12 is for undirected graphs, a variant (with larger r and which requires the computation of the pr. that t_1 goes extinct in the first step) can be established even for directed graphs, see the full version [2].

Concluding remarks. In this work we present faster Monte-Carlo algorithms for approximating fixation probability for undirected graphs (see the full version [2] for detailed comparison). An interesting open question is whether the fixation probability can be approximated in polynomial time for directed graphs.

References

1 B. Adlam, K. Chatterjee, and M. A. Nowak. Amplifiers of selection. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 471(2181), 2015. doi:10.1098/rspa.2015.0114.

2 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, and Martin Nowak. Faster monte-carlo algorithms for fixation probability of the moran process on undirected graphs. *CoRR*, abs/1706.06931, 2017. URL: http://arxiv.org/abs/1706.06931.

3 F. Débarre, C. Hauert, and M. Doebeli. Social evolution in structured populations. *Nature Communications*, 2014.

4 Josep Díaz, Leslie Ann Goldberg, George B. Mertzios, David Richerby, Maria Serna, and Paul G. Spirakis. On the fixation probability of superstars. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 469(2156), 2013.

5 Josep Díaz, Leslie Ann Goldberg, George B. Mertzios, David Richerby, Maria Serna, and Paul G. Spirakis. Approximating Fixation Probabilities in the Generalized Moran Process. *Algorithmica*, 69(1):78–91, 2014 (Conference version SODA 2012).

6 Josep Díaz, Leslie Ann Goldberg, David Richerby, and Maria Serna. Absorption time of the Moran process. *Random Structures & Algorithms*, 48(1):137–159, 2016.

7 W.J. Ewens. *Mathematical Population Genetics 1: I. Theoretical Introduction*. Interdisciplinary Applied Mathematics. Springer, 2004.

8 Marcus Frean, Paul B. Rainey, and Arne Traulsen. The effect of population structure on the rate of evolution. *Proceedings of the Royal Society B: Biological Sciences*, 280(1762), 2013.

9 Andreas Galanis, Andreas Göbel, Leslie Ann Goldberg, John Lapinskas, and David Richerby. Amplifiers for the Moran Process. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55, pages 62:1–62:13, 2016.

10 Rasmus Ibsen-Jensen, Krishnendu Chatterjee, and Martin A Nowak. Computational complexity of ecological and evolutionary spatial dynamics. *Proceedings of the National Academy of Sciences*, 112(51):15636–15641, 2015.

11 Samuel Karlin and Howard M. Taylor. *A First Course in Stochastic Processes, Second Edition*. Academic Press, 2 edition, April 1975.

12 Erez Lieberman, Christoph Hauert, and Martin A. Nowak. Evolutionary dynamics on graphs. *Nature*, 433(7023):312–316, January 2005. doi:10.1038/nature03204.

13 P. A. P. Moran. *The Statistical Processes of Evolutionary Theory*. Oxford University Press, Oxford, 1962.

14 Martin A. Nowak. *Evolutionary Dynamics: Exploring the Equations of Life*. Harvard University Press, 2006.

15 Paulo Shakarian, Patrick Roos, and Anthony Johnson. A review of evolutionary graph theory with applications to game theory. *Biosystems*, 107(2):66–80, 2012.

The 2CNF Boolean Formula Satisfiability Problem and the Linear Space Hypothesis*

Tomoyuki Yamakami

Faculty of Engineering, University of Fukui, Japan
TomoyukiYamakami@gmail.com

Abstract

We aim at investigating the solvability/insolvability of nondeterministic logarithmic-space (NL) decision, search, and optimization problems parameterized by size parameters using simultaneously polynomial time and sub-linear space on multi-tape deterministic Turing machines. We are particularly focused on a special NL-complete problem, 2SAT – the 2CNF Boolean formula satisfiability problem – parameterized by the number of Boolean variables. It is shown that 2SAT with n variables and m clauses can be solved simultaneously polynomial time and $(n/2^{c\sqrt{\log n}})\,polylog(m+n)$ space for an absolute constant $c > 0$. This fact inspires us to propose a new, practical working hypothesis, called the linear space hypothesis (LSH), which states that $2SAT_3$ – a restricted variant of 2SAT in which each variable of a given 2CNF formula appears as literals in at most 3 clauses – cannot be solved simultaneously in polynomial time using strictly "sub-linear" (i.e., $n^{\varepsilon}\,polylog(n)$ for a certain constant $\varepsilon \in (0,1)$) space. An immediate consequence of this working hypothesis is $\mathrm{L} \neq \mathrm{NL}$. Moreover, we use our hypothesis as a plausible basis to lead to the insolvability of various NL search problems as well as the nonapproximability of NL optimization problems. For our investigation, since standard logarithmic-space reductions may no longer preserve polynomial-time sub-linear-space complexity, we need to introduce a new, practical notion of "short reduction." It turns out that $\overline{2SAT_3}$ is complete for a restricted version of NL, called Syntactic NL or simply SNL, under such short reductions. This fact supports the legitimacy of our working hypothesis.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes, F.4.3 Formal Languages, G.1.2 Approximation, G.1.6 Optimization

Keywords and phrases sub-linear space, linear space hypothesis, short reduction, Boolean formula satisfiability problem, NL search, NL optimization, Syntactic NL

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.62

1 Background and Main Contributions

1.1 Motivational Discussion: Space Complexity of Parameterized 2SAT

Since Cook [4] demonstrated its NP-completeness, the Boolean formula satisfiability problem (SAT) of determining whether a given Boolean formula is satisfied by a suitably-chosen variable assignment has been studied extensively for more than 50 years. As its restricted variant, the *kCNF Boolean formula satisfiability problem* (kSAT), for an integer index $k \geq 3$, whose input formulas are of k-conjunctive normal form (kCNF) has also been a centerpiece of computational complexity theory. Since kSAT is complete for NP (nondeterministic

* This work was done in part at the University of Toronto between August 1, 2016 and March 30, 2017 and was supported by the Natural Sciences and Engineering Research Council of Canada.

polynomial time) [4], its solvability is linked to the computational complexity of all other NP problems; for instance, if kSAT is solved in polynomial time, then so are all NP problems. A recent study has been focused on the solvability of kSAT with n Boolean variables and m clauses within "sub-exponential" (which means $2^{\varepsilon n}poly(n+m)$ for an absolute constant $\varepsilon \in (0,1)$ and a suitable polynomial $poly(\cdot)$) runtime. In this line of study, Impagliazzo, Paturi, and Zane [10] took a new approach toward kSAT and its search version, Search-kSAT, parameterized by the number $m_{vbl}(x)$ of Boolean variables and the number $m_{cls}(x)$ of clauses in a given kCNF formula x as natural "size parameters" (which were called "complexity parameters" in [10]). To discuss such sub-exponential-time solvability for a wide range of NP-complete problems, Impagliazzo et al. further devised a crucial notion of *sub-exponential-time reduction family (or SERF-reduction)*, which preserves the sub-exponential-time complexity, and they cleverly demonstrated that the two size parameters, $m_{vbl}(x)$ and $m_{cls}(x)$, make Search-kSAT SERF-equivalent (that is, the both are SERF-reducible to each other). As a working hypothesis, Impagliazzo and Paturi [9] formally proposed the *exponential time hypothesis* (ETH), which asserts the insolvability of kSAT parameterized by $m_{vbl}(x)$ (succinctly denoted by $(k\mathrm{SAT}, m_{vbl})$) in sub-exponential time for all indices $k \geq 3$. Their hypothesis is obviously a stronger assertion than P $\neq$ NP and it has then led to intriguing consequences, including finer lower bounds on the solvability of various parameterized NP problems (see, e.g., a survey [14]).

Whereas ETH concerns with kSAT for $k \geq 3$, we are focused on the remaining case of $k = 2$. The decision problem 2SAT is known to be complete* for NL (nondeterministic logarithmic space) under log-space reductions. Since 2SAT already enjoys a polynomial-time algorithm (because NL $\subseteq$ P), we are more concerned with how much memory space such an algorithm requires to run. An elaborate algorithm solves 2SAT with n variables and m clauses using simultaneously polynomial time and $(n/2^{c\sqrt{\log n}})\,polylog(m+n)$ space (Theorem 4.2), where $c > 0$ is a constant and $polylog(\cdot)$ is a suitable polylogarithmic function. This space bound is slightly below n; however, it is not yet known that 2SAT parameterized by $m_{vbl}(x)$ (or $m_{cls}(x)$) can be solved in polynomial time using strictly "sub-linear" space. Here, the informal term "sub-linear" for a size parameter $m(x)$ refers to a function of the form $m(x)^{\varepsilon}\ell(|x|)$ on input instances x for a certain absolute constant $\varepsilon \in (0,1)$ and an appropriately-chosen polylogarithmic function $\ell(n)$. Of course, this multiplicative factor $\ell(|x|)$ becomes redundant if $m(x)$ is relatively large (for example, $m(x) \geq \log^k |x|$ for any constant $k > 0$) and thus "sub-linear" turns out to be simply $m(x)^{\varepsilon}$.

In parallel to a restriction of SAT to kSAT, for polynomial-time sub-linear-space solvability, we further limit 2SAT to $2\mathrm{SAT}_k$, which consists of all satisfiable formulas in which each variable appears as literals in at most k clauses. Notice that $2\mathrm{SAT}_k$ for each $k \geq 3$ is also NL-complete (Proposition 4.1) as 2SAT is; in contrast, $k\mathrm{SAT}_2$ already falls into L for any index $k \geq 2$.

1.2 Sub-Linear Space and Short Reductions

All (parameterized) decision problems solvable in polynomial time using sub-linear space form a new complexity class PsubLIN (whose prefix "P" refers to "polynomial time"), which is located between L and P. This class PsubLIN naturally includes, for example, DCFL (deterministic context-free) because Cook [5] earlier showed that every language in DCFL is

* This is because Jones, Lien, and Laaser [13] demonstrated the NL-completeness of the complement of SAT (called UNSAT_2 in [13]) and Immerman [8] and Szelepcsényi [18] proved the closure of NL under complementation.

recognized in polynomial time using $O(\log^2 n)$-space, where n is input size. Unfortunately, there is no separation known among L, NL, PsubLIN, and P.

It turns out that PsubLIN does not seem to be closed under standard log-space reductions; thus, those reductions are no longer suitable tools to discuss the solvability of NL-complete problems in polynomial time using sub-linear space. Therefore, we need to introduce a much weaker form of reductions, called *short reductions*, which preserve polynomial-time, sub-linear-space complexity. Intuitively speaking, a short reduction is a reduction between two (parameterized) decision problems computed by a reduction machine (or a reduction function) that can generate strings of size parameter proportional to or less than size parameter of its input string. In particular, we will define three types of such short reductions in Section 3: short L-m-reducibility ($\leq_m^{\mathrm{sL}}$), short L-T-reducibility ($\leq_T^{\mathrm{sL}}$), and short sub-linear-space-T-reducibility ($\leq_T^{\mathrm{sSLRF}}$).

As noted earlier, Impagliazzo et al. demonstrated in [10, Corollary 2] that $(k\mathrm{SAT}, m_{vbl})$ is SERF-equivalent to $(k\mathrm{SAT}, m_{cls})$. Similarly, we can give a short reduction from $2\mathrm{SAT}_3$ with m_{vbl} to $2\mathrm{SAT}_3$ with m_{cls}, and vice verse; in other words, $(2\mathrm{SAT}_3, m_{vbl})$ and $(2\mathrm{SAT}_3, m_{cls})$ are equivalent under short L-T-reductions (Lemma 4.3(2)). On the contrary, such equivalence is not known for 2SAT and this circumstance signifies the importance of $2\mathrm{SAT}_3$.

Another importance of $2\mathrm{SAT}_3$ can be demonstrated by showing that $2\mathrm{SAT}_3$ is actually one of the hardest problems in a natural subclass of NL, which we call *Syntactic NL* or simply *SNL*. An *SNL formula* $\Psi \equiv \Psi(x)$ is of the form $\exists T \forall i_1 \cdots \forall i_r \forall y_1 \cdots \forall y_s \exists z_1 \cdots \exists z_t\ \psi$, starting with a second-order existential quantifier, followed by first-order quantifiers, with a supporting *semantic model*. From this model, we define a *certificate size* $m_{cert}(x)$. Their precise definitions will be given in Section 4. We say that Ψ *syntactically expresses* A if, for every x, $x \in A$ exactly when $\Psi(x)$ is true. The notation SNL stands for the collection of all (A, m), each of which is expressed syntactically by an appropriate SNL-formula Ψ and satisfies $m(x) = cm_{cert}(x)$ for a certain constant $c > 0$.

▶ **Theorem 1.1.** $(\overline{2\mathrm{SAT}_3}, m_{vbl})$ *is complete for* SNL *under short SLRF-T-reductions.*

1.3 A New, Practical Working Hypothesis for $2\mathrm{SAT}_3$

Since its introduction in 2001, ETH for kSAT ($k \geq 3$) has served as a driving force to obtain finer lower bounds on the sub-exponential-time computability of various parameterized NP problems, since those bounds do not seem to be obtained directly from the popular assumption of P $\neq$ NP. In a similar vein, we wish to propose a new working hypothesis, called the *linear space hypothesis* (LSH) for $2\mathrm{SAT}_3$, in which no deterministic algorithm solves $(2\mathrm{SAT}_3, m_{vbl})$ simultaneously in polynomial time using sub-linear space. More precisely:

> The Linear Space Hypothesis (LSH) for $2\mathrm{SAT}_3$: For any choice of $\varepsilon \in (0, 1)$ and any polylogarithmic function ℓ, no deterministic Turing machine solves $2\mathrm{SAT}_3$ parameterized by m_{vbl} simultaneously in polynomial time using $m_{vbl}(x)^{\varepsilon}\ell(|x|)$ space, where x refers to an input instance to $2\mathrm{SAT}_3$.

We can replace m_{vbl} in the above definition by m_{cls} (see Section 4), and thus we often omit it. Consider the case of L = NL. Since $2\mathrm{SAT}_3$ belongs to L, it is also in PsubLIN. This consequence contradicts LSH for $2\mathrm{SAT}_3$. Therefore, we immediately obtain:

▶ **Theorem 1.2.** *If LSH for* $2\mathrm{SAT}_3$ *is true, then* L $\neq$ NL.

From Theorem 1.2, our working hypothesis LSH for $2\mathrm{SAT}_3$ is expected to lead to finer, better consequences than what the assumption L $\neq$ NL can lead to.

Let δ_3 denote the infimum of a real number $\varepsilon \in [0,1]$ for which there is a deterministic Turing machine solving $2\mathrm{SAT}_3$ simultaneously in polynomial time using at most $m_{vbl}(x)^{\varepsilon}\ell(|x|)$ space on instances x for a certain fixed polylogarithmic function ℓ. Here, we acknowledge three possible cases: (i) $\delta_3 = 0$, (ii) $0 < \delta_3 < 1$, and (iii) $\delta_3 = 1$, and one of them must be true after all. The hypothesis LSH for $2\mathrm{SAT}_3$ exactly matches (iii).

▶ **Proposition 1.3.** *The working hypothesis LSH for* $2\mathrm{SAT}_3$ *is true iff* $\delta_3 = 1$ *holds.*

For any $\leq_r$-reduction, the notation $\leq_r(\mathrm{SNL})$ refers to the collection of all (parameterized) decision problems that can be reduced by $\leq_r$-reductions to certain problems in SNL.

▶ **Proposition 1.4.** *The following statements are all logically equivalent. (1)* $(2\mathrm{SAT}_3, m_{vbl}) \in \mathrm{PsubLIN}$. *(2)* $\mathrm{SNL} \subseteq \mathrm{PsubLIN}$. *(3)* $\leq_T^{\mathrm{sLRF}}(\mathrm{SNL}) \subseteq \mathrm{PsubLIN}$.

Proposition 1.4(3) can be compared to the fact that $\leq_m^{\mathrm{L}}(\mathrm{SNL}) = \mathrm{NL}$.

Furthermore, we seek two other characterizations of the hypothesis LSH for $2\mathrm{SAT}_3$. The first problem is a variant of a well-known NP-complete problem, called the $\{0,1\}$-*linear programming problem* (LP_2). In what follows, a vector of dimension n means an $n \times 1$ matrix and a rational number is treated as a pair of appropriate integers.

$(2,k)$-Entry $\{0,1\}$-Linear Programming Problem ($\mathrm{LP}_{2,k}$):

- Instance: a rational $m \times n$ matrix A and a rational vector b of dimension n, where $m, n \geq 1$ and each row of A has at most two nonzero entries and each column of A has at most k non-zero entries.
- Question: is there any $\{0,1\}$-vector x satisfying $Ax \geq b$?

As natural size parameters $m_{col}(x)$ and $m_{row}(x)$, we take the numbers of columns and of rows of A for instance $x = (A, b)$ given to $\mathrm{LP}_{2,k}$, respectively.

Another problem to consider is a variant of the *directed s-t connectivity problem*[†] (DSTCON) of asking whether a path between two given vertices exists in a directed graph.

Degree-k Directed s-t Connectivity Problem (kDSTCON):

- Instance: a directed graph $G = (V, E)$ of degree (i.e., indegree plus outdegree) at most k, and two designated vertices s and t.
- Question: is there any path from s to t in G?

For any instance $x = (G, s, t)$ to kDSTCON, $m_{ver}(x)$ and $m_{edg}(x)$ respectively denote the number of vertices and that of edges in G.

▶ **Theorem 1.5.** *The following statements are logically equivalent: (1) LSH for* $2\mathrm{SAT}_3$*, (2) LSH for* $\mathrm{LP}_{2,3}$ *(with* m_{row} *or* m_{col}*), and (3) LSH for* 3DSTCON *(with* m_{ver} *or* m_{edg}*).*

This theorem allows us to use $\mathrm{LP}_{2,3}$ and 3DSTCON for LSH as substitutes for $2\mathrm{SAT}_3$.

1.4 Four Examples of How to Apply the Working Hypothesis

To demonstrate the usefulness of LSH for $2\mathrm{SAT}_3$, we will seek four applications of LSH in the fields of search problems and optimization problems. Although many NL decision problems have been turned into *NL search problems* (whose precise definition is given in Section 6), not all NL problems can be "straightforwardly" converted into a framework of NL search problems. For example, 2SAT is NL-complete but the problem of finding a truth assignment (when

[†] This is also known as the graph accessibility problem and the graph reachability problem in the literature.

variables are ordered in an arbitrarily fixed way) that satisfies a given 2CNF formula does not look like a legitimate form of NL search problem. In addition, its optimization version, Max2SAT, is already complete for APX (polynomial-time approximable NP optimization) instead of NLO (NL optimization class) under polynomial-time approximation-preserving reductions (see [1]).

First, we will see two simple applications of LSH for $2SAT_3$ in the area of NL search problems. Earlier, Jones et al. [13] discussed the NL-completeness of a decision problem concerning *one-way nondeterministic finite automata* (or 1nfa's). We modify this problem into an associated search problem, called Search-1NFA, as given below.

1NFA Membership Search Problem (Search-1NFA):
- Instance: a 1nfa $M = (Q, \Sigma, \delta, q_0, F)$ with no λ-moves, and a parameter 1^n, where λ is the empty string for $n \in \mathbb{N}$.
- Solution: an input string x of length n accepted by M (i.e., when x is written on M's read-only input tape, M eventually enters a final state in F before or on reading the last symbol of x).

As a meaningful size parameter m_{nfa}, we set $m_{nfa}(x) = |Q||\Sigma|n$ for instance $x = (M, 1^n)$.

▶ **Theorem 1.6.** *Assuming that LSH for* $2SAT_3$*, for every fixed value* $\varepsilon \in (0, 1/2)$*, there is no polynomial-time* $O(n^{1/2-\varepsilon})$*-space algorithm for* (Search-1NFA, m_{nfa}).

Jenner [11] presented a few variants of the well-known *knapsack problem* and showed their NL-completeness. Here, we choose one of them that fit into the NL-search framework by a small modification. Given a string x, a substring z of x is called *unique* if there exists a unique pair u, v satisfying $x = uzv$. Write $[n]$ for the set $\{1, 2, \ldots, n\}$.

Unique Ordered Concatenation Knapsack Search Problem (Search-UOCK):
- Instance: a string w and a sequence $(w_1, w_2, \ldots, w_n)$ of strings over a certain fixed alphabet Σ such that, for every $i \in [n]$, if w_i is a substring of w, then w_i is unique.
- Solution: a sequence $(i_1, i_2, \ldots, i_k)$ of indices with $k \geq 1$ such that $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ and $w = w_{i_1} w_{i_2} \cdots w_{i_k}$.

Our size parameter m_{elm} for Search-UOCK is the number of elements $w_1, w_2, \ldots, w_n$ in the above definition (namely, $m_{elm}(x) = n$ for instance x).

▶ **Theorem 1.7.** *If LSH for* $2SAT_3$ *holds, then, for any* $\varepsilon > 0$*, there is no polynomial-time* $O(n^{1/2-\varepsilon})$*-space algorithm for* (Search-UOCK, m_{elm}).

We then turn to the area of *NL optimization problems* (or *NLO problems*, in short) [19, 20]. See Section 6 for their formal definition. We will consider a problem that belongs to $LSAS_{NLO}$ but does not seem to be solvable using log space. Here, $LSAS_{NLO}$ is the collection of NLO problems that have log-space approximation schemes, where a *log-space approximation scheme* for an NLO problem P is a deterministic Turing machine M that takes any input of the form (x, k) and outputs a solution y of P using space at most $f(k) \log |x|$ for a certain log-space computable function $f : \mathbb{N} \to \mathbb{N}$ for which the performance ratio R satisfies $R(x, y) \leq 1 + \frac{1}{k}$. Such a solution y is called a $(1 + \frac{1}{k})$*-approximate solution*. Notice that the *performance ratio* is a ratio between the value of an optimal solution and that of M's output.

In 2007, Tantau [19] presented an NL maximization problem, called Max-HPP, which falls into $LSAS_{NLO}$. This problem was later rephrased in [20, arXiv version] in terms of complete graphs and it was shown to be computationally hard for LO_{NLO} (log-space computable NL optimization) under *approximation-preserving exact* NC^1*-reduction*.

Maximum Hot Potato Problem (Max-HPP):

- Instance: an $n \times n$ matrix A whose entries are drawn from $[n]$, a number $d \in [n]$, and a start index $i_1 \in [n]$, where $n \in \mathbb{N}^+$.
- Solution: an index sequence $\mathcal{S} = (i_1, i_2, \ldots, i_d)$ of length d with $i_j \in [n]$ for any $j \in [d]$.
- Measure: total weight $w(\mathcal{S}) = \sum_{j=1}^{d-1} A_{i_j i_{j+1}}$.

We use the number n of columns in a given matrix as size parameter $m_{col}(A, d, i_1)$. We can show that, under the assumption of LSH for $2\mathrm{SAT}_3$, (Max-HPP, m_{col}) cannot have polynomial-time $O(k^{1/3} \log m_{col}(x))$-space approximation schemes of finding $(1 + \frac{1}{k})$-approximate solutions for instances x.

▶ **Theorem 1.8.** *If LSH for* $2\mathrm{SAT}_3$ *is true, then, for any* $\varepsilon > 0$*, there is no polynomial-time* $O(k^{1/3} \log m_{col}(x))$*-space algorithm finding* $(1+\frac{1}{k})$*-approximate solutions of* (Max-HPP, m_{col})*, where* x *is any instance and* k *is an approximation parameter.*

The fourth example concerns with the computational complexity of transforming one type of finite automata into another type. It is known that we can convert a 1nfa M to an "equivalent" *one-way deterministic finite automaton* (or 1dfa) M' in the sense that both M and M' recognize exactly the same language. In particular, we consider the case of transforming an n-state *unary* 1nfa into its equivalent *unary* 1dfa, where a unary finite automaton takes a single-letter input alphabet. A standard procedure of such transformation requires polynomial-time and $O(n)$ space (cf. [7]). Under LSH for $2\mathrm{SAT}_3$, we can demonstrate that this space bound cannot be made significantly smaller.

▶ **Theorem 1.9.** *If LSH for* $2\mathrm{SAT}_3$ *is true, then, for any constant* $\varepsilon \in (0, 1)$*, there is no polynomial-time* $O(n^{\varepsilon})$*-space algorithm that takes an* n*-state unary 1nfa as input and produces an equivalent unary 1dfa of* $O(n \log n)$ *states.*

2 Basic Notions and Notation

Let $\mathbb{N}$ be the set of *natural numbers* (i.e., nonnegative integers) and set $\mathbb{N}^+ = \mathbb{N} - \{0\}$. Two notations $\mathbb{R}$ and $\mathbb{R}^{\geq 0}$ denote respectively the set of all *real numbers* and that of all *nonnegative real numbers*. For any two integers m and n with $m \leq n$, the notation $[m, n]_{\mathbb{Z}}$ denotes the set $\{m, m+1, m+2, \ldots, n\}$, which is an *integer interval* between m and n. For simplicity, when $n \geq 1$, we write $[n]$ for $[1, n]_{\mathbb{Z}}$.

In this paper, all *polynomials* are assumed to have nonnegative integer coefficients. All *logarithms* are to base 2. A *polylogarithmic (or polylog) function* ℓ is a function mapping $\mathbb{N}$ to $\mathbb{R}^{\geq 0}$ such that there exists a polynomial p for which $\ell(n) = p(\log n)$ holds for all $n \in \mathbb{N}$, provided that "$\log 0$" is conventionally set to be 0.

In a course of our study on polynomial-time sub-linear-space computability, it is convenient to expand the standard framework of decision problems to problems *parameterized* by properly chosen "size parameters" (called "complexity parameters" in [10]), which serve as a basis unit of the time/space complexity of an algorithm. In this respect, we follow a framework of Impagliazzo et al. [10] to work with a flexible choice of size parameter. A standard size parameter is the total length $|x|$ of the binary representation of an input instance x and it is often denoted by $||$. More generally, a *(log-space) size parameter* $m(x)$ for a problem P is a function mapping Σ^* (where Σ is an input alphabet) to $\mathbb{N}$ such that (1) m must be computed using log space (that is, by a certain Turing machine that takes input x and outputs $m(x)$ in unary on an output tape using at most $c \log |x| + d$ space for certain constants $c, d > 0$) and (2) there exists a polynomial p satisfying $m(x) \leq p(|x|)$ for all instances x of P.

As key examples, for any graph-related problem (such as 3DSTCON), $m_{edg}(x)$ and $m_{ver}(x)$ denote respectively the total number of edges and that of vertices in a given graph instance x. Clearly, m_{ver} and m_{edg} are log-space computable. To emphasize the use of size parameter m, we often write (P, m) in place of P. We say that a multi-tape Turing machine M uses *logarithmic space* (or *log space*, in short) with respect to size parameter m if there exist two absolute constants $c, d \geq 0$ such that each of the work tapes (not including input and output tapes) used by M on x are upper-bounded by $c \log m(x) + d$ on every input x.

Two specific notations L and NL respectively stand for the classes of all decision problems solvable on multi-tape deterministic and nondeterministic Turing machines using log space. It is known that the additional requirement of "polynomial runtime" does not change these classes. More generally, PTIME,SPACE($s(n)$) expresses a class composed of all (parameterized) decision problems (P, m) solvable deterministically in polynomial time (in $|x|$) using space at most $s(m(x))$ on any instance x given to P.

To define NL search and optimization problems in Section 6, it is convenient for us to use a practical notion of "auxiliary Turing machine" (see, e.g., [20]). An *auxiliary Turing machine* is a multi-tape deterministic Turing machine equipped with an extra read-only *auxiliary input tape*, in which a tape head scans each auxiliary input symbol only once by moving from the left to the right. Given two alphabets Σ and Γ, a (parameterized) decision problem (P, m) with $P \subseteq \Sigma^* \times \Gamma^*$ is in auxL if there exist a polynomial p and an auxiliary Turing machine M that takes a standard input x and an auxiliary input y of length $p(|x|)$ and decides whether M accepts (x, y) or not in time polynomial in $|x|$ using space logarithmic in $m(x)$. Its functional version is denoted by auxFL, provided that each underlying Turing machine is equipped with an extra *write-only* output tape (in which a tape head moves to the right whenever it writes a non-blank output symbol) and that the machine produces output strings of at most polynomial length.

3 Sub-Linear Space and Short Reductions

Recall from [9, 10] that the term "sub-exponential" means $2^{\varepsilon m(x)}\, poly(|x|)$ for a certain constant $\varepsilon \in (0, 1)$. In contrast, our main subject is polynomial-time, sub-linear-space computability, where the term "sub-linear" refers to functions of the form $m(x)^{\varepsilon}\, polylog(|x|)$ on input instances x for a certain constant $\varepsilon \in (0, 1)$ and a certain polylogarithmic function $polylog(n)$. As noted in Section 1.2, the multiplicative factor $polylog(|x|)$ can be eliminated whenever $m(x)$ is relatively large.

First, we will provide basic definitions for (parameterized) decision problems. A decision problem P parameterized by size parameter m is said to be *solvable in polynomial time using sub-linear space* if, for a certain choice of constant $\varepsilon \in (0, 1)$, there exist a deterministic Turing machine M_ε, a polynomial p_ε, and a polylogarithmic function ℓ_ε for which M solves P simultaneously in at most $p_\varepsilon(|x|)$ steps using space at most $m(x)^{\varepsilon}\ell_\varepsilon(|x|)$ for all instances x given to P.

The notation PsubLIN expresses the collection of all (parameterized) decision problems (P, m) that are solvable in polynomial time using sub-linear space. In other words, PsubLIN $= \bigcup_{\varepsilon\in(0,1)}$ PTIME,SPACE($m(x)^{\varepsilon}\ell(|x|)$) for input instances x, where m refers to an arbitrary (log-space) size parameter and ℓ refers to any polylogarithmic function. It thus follows that L $\subseteq$ PsubLIN $\subseteq$ P but none of these inclusions is known to be proper.

The notion of reducibility among decision problems is quite useful in measuring the relative complexity of the problems. For the class PsubLIN, in particular, we need a restricted form of reducibility, which we call "short" reducibility, satisfying a special property that any

outcome of the reduction is linearly upper-bounded in size by an input of the reduction. We will define such restricted reductions for (parameterized) decision problems of our interest.

We begin with a description of *L-m-reducibility* for (parameterized) decision problems. Given two (parameterized) decision problems (P_1, m_1) and (P_2, m_2), we say that (P_1, m_1) is *L-m-reducible to* (P_2, m_2), denoted by $(P_1, m_1) \leq_m^{\mathrm{L}} (P_2, m_2)$, if there is a function $(f, ||) \in \mathrm{FL}$ (where $||$ refers to the bit length) and two constants $k_1, k_2 > 0$ such that, for any input string x, (i) $x \in P_1$ iff $f(x) \in P_2$ and (iii) $m_2(f(x)) \leq m_1(x)^{k_1} + k_1$. Notice that all functions in FL are, by their definition, polynomially bounded.

Concerning polynomial-time sub-linear-space solvability, we introduce a restricted variant of this L-m-reducibility, which we call the *short L-m-reducibility* (or sL-m-reducibility, in short), obtained by replacing the equality $m_2(f(x)) \leq m_1(x)^{k_1} + k_1$ in the above definition of $\leq_m^{\mathrm{L}}$ with $m_2(f(x)) \leq k_1 m_1(x) + k_1$. To express this new reducibility, we use a new notation of $\leq_m^{\mathrm{sL}}$.

Since many-one reducibility is too restrictive to use, we need a stronger notion of Turing reduction, which fits into a framework of polynomial-time, sub-linear-space computability. Our reduction is actually a *polynomial-time sub-linear-space reduction family* (SLRF, in short), performed by oracle Turing machines. A (parameterized) decision problem (P_1, m_1) is *SLRF-T-reducible to* another one (P_2, m_2), denoted by $(P_1, m_1) \leq_T^{\mathrm{SLRF}} (P_2, m_2)$, if, for every fixed value $\varepsilon > 0$, there exist an oracle Turing machine M_ε equipped with an extra write-only query tape, a polynomial p_ε, a polylog function ℓ_ε, and three constants $k_1, k_2 \geq 1$ such that, for every instance x to P_1, (1) $M_\varepsilon^{P_2}$ runs in at most $p_\varepsilon(|x|)$ time using at most $m_1(x)^\varepsilon \ell_\varepsilon(|x|)$ space, provided that its query tape is not subject to this space bound, (2) if $M_\varepsilon^{P_2}(x)$ makes a query to P_2 with query word z written on the query tape, then z satisfies both $m_2(z) \leq m_1(x)^{k_1} + k_1$ and $|z| \leq |x|^{k_2} + k_2$, and (3) after M_ε makes a query, in a single step, it automatically erases its query tape, it returns its tape head back to the initial cell, and oracle P_2 informs the machine of its answer by changing the machine's inner state.

The *short SLRF-T-reducibility* (or sSLRF-T-reducibility, in short) is obtained from the SLRF-reducibility by substituting $m_2(z) \leq k_1 m_1(x) + k_1$ for the above inequality $m_2(z) \leq m_1(x)^{k_1} + k_1$. The notation $\leq_T^{\mathrm{sSLRF}}$ denotes this restricted reducibility. In the case where M_ε is limited to log-space usage, we use a different notation of $\leq_T^{\mathrm{sL}}$. Note that any $\leq_T^{\mathrm{sSLRF}}$-reduction is an $\leq_T^{\mathrm{SLRF}}$-reduction but the converse is not true because there is a pair of problems reducible by $\leq_T^{\mathrm{SLRF}}$-reductions but not by $\leq_T^{\mathrm{sSLRF}}$-reductions.

For any reduction $\leq_r$, a decision problem P is said to be *$\leq_r$-complete* for a given class $\mathcal{C}$ of problems if (1) $P \in \mathcal{C}$ and (2) every problem Q in $\mathcal{C}$ is $\leq_r$-reducible to P. We use the notation $\leq_r(\mathcal{C})$ to express the collection of all problems that are $\leq_r$-reducible to certain problems in $\mathcal{C}$. When $\mathcal{C}$ is a singleton, say, $\mathcal{C} = \{A\}$, we write $\leq_r(A)$ instead of $\leq_r(\{A\})$.

It follows that $(P_1, m_1) \leq_m^{\mathrm{L}} (P_2, m_2)$ implies $(P_1, m_1) \leq_T^{\mathrm{L}} (P_2, m_2)$, which further implies $(P_1, m_1) \leq_T^{\mathrm{SLRF}} (P_2, m_2)$. The same statement holds for $\leq_m^{\mathrm{sL}}$, $\leq_T^{\mathrm{sL}}$, and $\leq_T^{\mathrm{sSLRF}}$. Moreover, $(P_1, m_1) \leq_m^{\mathrm{sL}} (P_2, m_2)$ implies $(P_1, m_1) \leq_m^{\mathrm{L}} (P_2, m_2)$. The same holds for $\leq_T^{\mathrm{sSLRF}}$ and $\leq_T^{\mathrm{SLRF}}$.

Here are other basic properties of SLRF-T- and sSLRF-T-reductions.

▶ **Lemma 3.1.**

1. *The reducibilities $\leq_T^{\mathrm{SLRF}}$ and $\leq_T^{\mathrm{sSLRF}}$ are reflexive and transitive.*
2. *The class* PsubLIN *is closed under $\leq_T^{\mathrm{sSLRF}}$-reductions.*
3. *There exist recursive decision problems X and Y such that $X \leq_T^{\mathrm{SLRF}} Y$ but $X \not\leq_T^{\mathrm{sSLRF}} Y$. A similar statement holds also for $\leq_m^{\mathrm{L}}$ and $\leq_m^{\mathrm{sL}}$.*

4 The 2CNF Boolean Formula Satisfiability Problem and SNL

We will make a brief discussion on 2SAT (2CNF Boolean formulas satisfiability problem) and the complexity class SNL. As noted in Section 1.1, 2SAT is NL-complete under L-m-reductions.

In what follows, we are focused on two specific size parameters: $m_{vbl}(x)$ and $m_{cls}(x)$, which respectively denote the numbers of propositional variables and clauses appearing in formula-related instance x (not necessarily limited to instances of 2SAT).

We further restrict 2SAT by limiting the number of literals appearing in an input Boolean formula as follows. Let $k \in \mathbb{N}^+$. We denote by $2\mathrm{SAT}_k$ the collection of all formulas ϕ in 2SAT such that, for each variable v in ϕ, the number of occurrences of v and $\overline{v}$ is at most k. Since $2\mathrm{SAT}_1$ and $2\mathrm{SAT}_2$ are solvable using only log space, we force our attention on the case of $k \geq 3$. From $(2\mathrm{SAT}, ||) \leq_m^{\mathrm{L}} (2\mathrm{SAT}_3, ||)$ with a help of the fact that 2SAT is NL-complete, we can immediately obtain the following.

▶ **Proposition 4.1.** *For each index $k \geq 3$, $2\mathrm{SAT}_k$ is* NL*-complete.*

To solve 2SAT in polynomial time, we need slightly larger than sub-linear space.

▶ **Theorem 4.2.** *For a certain constant $c > 0$ and a polylog function $\ell(n)$, 2SAT with n variables and m clauses can be solved in polynomial time using $n^{1-c/\sqrt{\log n}}\ell(m+n)$ space.*

For any reduction $\leq_r$ defined in Section 3, we write $(P_1, m_1) \equiv_r (P_2, m_2)$ if both $(P_1, m_1) \leq_r (P_2, m_2)$ and $(P_2, m_2) \leq_r (P_1, m_1)$ hold.

▶ **Lemma 4.3.** *Let $m \in \{m_{vbl}, m_{cls}\}$ and $k \geq 3$. (1) $(2\mathrm{SAT}_k, m) \equiv_m^{\mathrm{sL}} (2\mathrm{SAT}_3, m)$ and (2) $(2\mathrm{SAT}_3, m_{vbl}) \equiv_m^{\mathrm{sL}} (2\mathrm{SAT}_3, m_{cls})$.*

Contrary to Lemma 4.3(2), it is still unknown whether $(2\mathrm{SAT}, m_{vbl}) \equiv_T^{\mathrm{sL}} (2\mathrm{SAT}, m_{cls})$.

Hereafter, we will define the notion of SNL formulas, which induce the complexity class SNL. Let $x = (S_1, \ldots, S_a, x_1, \ldots, x_b)$ be any instance, including "sets" S_i and "objects" x_j. An *SNL formula* Ψ is of the form $\exists T \forall i_1 \cdots \forall i_r \forall y_1 \cdots \forall y_s \exists z_1 \cdots \exists z_t\ \psi$, where ψ is a quantifier-free formula, which is a Boolean combination of *atomic formulas* of the following forms: $T(i, v)$, $(u_1, \ldots, u_k) \in S_j$, $u = v$, $i \leq j$, and $symb(v, i) = a$ (i.e., a is the ith symbol of v), where T is a second-order predicate symbol, and $i_1, \ldots, i_r, y_1, \ldots, y_s, z_1 \ldots, z_t$ are first-order variables, having the following *semantic model* for Ψ. In this model, T ranges over a subset of $[p(|x|)] \times U_x$ (where U_x is a universe) with $|U_x| \leq cm(x)$, each i_j ranges a number in $[p_j(|x|)]$, each y_j takes an element in another universe $U_{x,j}$ with $|U_{x,j}| \leq c_j m(x)$, and each z_j ranges over a set $Z_{x,j}$ of at most e elements (i.e., $|Z_{x_j}| \leq e$) for absolute constants $c, c_j, e \geq 1$ and polynomials p, p_j, not depending on the choice of x. A *certificate size* $m_{cert}(x)$ is defined to be $|U_x|$ as our basis size parameter.

As a quick example, let us consider a (parameterized) decision problem (A, m) such that there are a polynomial p, a constant $c > 0$, and a deterministic Turing machine M recognizing A simultaneously in time at most $p(|x|)$ using space at most $\log_{|\Gamma|} m(x) + c$ for every instance x to A, where Γ is a work-tape alphabet. We assume that M terminates in a configuration in which the work tape is blank and all tape heads return to the initial position. For our convenience, δ is extended to include a special transition from an accepting configuration to itself. To express (A, m), we define an SNL-formula $\Psi \equiv \Psi(x)$ as: $\exists T[Func(T) \wedge \exists v_0 \exists v_1 [T(1, v_0) \wedge T(last(T), v_1) \wedge v_1 \in ACC_x \wedge \forall i \forall v \exists w [T(i, v) \rightarrow (v, w) \in Tran_\delta \wedge T(i+1, w)]]]$ with a semantic model supporting $T \subseteq [p(|x|)] \times U_x$, $i \in [p(|x|)]$, $v_0, x_1, v, w, \in U_x$, where $U_x = \Gamma^{\log_{|\Gamma|} m(x)+c}$, ACC_x is the set of a unique accepting configuration, $last(T)$ indicates

the largest index i that ensures $\exists v[T(i,v)]$, $Trans_\delta$ expresses a δ-transition between two configurations, and $Func(T)$ asserts that T represents a function $f(i)=z$ satisfying $T(i,z)$. Note that $|U_x| \leq |\Gamma|^{c+1}m(x)$. Hence, (A,m) belongs to SNL.

5 The Working Hypothesis LSH for $2SAT_3$

The exponential time hypothesis (ETH) has served as a driving force to obtain better lower bounds on the computational complexity of various important problems (see, e.g., [14]).

In Theorem 4.2, we have seen that 2SAT with n variables and m clauses can be solved in polynomial time using $n^{1-c/\sqrt{\log n}}polylog(m+n)$ space for a certain constant $c>0$; however, it is not yet known to be solved in polynomial time using sub-linear space. This circumstance encourages us to propose (in Section 1.3) a practical working hypothesis – the *linear space hypothesis* (LSH) for $2SAT_3$ – which asserts the insolvability of $(2SAT_3, m_{ver})$ in polynomial time using sub-linear space. The choice of m_{vbl} does not matter; as shown in Lemma 4.3(2) with a help of Lemma 3.1(2), we can replace m_{vbl} in the definition of LSH by m_{cls}. Theorem 1.5 has further given two alternative definitions to LSH in terms of $LP_{2,3}$ and 3DSTCON.

As noted in Section 1.3, Theorem 1.2 states that the above working hypothesis leads to $L \neq NL$. Moreover, Proposition 1.3 asserts that LSH for $2SAT_3$ is equivalent to $\delta_3 = 1$.

The working hypothesis LSH concerns with $2SAT_3$ but it also carries over to 2SAT.

▶ **Lemma 5.1.** *Assuming that LSH for* $2SAT_3$ *is true, each of the following statements holds: (1)* $\leq_T^{sSLRF}(2SAT_3, m_{vbl}) \not\subseteq PsubLIN$ *and (2)* $(2SAT, m_{vbl}) \notin PsubLIN$.

As another consequence of LSH for $2SAT_3$, we can show the existence of a pair of problems in the class $\leq_T^{sSLRF}(2SAT_3, m_{vbl})$, which are incomparable with respect to $\leq_T^{sSLRF}$-reductions. This indicates that the class $\leq_T^{sSLRF}(2SAT_3, m_{vbl})$ has a fine, complex structure with respect to sSLRF-T-reducibility.

▶ **Theorem 5.2.** *Assuming LSH for* $2SAT_3$*, there are two decision problems* (A, m_A) *and* (B, m_B) *in* $\leq_T^{sSLRF}(2SAT_3, m_{vbl})$ *such that* $(A, m_A) \not\leq_T^{sSLRF} (B, m_B)$ *and* $(B, m_B) \not\leq_T^{sSLRF} (A, m_A)$.

6 Proofs of the Four Examples of LSH Applications

In Section 1.4, we have described four examples of how to apply our working hypothesis LSH for $2SAT_3$. Here, we will give three of their proofs.

First, we will briefly describe *(parameterized) NL search problems*. In general, a search problem parameterized by (log-space) size parameter m is expressed as (I, SOL, m), where I consists of (admissible) instances and SOL is a function from I to a set of strings (called a *solution space*) such that, for any $(x,y) \in I \circ SOL$, $y \in SOL(x)$ implies $|y| \leq am(x)+b$ for certain constants $a, b > 0$, where $I \circ SOL$ stands for $\{(x,y) \mid x \in I, y \in SOL(x)\}$. In particular, when we use the standard "bit length" of instances, we omit "||" and write (I, SOL) instead of $(I, SOL, ||)$. Of all search problems, *(parameterized) NL search problems* are (parameterized) search problems (I, SOL, m) for which $I \in L$ and $I \circ SOL \in$ auxL. Finally, we denote by Search-NL the collection of all (parameterized) NL search problems.

We say that a deterministic Turing machine M *solves* (I, SOL, m) if, for any instance $x \in I$, M takes x as input and produces a solution in $SOL(x)$ if $SOL(x) \neq \emptyset$, and produces a designated symbol $\perp$ ("no solution") otherwise. Now, we recall from Section 1.4 a special

NL search problem, called Search-1NFA, in which we are asked to find an input of length n accepted by a given λ-free 1nfa M. Theorem 1.6 states that no polynomial-time $O(n^{1/2-\varepsilon})$-space algorithm solves $(\text{Search-1NFA}, m_{nfa})$.

Proof of Theorem 1.6. Toward a contradiction, we assume that $(\text{Search-1NFA}, m_{nfa})$ is solved by a deterministic Turing machine M in time polynomial in $|y|$ using space at most $cm_{nfa}(y)^{1/2-\varepsilon}$ on instances y, where $c, \varepsilon > 0$ are constants. Our aim is to show that $(\text{3DSTCON}, m_{ver})$ can be solved in polynomial time using sub-linear space, because this contradicts LSH for 3DSTCON, which is equivalent to LSH for 2SAT_3 by Theorem 1.5(3).

Let $x = (G, s, t)$ be any instance to 3DSTCON with $G = (V, E)$ and $s, t \in V$. Let $n = |V|$. Associated with this x, we define a 1nfa $N = (Q, \Sigma, \delta, q_0, F)$ as follows. First, let $Q = V$ and $\Sigma = [0, 3]_{\mathbb{Z}}$. Define $q_0 = s$ and $F = \{t\}$. For each $v \in V$, consider its neighbor $out(v) = \{w \in V \mid (v, w) \in E\}$. We assume that all elements in $out(v)$ are enumerated in a fixed linear order as $out(v) = \{w_1, w_2, \ldots, w_k\}$ with $0 \leq k \leq 3$. The transition function δ is defined as $\delta(v, i) = \{w_i\}$ if $0 \leq i \leq k$.

Supposedly, $\gamma = (v_1, v_2, \ldots, v_d)$ is a path from $s = v_1$ to $t = v_d$ in G. For each index $i \in [d]$, we choose an index $\ell(v_i)$ satisfying $v_{i+1} = w_{\ell(v_i)} \in out(v_i)$ and we then set $z = \ell(v_1)\ell(v_2)\cdots\ell(v_{d-1})0^{n-d+1}$. When N reads z, it eventually enters v_d, which is a halting state, and therefore N accepts z. On the contrary, in the case where there is no path from s to t in G, N never accepts any input. Therefore, it follows that (*) 3DSTCON has a path from s to t iff N accepts z.

Finally, we set $y = (N, 1^n)$ as an instance to Search-1NFA parameterized by m_{nfa}. Note that $m_{nfa}(y) = |Q||\Sigma|n \leq 4|V|^2 = 4m_{ver}(z)^2$. By (*), 3DSTCON can be solved by running M on y in polynomial time; moreover, the space required for this computation is upper-bounded by $cm_{nfa}(y)^{1/2-\varepsilon} \leq 2cm_{ver}(x)^{1-2\varepsilon}$, which is obviously sub-linear. ◀

Another NL search problem, Search-UOCK, asks to find, for a given string w, an index sequence $(i_1, \ldots, i_k)$ in increasing order that makes the concatenation $w_{i_1} \cdots w_{i_k}$ equal to w among $\{w_1, w_2, \ldots, w_n\}$. Here, we present the proof of Theorem 1.7.

Proof of Theorem 1.7. Let us assume that there is a polynomial-time $cm_{elm}(x)^{1/2-\varepsilon}$-space algorithm A for $(\text{Search-UOCK}, m_{elm})$ on instances x for certain constants $\varepsilon, c > 0$. We will use this A to solve $(\text{3DSTCON}, m_{ver})$ in polynomial time using sub-linear space.

Let $x = (G, s, t)$ be any instance to 3DSTCON with $G = (V, E)$. For simplicity of our argument, let $V = \{1, 2, \ldots, n\}$, $s = 1$, and $t = n$. Now, we define $\langle i, j\rangle = (i-1)n + j$ for each pair $i, j \in [n]$. First, we modify G into another graph $G' = (V', E')$, where $V' = \{\langle i, j\rangle \mid i, j \in [n]\}$ and $E' = \{(\langle i, j\rangle, \langle i', j'\rangle) \in V' \times V' \mid i' = i + 1, (j, j') \in E\}$. Note that $|V'| = n^2$ and $|E'| = |V||E| \leq 3|V|^2 = 3n^2$ since $|E| \leq 3|V|$. Moreover, let $s' = \langle 1, s\rangle$ and $t' = \langle n, t\rangle$. This new graph G' satisfies the following property, called the *topological order*: for any pair $i, j \in V'$, $(i, j) \in E'$ implies $i < j$.

From (G', s', t'), we want to define $w = bin(1)\#bin(2)\#\cdots\#bin(n)\#$, where $bin(i)$ indicates the binary representation of a natural number i and $\#$ is a designated separator not in $\{0, 1\}$. Moreover, for each edge $(i, j) \in E'$, we define $w_{ij} = bin(i+1)\#bin(i+2)\#\cdots\#bin(j)\#$. It follows that, for each w_{ij}, if w_{ij} is a substring of w, then w_{ij} must be unique. Note that $z = (w, w_{ij})_{(i,j)\in E'}$ is an instance to Search-1NFA with $m_{elm}(z) = |E'| \leq 3n^2 = 3m_{ver}(x)^2$.

By running A on input z, we can solve $(\text{3DSTCON}, m_{edg})$ for instance x in time polynomial in $|x|$ using space at most $cm_{elm}(z)^{1/2-\varepsilon}$, which equals $3cm_{ver}(x)^{1-2\varepsilon}$. This contradicts LSH for 3DSTCON, which implies LSH for 2SAT_3 by Theorem 1.5(3). ◀

The next practical application of the working hypothesis LSH for $2SAT_3$ targets the area of combinatorial NL optimization. An *NL optimization problem* (or an NLO problem) P is a tuple $(I, SOL, mes, goal)$ with $I \in \mathrm{L}$, $I \circ SOL \in \mathrm{auxL}$, $mes : I \circ SOL \to \mathbb{N}^+$ in auxFL, and $goal \in \{\textsc{max}, \textsc{min}\}$. See [19, 20] for its precise definition. Let NLO stand for the class of all NLO problems. An *optimal solution* y for instance x must satisfy $mes(x, y) = mes^*(x)$, where $mes^*(x) = goal_{y \in SOL(x)}\{mes(x, y)\}$. The *performance ratio* R of a solution y on an instance x is $R(x, y) = \max\{\frac{mes^*(x)}{mes(x,y)}, \frac{mes(x,y)}{mes^*(x)}\}$.

We say that an NLO problem $P = (I, SOL, mes, goal)$ parameterized by size parameter m is *solvable using log space* if there is a deterministic Turing machine that takes any instance $x \in I$ and outputs an optimal solution in $SOL(x)$ using logarithmically many tape cells in terms of size parameter $m(x)$. We write $\mathrm{LO}_{\mathrm{NLO}}$ to denote the class of all NLO problems solvable in polynomial time.

An NPO problem P is said to be *log-space γ-approximable* if there is a log-space Turing machine such that, for any instance x, if $SOL(x) \neq \emptyset$, then M outputs a solution in $SOL(x)$ with $R(x, M(x)) \leq \gamma$; otherwise, M outputs $\perp$ ("no solution"). The notation $\mathrm{LSAS}_{\mathrm{NLO}}$ denotes the class of NLO problems P for which there exists a log-space approximation scheme for P, where a *log-space approximation scheme* for P is a deterministic Turing machine M that takes inputs of the form (x, k) and outputs a solution y of P using space at most $f(k) \log |x|$ for a certain log-space computable function $f : \mathbb{N} \to \mathbb{N}$ such that the performance ratio R satisfies $R(x, y) \leq 1 + 1/k$. It follows that $\mathrm{LO}_{\mathrm{NLO}} \subseteq \mathrm{LSAS}_{\mathrm{NLO}} \subseteq \mathrm{NLO}$. Here, we are focused on problems in $\mathrm{LSAS}_{\mathrm{NLO}}$, that is, NLO problems having log-space approximation schemes.

Let us recall an NLO problem, called Max-HPP, from Section 1.4. Theorem 1.8 states that no polynomial-time $O(k^{1/3} \log m_{col}(x))$-space algorithm that finds $(1 + \frac{1}{k})$-approximate solutions solves $(\text{Max-HPP}, m_{col})$. To prove this theorem, we state a useful supporting lemma. An optimization problem $(I, SOL, mes, goal)$ parameterized by m is said to be *$g(m(x))$-bounded* if $mes(x, y) \leq g(m(x))$ holds for any $(x, y) \in I \circ SOL$.

▶ **Lemma 6.1.** *Let $c \geq 1$. Every $O(m(x)^c)$-bounded maximization problem in $\mathrm{LSAS}_{\mathrm{NLO}}$, parameterized by log-space size parameter $m(x)$, whose $(1 + \frac{1}{k})$-approximate solutions are found using $O(k^{\frac{1}{2c+1}} \log m(x))$ space can be solved in polynomial time using $O(m(x)^{1/2-\varepsilon})$ space on instances x for a certain constant $\varepsilon \in (0, 1/2)$.*

Proof of Theorem 1.8. Let $\varepsilon > 0$. Note that $(\text{Max-HPP}, m_{col})$ is $m_{col}(z)$-bounded for any instance z. Assume that there is a polynomial-time $O(k^{1/3} \log m_{col}(z))$-space algorithm of finding $(1 + \frac{1}{k})$-approximate solutions of Max-HPP on instances z. Lemma 6.1 then implies that $(\text{Max-HPP}, m_{col})$ is solved by a certain deterministic Turing machine M in polynomial time using space at most $cm_{col}(z)^{1/2-\varepsilon}$ on instances z for a certain constant $c > 0$. We want to use this machine M to solve $(\text{3DSTCON}, m_{ver})$ in polynomial time using sub-linear space.

Let $x = (G, s, t)$ be any instance given to 3DSTCON with $G = (V, E)$ and $n = |V| \geq 2$. We define another graph $G' = (V', E')$, where $V' = \{(i, v) \mid i \in [n], v \in V\}$ and $E' = \{((i, u), (i + 1, v)) \mid i \in [n - 1], (u, v) \in E\}$. Note that $|V'| = n^2$. We set $s' = (1, s)$ and $t' = (n, t)$. From this graph G', we want to construct an instance $z = (A, n, s')$ to Max-HPP, where A is a $|V'| \times |V'|$ matrix. By identifying vertices in V' with numbers in $[n^2]$, we set $A_{s't'} = A_{t's'} = A_{vv} = 1$ for any $v \in V' - \{t\}$ and $A_{t'v} = A_{vs'} = 1$ for all $v \in V'$. For any other pair $(u, v) \in V' \times V'$, if $(u, v) \in E'$, then we define $A_{vw} = n$; otherwise, define $A_{uv} = 1$. Note that $m_{col}(z) = n^2 = m_{ver}(x)^2$.

If there is a path $(v_1, v_2, \ldots, v_k)$ from s' to t' in G', then we define $v_{k+j} = v_k$ for all indices $j \in [n^2 - k]$. It then follows that $\sum_{i=1}^{n^2-1} A_{v_i v_{i+1}} = (n^2 - 1)n$ and clearly this is

optimal. On the contrary, let $\gamma = (v_1, v_2, \ldots, v_{n^2})$ be an optimal solution with an optimal value $(n^2-1)n$. By the requirement of Max-HPP, v_1 must be s'. Moreover, $A_{v_i v_{i+1}} = n$ holds for each $i \in [n^2-1]$. Hence, if we allow a self-loop at vertex t' in G', then γ forms a path from s'. Since $|V'| = n^2$, γ must include t'. Hence, γ contains a subpath from s' to t' in G'.

We then run M on the input z to obtain an optimal index sequence γ. By the above argument, if $w(\gamma) = (n^2-1)n$, then a path from s to t exists; otherwise, there is no path from s to t. Since M uses at most $cm_{col}(z)^{1/2-\varepsilon}$ space, the space usage of the whole procedure is at most $cm_{col}(z)^{1/2-\varepsilon}$, which turns out to be $cm_{ver}(x)^{1-2\varepsilon}$ by $m_{col}(z) = m_{ver}(x)^2$. Therefore, 3DSTCON is solvable in polynomial time using sub-linear space. This contradicts LSH for 3DSTCON, which is equivalent to LSH for $2SAT_3$ by Theorem 1.5(3). ◀

References

1 G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties, Springer-Verlag, 2003.

2 G. Barnes, J. F. Buss, W. L. Ruzzo, and B. Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. SIAM J. Comput. 27 (1998) 1273–1282.

3 C. Calabro, R. Impagliazzo, V. Kabanets, and R. Paturi. The complexity of unique k-SAT: an isolation lemma for k-CNFs. J. Comput. System Sci. 74 (2008) 386–393.

4 S. A. Cook. The complexity of theorem-proving procedures. In the Proc. of STOC'71, pp.151–158, 1971.

5 S. A. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In the Proc. of STOC'79, pp.338–345, 1979.

6 J. L. Gross, J. Yellen, and P. Zhang. Handbook of Graph Theory. CRC Press, 2014.

7 J. E. Hopcroft and J. D. Ullman. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.

8 N. Immerman. Nondeterministic space is closed under complement. SIAM J. Comput. 17 (1988) 935–938.

9 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. J. Comput. System Sci. 62 (2001) 367–375.

10 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? J. Comput. System Sci. 63 (2001) 512–530.

11 B. Jenner. Knapsack problems for NL. Inform. Process. Lett. 54 (1995) 169–174.

12 N. D. Jones. Space-bounded reducibility among combinatorial problems. J. Comput. System Sci. 11 (1975) 68–75.

13 N. D. Jones, Y. E. Lien, and W. T. Laaser. New problems complete for nondeterministic log space. Math. Systems Theory 10 (1976) 1–17.

14 D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the exponential time hypothesis. Bulletin of the EATCS, No.105, pp.41–71, 2011.

15 O. Reingold. Undirected connectivity in log-space. J. ACM 55 (2008) article 17.

16 W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. J. Comput. System Sci. 4 (1970) 177–192.

17 I. H. Sudborough. On tape-bounded complexity classes and multihead finite automata. J. Comput. System Sci. 10 (1975) 62–76.

18 R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. Acta Inform. 26 (1988) 279–284.

19 T. Tantau. Logspace optimization problems and their approximation properties. Theory Comput. Syst. 41 (2007) 327–350.

20 T. Yamakami. Uniform-circuit and logarithmic-space approximations of refined combinatorial optimization problems. In the Proc. of COCOA 2013, LNCS, vol.8287, pp.318–329 (2013). A complete version is available at arXiv:1601.01118v1, January 2016.

Variations on Inductive-Recursive Definitions

Neil Ghani[1], Conor McBride[2], Fredrik Nordvall Forsberg[3], and Stephan Spahn[4]

1 **University of Strathclyde, Glasgow, Scotland**
2 **University of Strathclyde, Glasgow, Scotland**
3 **University of Strathclyde, Glasgow, Scotland**
4 **Middlesex University, London, England**

Abstract

Dybjer and Setzer introduced the definitional principle of inductive-recursively defined families – i.e. of families $(\mathsf{U} : \mathsf{Set}, \mathsf{T} : \mathsf{U} \to D)$ such that the inductive definition of U may depend on the recursively defined T – by defining a type $\mathsf{DS}\ D\ E$ of codes. Each $c : \mathsf{DS}\ D\ E$ defines a functor $[\![\ c\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$, and $(\mathsf{U}, \mathsf{T}) = \mu[\![\ c\ ]\!] : \mathsf{Fam}\ D$ is exhibited as the initial algebra of $[\![\ c\ ]\!]$.

This paper considers the composition of DS-definable functors: Given $F : \mathsf{Fam}\ C \to \mathsf{Fam}\ D$ and $G : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$, is $G \circ F : \mathsf{Fam}\ C \to \mathsf{Fam}\ E$ DS-definable, if F and G are? We show that this is the case if and only if powers of families are DS-definable, which seems unlikely. To construct composition, we present two new systems UF and PN of codes for inductive-recursive definitions, with $\mathsf{UF} \hookrightarrow \mathsf{DS} \hookrightarrow \mathsf{PN}$. Both UF and PN are closed under composition. Since PN defines a potentially larger class of functors, we show that there is a model where initial algebras of PN-functors exist by adapting Dybjer-Setzer's proof for DS.

1998 ACM Subject Classification F.3.3 Studies of Program Constructs, F.4.1. Mathematical Logic

Keywords and phrases Type Theory, induction-recursion, initial-algebra semantics

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.63

1 Introduction

Codes for inductive-recursive definitions were introduced in a series of papers by Dybjer and Setzer [6, 7, 8]. An initial motivation [5] was to give generic rules that can be specialised to define most types occurring in Martin-Löf Type Theory [13], including inductive families [4] and Tarski-style universes [14]. An inductive-recursive definition defines not only a type, but more generally a family $(\mathsf{U} : \mathsf{Set}, \mathsf{T} : \mathsf{U} \to D)$ of types for some $D : \mathsf{Set}_1$, where the inductive definition of U may depend on the recursively defined T; examples can be found in Section 2. To represent such definitions, Dybjer and Setzer introduced a type $\mathsf{DS}\ D\ E$ of codes representing functors $\mathsf{Fam}\ D \to \mathsf{Fam}\ E$. The family $(\mathsf{U}, \mathsf{T}) = \mu[\![\ c\ ]\!] : \mathsf{Fam}\ D$ arises as the initial algebra of a functor $[\![\ c\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ D$ represented by a code $c : \mathsf{DS}\ D\ D$.

Induction-recursion is important as it is the strongest form of inductive definition we have, surpassing, for example, inductive definitions [10] and inductive families [2]. This paper asks the following fundamental and significant question:

Is the theory of inductive-recursive definitions, as currently understood, optimal?

We still believe that conceiving of inductive-recursive definitions as initial algebras in the category $\mathsf{Fam}\ D$ is the right thing to do. However, the current type of codes for generating such functors may not actually be optimal for this purpose. We come to this conclusion by considering the question of composition of codes. Given $[\![\ c\ ]\!] : \mathsf{Fam}\ C \to \mathsf{Fam}\ D$ and

$[\![\ d\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$ represented by Dybjer-Setzer codes $c : \mathsf{DS}\ C\ D$ and $d : \mathsf{DS}\ D\ E$ respectively, is $[\![\ d\ ]\!] \circ [\![\ c\ ]\!] : \mathsf{Fam}\ C \to \mathsf{Fam}\ E$ DS-definable, i.e. is there a code $d \bullet c : \mathsf{DS}\ C\ E$ such that $[\![\ d \bullet c\ ]\!] = [\![\ d\ ]\!] \circ [\![\ c\ ]\!]$? A positive answer would allow modularity in datatype definitions, as one can then replace all inductive arguments (U, T) in a datatype by $F(U, T)$ for any DS-definable functor F by composing with the code for F. For instance, a code for an inductive definition of multiway trees, where each node has a list of subtrees, can be constructed by composing a code for lists with itself. Other classes of data types such as inductive definitions or inductive families are closed under composition [10] – it is a property naturally to be expected from the viewpoint of initial algebra semantics.

It is currently unknown whether DS is closed under composition, although we suspect that it is not. In support of this claim, we show that DS is closed under composition *if and only if* powers $A \to [\![\ c\ ]\!]$ of codes are definable (Section 2, where we also recall Dybjer and Setzer's codes). Since such a power operation is unlikely to exist, we are led to investigate alternative systems of inductive-recursive definitions that *are* closed under composition.

We first introduce a system UF of *uniform* codes for inductive-recursive definitions, and their decoding (Section 3). This system can be regarded as a subsystem of DS, and as such, it is clear that UF-functors have initial algebras since DS-functors do. The novum is that uniformity of codes can be exploited to define powers, which in turn means that a composition operator for uniform codes is obtainable, i.e. we have isolated a subclass of DS-functors that is closed under composition. Next we introduce another system PN of *polynomial* codes, and their decoding (Section 4). Notably, PN contains a constructor for the dependent product of codes which ensures that PN is closed under composition. PN is a supersystem of DS and hence we cannot inherit initial algebras for PN from DS, but must prove their existence directly: we do so by adapting Dybjer-Setzer's proof to our more general setting. The introduction of new alternative formulations of induction-recursion has the potential to have significant impact if they – as we believe to be the case – have better properties than the current one and hence come to supplant the current formulation.

Type-theoretic notation and assumptions. We work informally in a standard type theory with dependent function spaces $\big(x : A\big) \to B(x)$ (written $A \to B$ if x does not occur in B), dependent pair types $\big(\Sigma x : A\big)B(x)$ (written $A \times B$ if x does not occur in B), coproducts $A + B$ with injections inl and inr, and an identity type which we shall simply write as $a = b$. Finite enumerations are denoted by $\{a_1, a_2, \ldots, a_m\}$; instances include $0 = \{\}$, $1 = \{\star\}$ and $2 = \{\mathsf{ff}, \mathsf{tt}\}$. We write anonymous functions as $(a \mapsto b)$, or $(\_ \mapsto b)$ when the argument is not used by the function. We assume two universes à la Russell $\mathsf{Set} : \mathsf{Set}_1$, with $A : \mathsf{Set}$ implying $A : \mathsf{Set}_1$. We assume function extensionality, i.e. that pointwise equal functions are equal. This is essential in our development. For simplicity, we also assume uniqueness of identity proofs, i.e. that if $p : a = b$ and $q : a = b$, then $p = q$, but this assumption should be avoidable with a little more work. In any case, both of these assumptions are valid in extensional Type Theory [14], which readily has a set-theoretic interpretation. The content of this paper (except for the set-theoretical model of PN) has been formalised in Agda[1].

2 Dybjer-Setzer Codes DS for Inductive-Recursive Definitions

We recall the system of Dybjer-Setzer codes DS, how codes represent inductive-recursive definitions, and finally prove powers to be necessary and sufficient for DS to be closed under composition.

[1] Available at `http://personal.cis.strath.ac.uk/fredrik.nordvall-forsberg/variantsIR/`.

2.1 Definition of DS and its Decoding

For $D,\ E : \mathsf{Set}_1$, the type $\mathsf{DS}\ D\ E$ consists of codes that represent functors $\mathsf{Fam}\ D \to \mathsf{Fam}\ E$ describing the constructors of inductive-recursive definitions.

▶ **Definition 1.** Given $D, E : \mathsf{Set}_1$, the large type $\mathsf{DS}\ D\ E : \mathsf{Set}_1$ of *Dybjer-Setzer codes* is inductively defined by the following generators:

$$\iota : E \to \mathsf{DS}\ D\ E$$
$$\sigma : \big(A : \mathsf{Set}\big) \to (A \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$$
$$\delta : \big(A : \mathsf{Set}\big) \to ((A \to D) \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$$

Here ι shall represent trivial functors, σ sums of functors, and δ dependent sums. See Dybjer and Setzer [7] for a more in-depth explanation, and the examples below for intuition. Note that Dybjer and Setzer only considered systems of the form $\mathsf{DS}\ D\ D$, i.e. where $E = D$. For our purposes the more general formulation will be clearer; it also accounts for the fact that $\mathsf{DS}\ D\ E$ is functorial covariantly in E and contravariantly in D.

▶ **Example 2** (W-types)**.** By choosing $D = E = 1$, we can use $\mathsf{DS}\ 1\ 1$ to represent inductive definitions. Let us encode Martin-Löf's type $\mathsf{W}\ S\ P : \mathsf{Set}$ of wellfounded trees, where $S : \mathsf{Set}$ encodes the possible shapes of the tree, and $P : S \to \mathsf{Set}$ maps each shape to its branching degree. This type is inductively defined by the constructor

$$\mathsf{sup} : \big(s : S\big) \to (P(s) \to \mathsf{W}\ S\ P) \to \mathsf{W}\ S\ P$$

Here we see that sup takes one non-inductive argument $s : S$, followed by an inductive argument $P(s) \to \mathsf{W}\ S\ P$, which depends on the first non-inductive one. We will see shortly in Example 5 that $\mathsf{W}\ S\ P$ can be represented by the code $c_{\mathsf{W}\ S\ P} : \mathsf{DS}\ 1\ 1$ with $c_{\mathsf{W}\ S\ P} = \sigma\, S\, (s \mapsto \delta\, P(s)\, (\_ \mapsto \iota\star))$ where σ is used for the non-inductive argument and δ for the inductive one, finally finishing off with a trivial ι.

▶ **Example 3** (A universe closed under W-types)**.** We get considerably more power by choosing $D = E = \mathsf{Set}$. Now we can represent a universe containing 2 that is *closed under* W*-types* by the code $c_{2\mathsf{W}} : \mathsf{DS}\ \mathsf{Set}\ \mathsf{Set}$, where

$$c_{2\mathsf{W}} = \sigma\ \{\mathsf{two}, \mathsf{w}\}\ (\mathsf{two} \mapsto \iota\ 2; \mathsf{w} \mapsto \delta\ 1\ (X \mapsto (\delta\ (X\star)\ (Y \mapsto \iota\ (\mathsf{W}\ (X\star)\ Y)))))$$

First we offer a choice between two constructors: two and w using σ. In the two case, we use an ι code to ensure the name two decodes to 2; in the w case, we ask for a name s for the shapes of the W-type using $\delta\ 1$, and for every element in the decoding of that name, we ask for a name for the branching degrees using $\delta\ (X\star)$ – here $X : 1 \to \mathsf{Set}$ represents the decoding of the name s. The rest of the code gets to depend on the decoding $Y : X\star \to \mathsf{Set}$ of this family, and we finish by declaring that this constructor decodes to $\mathsf{W}\ (X\star)\ Y$. Note that this code can be written as a coproduct of codes $c_2\ +_{\mathsf{DS}}\ c_{\mathsf{W}}$: generally for $c\ d : \mathsf{DS}\ D\ E$, we define their coproduct $c +_{\mathsf{DS}} d = \sigma\ 2\ (\mathsf{ff} \mapsto c\,;\ \mathsf{tt} \mapsto d)$. We will return to this in Example 11.

Decoding of Dybjer-Setzer codes as functors on families make the above intuitions precise. For $D : \mathsf{Set}_1$, $\mathsf{Fam}\ D$ is the category where objects are families of Ds, i.e. pairs (A, P) where $A : \mathsf{Set}$ and $P : A \to D$; a morphism $(A, P) \to (B, Q)$ consists of a function $f : A \to B$ together with a proof that $Q(f(a)) = P(a)$ for each $a : A$. For future reference, we note that Fam is a functor with action on morphisms $\mathsf{Fam}(h)\,(A, P) = (A, h \circ P)$ and moreover a monad with unit $\eta_{\mathsf{Fam}}(e) = (1, \_ \mapsto e)$ and multiplication $\mu_{\mathsf{Fam}} : \mathsf{Fam}\ (\mathsf{Fam}\ D) \to \mathsf{Fam}\ D$ given by $\mu_{\mathsf{Fam}}(A, P) = \big((\Sigma x : A)(P(x)_0), (x, y) \mapsto (P(x))_1\ y\big)$ where we have written $P(x)_0$ and $P(x)_1$ for the components of the family $P(x) = (P(x)_0, P(x)_1)$.

▶ **Definition 4.** Let $D, E : \mathsf{Set}_1$ and $c : \mathsf{DS}\ D\ E$. We define the *decoding* of c as the functor $[\![\ c\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$ given by $[\![\ c\ ]\!](A,P) = ([\![\ c\ ]\!]_0(A,P), [\![\ c\ ]\!]_1(A,P))$, where $[\![\ \_\ ]\!]_0 : \mathsf{DS}\ D\ E \to \mathsf{Fam}\ D \to \mathsf{Set}$ and $[\![\ \_\ ]\!]_1 : \big(c : \mathsf{DS}\ D\ E\big) \to \big(Z : \mathsf{Fam}\ D\big) \to [\![\ c\ ]\!]_0\ Z \to E$ are defined by

$$
\begin{aligned}
&[\![\ \iota\ e\ ]\!]_0\ (U,T) = 1 && [\![\ \iota\ e\ ]\!]_1\ (U,T)\ \star = e\\
&[\![\ \sigma\ A\ f\ ]\!]_0\ (U,T) = \big(\Sigma a : A\big)([\![\ f\ a\ ]\!]_0\ (U,T)) && [\![\ \sigma\ A\ f\ ]\!]_1\ (U,T)\ (a,x) = [\![\ f\ a\ ]\!]_1\ (U,T)\ x\\
&[\![\ \delta\ A\ F\ ]\!]_0\ (U,T) = && [\![\ \delta\ A\ F\ ]\!]_1\ (U,T)\ (g,x) =\\
&\quad \big(\Sigma g : A \to U\big)([\![\ F\ (T \circ g)\ ]\!]_0\ (U,T)) && \qquad [\![\ F\ (T \circ g)\ ]\!]_1\ (U,T)\ x
\end{aligned}
$$

▶ **Example 5.** For decoding Example 2, note that $\mathsf{Fam}\ 1 \cong \mathsf{Set}$ since the second component of such a family is trivial. Thus, if $(W,T) : \mathsf{Fam}\ 1$, then

$$[\![\ c_{\mathsf{W}\ S\ P}\ ]\!]_0(W,T) = \big(\Sigma s : S\big)\big((P(s) \to W) \times 1\big) \tag{1}$$

such that indeed $\mathsf{sup} : [\![\ c_{\mathsf{W}\ S\ P}\ ]\!]_0(\mathsf{W}\ S\ P, \_) \to \mathsf{W}\ S\ P$ (up to isomorphism), and initial algebras of $[\![\ c_{\mathsf{W}\ S\ P}\ ]\!] : \mathsf{Fam}\ 1 \to \mathsf{Fam}\ 1$ are W-types. Instead of leaving the fibres of the family trivial, we can "upgrade" the given code to do something interesting in the whole family. For instance, if we redefine $c_{\mathsf{W}\ S\ P} : \mathsf{DS}\ \mathsf{Set}\ \mathsf{Set}$ by

$$c_{\mathsf{W}\ S\ P} = \sigma\ S\ (s \mapsto \delta\ P(s)\ (Y \mapsto \iota\ \Big(\big(x : P(s)\big) \to Y\ x\Big)))$$

the index type decoding (1) stays the same, but the decoding $[\![\ c_{\mathsf{W}\ S\ P}\ ]\!]_1(W,T)$ applies T everywhere in the given structure. In particular, if we choose $S = \mathbb{N}$ and $P = \mathsf{Fin}$, where $\mathsf{Fin}\ n$ is a finite type with n elements, then $[\![\ c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin}}\ ]\!](W,T) \cong (\mathsf{List}\, W,\ [w_1, \ldots, w_n] \mapsto T\, w_1 \times \ldots \times T\, w_n)$. We will see a use of this upgraded code later in Example 21.

▶ **Example 6.** Similarly, the decoding of the code $c_{2\mathsf{W}} : \mathsf{DS}\ \mathsf{Set}\ \mathsf{Set}$ from Example 3 satisfies $[\![\ c_{2\mathsf{W}}\ ]\!]_0(U,T) \cong 1 + \big(\Sigma s : U\big)(T(s) \to U)$ with $[\![\ c_{2\mathsf{W}}\ ]\!]_1(U,T)\ (\mathsf{inl}\star) = 2$ and $[\![\ c_{2\mathsf{W}}\ ]\!]_1(U,T)\ (\mathsf{inr}\,(s,p)) = \mathsf{W}\ (T\,s)\ (T \circ p)$ which are the equations for a universe closed under W-types.

Dybjer and Setzer [7] also give rules ensuring that $[\![\ c\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ D$ has an initial algebra $(\mathsf{U}_{[\![\ c\ ]\!]}, \mathsf{T}_{[\![\ c\ ]\!]})$ for every $c : \mathsf{DS}\ D\ D$. We omit them here.

2.2 Composition of DS codes

We are now approaching the actual topic of the paper. Given DS-codes $c : \mathsf{DS}\ C\ D$ and $d : \mathsf{DS}\ D\ E$, is there a code $d \bullet c : \mathsf{DS}\ C\ E$ such that $[\![\ d \bullet c\ ]\!](U,T) \cong [\![\ d\ ]\!]([\![\ c\ ]\!](U,T))$? We immediately notice that it is easy to define postcomposition of any code by a ι or a σ code: the functor $[\![\ \iota\ e\ ]\!]$ ignores its argument, hence so must $[\![\ (\iota\ e) \bullet c\ ]\!]$, and for σ codes, we can just proceed structurally. The δ case, however, requires more thought. Again, looking first at the action on index types of the families, we find for the right hand side of the above equation

$$
\begin{aligned}
[\![\ \delta\ A\ F\ ]\!]_0([\![\ c\ ]\!]_0 Z) &= \big(\Sigma g : A \to [\![\ c\ ]\!]_0 Z\big)\big([\![\ F\big([\![\ c\ ]\!]_1(Z) \circ g\big)\ ]\!]_0([\![\ c\ ]\!]Z)\big)\\
&= \Big(\big(A \longrightarrow_{\mathsf{Fam}} [\![\ c\ ]\!]Z\big) \gg\!=_{\mathsf{Fam}} \big(g \mapsto [\![\ F\big([\![\ c\ ]\!]_1(Z) \circ g\big)\ ]\!]_0([\![\ c\ ]\!]Z)\big)\Big)_0
\end{aligned}
$$

where $\_ \gg\!=_{\mathsf{Fam}} \_ : \mathsf{Fam}\ D \to (D \to \mathsf{Fam}\ E) \to \mathsf{Fam}\ E$ is the bind of the Fam monad defined by $Z \gg\!=_{\mathsf{Fam}} h = \mu_{\mathsf{Fam}}\,(\mathsf{Fam}(h)\,Z)$, and

$$
\begin{aligned}
&\_ \longrightarrow_{\mathsf{Fam}} \_ : \big(S : \mathsf{Set}\big) \to \mathsf{Fam}\ D \to \mathsf{Fam}\ (S \to D)\\
&S \longrightarrow_{\mathsf{Fam}} (A,P) = (S \to A, g \mapsto P \circ g)
\end{aligned}
$$

is a power in the category of elements $\big(\Sigma D : \mathsf{Set}_1\big)(\mathsf{Fam}\ D)$ of the functor Fam. This suggests that to define $(\delta\ A\ F) \bullet c$, we need to internalise $\gg\!=_{\mathsf{Fam}}$ and $\longrightarrow_{\mathsf{Fam}}$ in the system DS. The first is readily achievable, because $\mathsf{DS}\ C$ is also a monad [11]:

▶ **Proposition 7.** *There is an operation* $\_ \gg\!= \_ : \mathsf{DS}\ C\ D \to (D \to \mathsf{DS}\ C\ E) \to \mathsf{DS}\ C\ E$ *such that* $[\![\ c \gg\!= g\ ]\!]\ Z \cong [\![\ c\ ]\!]\ Z \gg\!=_{\mathsf{Fam}} (e \mapsto [\![\ g\, e\ ]\!]\ Z)$ *for every* $Z : \mathsf{Fam}\ C$, $c : \mathsf{DS}\ C\ D$ *and* $g : D \to \mathsf{DS}\ C\ E$.

Thus it remains to define powers of codes. Here, however, we hit a wall trying to define $S \longrightarrow c$ by induction on c: to apply the inductive hypothesis on $f\ a$ in the following S-fold power of a σ code

$$S \to [\![\ \sigma\ A\ f\ ]\!]_0\ Z = S \to \big(\Sigma a : A\big)([\![\ f\ a\ ]\!]_0\ Z) \cong \big(\Sigma g : S \to A\big)\big((x : S) \to [\![\ f\ (g\ x)\ ]\!]_0\ Z\big)$$

we would need to generalise our construction to *dependent products* $\big(x : S\big) \to c(x)$ where $c : S \to \mathsf{DS}\ D\ E$. But, if we do so, we can no longer do an induction on c, and we are stuck. Even worse, any definition of composition necessarily involves powers:

▶ **Theorem 8.** *There is a composition operator for* DS *if and only if there is a power operator for* DS. *Here, by composition and power operators we mean terms*

$$\_ \bullet \_ : \mathsf{DS}\ D\ E \to \mathsf{DS}\ C\ D \to \mathsf{DS}\ C\ E$$
$$\_ \longrightarrow \_ : \big(S : \mathsf{Set}\big) \to \mathsf{DS}\ D\ E \to \mathsf{DS}\ D\ (S \to E)$$

respectively such that $[\![\ c \bullet d\ ]\!]\ Z \cong [\![\ c\ ]\!]([\![\ d\ ]\!]\ Z)$ *and* $[\![\ S \longrightarrow c\ ]\!]\ Z \cong (S \longrightarrow_{\mathsf{Fam}} [\![\ c\ ]\!]\ Z)$.

Proof. Given $\_ \longrightarrow \_$, we can define $\_ \bullet \_$ by

$$\begin{aligned}(\iota\ e) \bullet d &= \iota\ e\\ (\sigma\ A\ f) \bullet d &= \sigma\ A\ (a \mapsto (f\ a) \bullet d)\\ (\delta\ A\ F) \bullet d &= (A \longrightarrow d) \gg\!= (g \mapsto (F\ g) \bullet d)\end{aligned}$$

using Proposition 7. Conversely, $A \longrightarrow c \coloneqq (\delta\ A\ (h \mapsto \iota\ h)) \bullet c$ is a power operator. ◀

Two natural options suggest themselves as solutions: (i) restrict codes to ensure that no dependency arises in the definition of powers; (ii) devise a system with dependent products of codes. In the next two sections, we investigate new systems of codes for both of these solutions.

3 Uniform Codes UF for Inductive-Recursive Definitions

This section presents our first new system for induction-recursion with a native composition operation. The system UF of *uniform codes* is a subsystem of DS (Proposition 14). Informally, a uniform code is a DS code where, for every constructor in a term, all immediate subterms have the same root-constructor. Thus (the shape of) $\sigma\ A\ (a \mapsto \delta\ B(a)\ (h \mapsto \iota\ \phi(a, h)))$ is uniform, whereas $\sigma\ A\ f +_{\mathsf{DS}} \delta\ B\ G = \sigma\ 2\ (\mathsf{tt} \mapsto \sigma\ A\ f; \mathsf{ff} \mapsto \delta\ B\ G)$ is not since one subcode is a σ code while the other is a δ one. Uniform codes originated with Peter Hancock [12].

3.1 Definition of UF and its Decoding

Formally, we define a type of codes $\mathsf{Uni}\ D : \mathsf{Set}_1$ determining the code shapes, simultaneously with a function $\mathsf{Info} : \mathsf{Uni}\ D \to \mathsf{Set}_1$, which assigns to each code the information available for indexing codes depending on it, in a uniform way.

▶ **Definition 9.** Let $D, E : \mathsf{Set}_1$. The large type $\mathsf{UF}\ D\ E : \mathsf{Set}_1$ of *uniform codes for induction-recursion* is defined by $\mathsf{UF}\ D\ E \coloneqq \big(\Sigma c : \mathsf{Uni}\ D\big)(\mathsf{Info}\ c \to E)$, where $\mathsf{Uni}\ D : \mathsf{Set}_1$ and $\mathsf{Info} : \mathsf{Uni}\ D \to \mathsf{Set}_1$ are mutually defined by

$$\begin{aligned}
&\iota_{\mathsf{UF}} : \mathsf{Uni}\ D && \mathsf{Info}\ \iota_{\mathsf{UF}} = 1\\
&\sigma_{\mathsf{UF}} : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D && \mathsf{Info}\ (\sigma_{\mathsf{UF}}\ c\ A) = \big(\Sigma\gamma : \mathsf{Info}\ c\big)(A\ \gamma)\\
&\delta_{\mathsf{UF}} : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D && \mathsf{Info}\ (\delta_{\mathsf{UF}}\ c\ A) = \big(\Sigma\gamma : \mathsf{Info}\ c\big)(A\ \gamma \to D)
\end{aligned}$$

It is easy to see that $\mathsf{UF}\ D\ \_$ is functorial by function composition (alternatively, it is defined as the action of a container [1], and hence automatically functorial). This two-level presentation of codes $(\mathsf{Uni}, \mathsf{Info})$ has similarities with the $(\mathsf{SP}, \mathsf{Arg})$ presentation of Dybjer-Setzer codes in the original paper [6], where however SP was merely inductively defined, whereas here $(\mathsf{Uni}, \mathsf{Info})$ is itself an inductive-recursive definition. A further difference is that the definition of Uni is left-nested while SP as well as DS are right-nested in the sense of Pollack [15]. This can be seen as the source of uniformity in the definition.

▶ **Example 10** (W-types, again)**.** In order to get a feel for uniform codes, we return to the W-types of Example 2. A uniform code in $\mathsf{UF}\ 1\ 1$ representing the W-type $\mathsf{W}\ S\ P$ is $c_{\mathsf{W}\ S\ P,\mathsf{UF}} = \delta_{\mathsf{UF}}\ \big(\sigma_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (\_ \mapsto S)\big)\ ((\_, s) \mapsto P(s)) : \mathsf{Uni}\ 1$, together with the terminal map $\mathsf{Info}\ c_{\mathsf{W}\ S\ P,\mathsf{UF}} \to 1$. If we compare this to the Dybjer-Setzer code from Example 2, we see that the order of the (non-base-case) constructors is reversed:

$$\begin{aligned}
(\delta_{\mathsf{UF}}\ \big(\sigma_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (_ \mapsto S)\big)\ ((_, s) \mapsto P(s))\ ,\ _ \mapsto \star) &: \mathsf{UF}\ 1\ 1\\
\sigma\ S\ (s \mapsto \delta\ P(s)\ (_ \mapsto \iota\star)) &: \mathsf{DS}\ 1\ 1
\end{aligned}$$

Also this code can be "upgraded" to a more interesting $\mathsf{UF}\ \mathsf{Set}\ \mathsf{Set}$ code applying a given T everywhere. We get the same decoding as in Example 5 if we replace the trivial map $(\_ \mapsto \star) : \mathsf{Info}\ c_{\mathsf{W}\ S\ P,\mathsf{UF}} \to 1$ by the map $(s, Y, \star) \mapsto \big(x : P(s)\big) \to Y\ x$.

▶ **Example 11** (A universe closed under W-types, again)**.** Example 3 uses coproducts of DS codes. Coproducts of uniform codes a priori do not always exist as the different summands may have different shapes. However, we will prove coproducts of uniform codes to exist in Section 3.3. Assuming, for now, the coproduct $\_ +_{\mathsf{UF}} \_ : \mathsf{UF}\ D\ E \to \mathsf{UF}\ D\ E \to \mathsf{UF}\ D\ E$, we construct the code $c_{2,\mathsf{UF}} +_{\mathsf{UF}} c_{\mathsf{W},\mathsf{UF}} : \mathsf{UF}\ \mathsf{Set}\ \mathsf{Set}$ from the following summands – again note that the nesting is the other way around compared to the DS code in Example 3:

$$\begin{aligned}
c_{2,\mathsf{UF}} &= (\iota_{\mathsf{UF}}, \star \mapsto 2) : \mathsf{UF}\ \mathsf{Set}\ \mathsf{Set}\\
c_{\mathsf{W},\mathsf{UF}} &= \big(\delta_{\mathsf{UF}}\ \big(\delta_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (\star \mapsto 1)\big)\ ((\star, S) \mapsto S\star), ((\star, S), P) \mapsto \mathsf{W}\ (S\star)\ P\big) : \mathsf{UF}\ \mathsf{Set}\ \mathsf{Set}
\end{aligned}$$

Decoding of uniform codes $\mathsf{UF}\ D\ E$ is again given by functors $\mathsf{Fam}\ D \to \mathsf{Fam}\ E$. The definition is very similar to the decoding of DS codes except that UF codes have two components. We use the same notation $[\![-]\!]$ for decoding a uniform code as for decoding a DS code; this convention is reasonable since we will give a semantics-preserving translation from UF to DS in Section 3.2.

▶ **Definition 12.** Let $c : \mathsf{Uni}\ D$ and $\alpha : \mathsf{Info}\ c \to E$. The uniform code $(c, \alpha) : \mathsf{UF}\ D\ E$ induces a functor $[\![\ c, \alpha\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$ by $[\![\ c, \alpha\ ]\!]\ Z = \mathsf{Fam}(\alpha)\ ([\![\ c\ ]\!]_{\mathsf{Uni}}\ Z, [\![\ c\ ]\!]_{\mathsf{Info}}\ Z)$ where $[\![\ \_\ ]\!]_{\mathsf{Uni}} : \mathsf{Uni}\ D \to \mathsf{Fam}\ D \to \mathsf{Set}$ and $[\![\ \_\ ]\!]_{\mathsf{Info}} : \big(c : \mathsf{Uni}\ D\big) \to \big(Z : \mathsf{Fam}\ D\big) \to [\![\ c\ ]\!]_{\mathsf{Uni}}\ Z \to \mathsf{Info}\ c$ are simultaneously defined by induction on c:

$$\begin{aligned}
&[\![\ \iota_{\mathsf{UF}}\ ]\!]_{\mathsf{Uni}}\ (U,T) = 1 && [\![\ \iota_{\mathsf{UF}}\ ]\!]_{\mathsf{Info}}\ (U,T)\ \star = \star \\
&[\![\ \sigma_{\mathsf{UF}}\ c\ A\ ]\!]_{\mathsf{Uni}}\ (U,T) = && [\![\ \sigma_{\mathsf{UF}}\ c\ A\ ]\!]_{\mathsf{Info}}\ (U,T)\ (x,a) = \\
&\quad \big(\Sigma x : [\![\ c\ ]\!]_{\mathsf{Uni}}\ (U,T)\big)(A([\![\ c\ ]\!]_{\mathsf{Info}}\ (U,T)\ x)) && \quad ([\![\ c\ ]\!]_{\mathsf{Info}}\ (U,T)\ x, a) \\
&[\![\ \delta_{\mathsf{UF}}\ c\ A\ ]\!]_{\mathsf{Uni}}\ (U,T) = && [\![\ \delta_{\mathsf{UF}}\ c\ A\ ]\!]_{\mathsf{Info}}\ (U,T)\ (x,g) = \\
&\quad \big(\Sigma x : [\![\ c\ ]\!]_{\mathsf{Uni}}\ (U,T)\big)(A([\![\ c\ ]\!]_{\mathsf{Info}}\ (U,T)\ x) \to U) && \quad ([\![\ c\ ]\!]_{\mathsf{Info}}\ (U,T)\ x, T \circ g)
\end{aligned}$$

▶ **Example 13.** Decoding $c_{\mathsf{W}\ S\ P,\mathsf{UF}}$ from Example 10, we see that

$$[\![\ c_{\mathsf{W}\ S\ P,\mathsf{UF}}\ ]\!]_{\mathsf{Uni}}\ (U,T) = \big(\Sigma(\star, s) : 1 \times S\big)(P(s) \to U)$$

which is isomorphic to the domain of the W-type constructor sup, but this time nested the other way compared to the decoding of $c_{\mathsf{W}\ S\ P}$ in Example 5. By Theorem 18, we will have $[\![\ c +_{\mathsf{UF}} d\ ]\!]\ Z \cong [\![\ c\ ]\!]\ Z\ + [\![\ d\ ]\!]\ Z$, where the right hand side uses the coproduct of families. Hence $c_{2,\mathsf{UF}} +_{\mathsf{UF}} c_{\mathsf{W},\mathsf{UF}}$ from Example 11 decodes correctly.

3.2 Embedding of UF into DS

We embed UF into DS, i.e. we give a translation of codes which is *semantics-preserving* in that the decoding of a code is isomorphic to the decoding of its translation. Since UF codes are "backwards" compared to DS codes, this embedding resembles the well-known accumulator based algorithm for reversing a list. Define $\mathsf{accUFtoDS} : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$ (the second argument is the accumulator) by

$$\begin{aligned}
\mathsf{accUFtoDS}\ \iota_{\mathsf{UF}}\ F &= F\ \star \\
\mathsf{accUFtoDS}\ (\sigma_{\mathsf{UF}}\ c\ A)\ F &= \mathsf{accUFtoDS}\ c\ (\gamma \mapsto \sigma\ (A\ \gamma)\ (a \mapsto F\ (\gamma, a))) \\
\mathsf{accUFtoDS}\ (\delta_{\mathsf{UF}}\ c\ A)\ F &= \mathsf{accUFtoDS}\ c\ (\gamma \mapsto \delta\ (A\ \gamma)\ (h \mapsto F\ (\gamma, h)))
\end{aligned}$$

and define $\mathsf{UFtoDS} : \mathsf{UF}\ D\ E \to \mathsf{DS}\ D\ E$ by kicking things off with a ι:

$$\mathsf{UFtoDS}\ (c, \alpha) = \mathsf{accUFtoDS}\ c\ (\iota \circ \alpha)\ .$$

▶ **Proposition 14.** *The translation* UFtoDS *is an embedding, i.e. for every* $c : \mathsf{UF}\ D\ E$ *and* $Z : \mathsf{Fam}\ D$*, we have* $[\![\ \mathsf{UFtoDS}\ c\ ]\!]\ Z \cong [\![\ c\ ]\!]\ Z$*.*

3.3 Coproducts of Uniform Codes

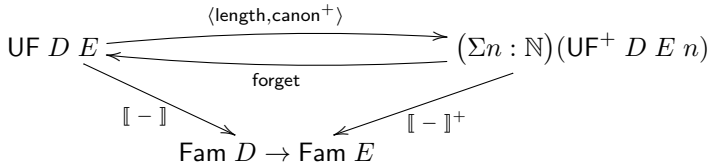
The coproduct $c +_{\mathsf{DS}} d := \sigma\ 2\ (\mathsf{tt} \mapsto c; \mathsf{ff} \mapsto d)$ of two DS codes is not in general the embedding of a uniform code, even if c and d are, as c and d may still have different shapes. Hence we cannot immediately use the same construction to define coproducts of uniform codes, but we note that whenever c and d do have the same shape, this construction still works. Our plan for constructing coproducts of uniform codes is then to find equivalent replacements of the summands, such that the new pair has a common shape, and then using the standard coproduct. To this end, we introduce an $\mathbb{N}$-indexed variant $\mathsf{UF}^+\ D\ E\ n = \big(\Sigma c : \mathsf{Uni}^+\ D\ n\big)(\mathsf{Info}^+\ c \to E)$ of UF for this section only. There are two differences between UF^+ and UF: firstly, UF^+ is indexed by the length n of its codes, and secondly in UF^+ the δ_{UF} and σ_{UF} codes are replaced by a combined code

$$\begin{aligned}
\delta\sigma : \big(c : \mathsf{Uni}^+\ D\ n\big) \to \big(A : \mathsf{Info}^+\ c \to \mathsf{Set}\big) \to \\
\big((\gamma : \mathsf{Info}^+\ c) \to A\ \gamma \to \mathsf{Set}\big) \to \mathsf{Uni}^+\ D\ (\mathsf{suc}\ n)
\end{aligned}$$

with $\mathsf{Info}^+\ (\delta\sigma\ c\ A\ B) = \big(\Sigma\gamma : \mathsf{Info}^+\ c\big)\big(\Sigma x : A\ \gamma\big)(B\ \gamma\ x \to D)$. The code $\delta\sigma$ should be thought of as a δ_{UF} code followed by a σ_{UF} code. We can recover "ordinary" σ_{UF} and δ_{UF} by $\sigma_+\ c\ A \coloneqq \delta\sigma\ c\ A\ (\_,\_ \mapsto 0)$ and $\delta_+\ c\ B \coloneqq \delta\sigma\ c\ (\_ \mapsto 1)(\gamma,\_ \mapsto B\,\gamma)$. We have just informally described translations $\mathsf{forget} : \mathsf{UF}^+\ D\ E\ n \to \mathsf{UF}\ D\ E$ and $\mathsf{canon}^+ : \big(c : \mathsf{UF}\ D\ E\big) \to \mathsf{UF}^+\ D\ E\ (\mathsf{length}\,c)$, where length counts the depth of the code c. A decoding $[\![-]\!]^+$ can be defined for UF^+ along the lines for the one for UF (alternatively, Proposition 15(ii) below can be used as a definition).

▶ **Proposition 15.** *Let $D, E : \mathsf{Set}_1$ and $Z : \mathsf{Fam}\ D$. If $c : \mathsf{UF}\ D\ E$ and $d : \mathsf{UF}^+\ D\ E\ n$, then (i) $[\![\ \mathsf{canon}^+\ c\ ]\!]^+\ Z \cong [\![\ c\ ]\!]\ Z$; and (ii) $[\![\ \mathsf{forget}\ d\ ]\!]\ Z \cong [\![\ d\ ]\!]^+\ Z$.*

This proposition can be summed up in the following commuting diagram:



Next, note $[\![\ \delta\sigma\ c\ 1\ (\_ \mapsto 0)\ ]\!]^+\ Z \cong [\![\ c\ ]\!]^+\ Z$. Thus we can pad out $c : \mathsf{UF}^+\ D\ E\ n$ to $\mathsf{pad}_k\ c : \mathsf{UF}^+\ D\ E\ (n + k + 1)$ without changing the meaning of the code:

▶ **Lemma 16.** *Let $k : \mathbb{N}$. There is an operation $\mathsf{pad}_k : \mathsf{UF}^+\ D\ E\ n \to \mathsf{UF}^+\ D\ E\ (n + k + 1)$ such that $[\![\ \mathsf{pad}_k\ c\ ]\!]^+\ Z \cong [\![\ c\ ]\!]^+\ Z$ for every $Z : \mathsf{Fam}\ D$.*

Since all UF^+ codes of the same length also are of the same shape, it is now easy to form coproducts of such codes. Define $\_ +_+ \_ : \mathsf{UF}^+\ D\ E\ n \to \mathsf{UF}^+\ D\ E\ n \to \mathsf{UF}^+\ D\ E\ (\mathsf{suc}\,n)$ by $(c, \alpha) +_+ (d, \beta) = (c +_{\mathsf{Uni}} d, [\alpha, \beta] \circ (c +_{\mathsf{Info}} d))$ where $\_ +_{\mathsf{Uni}} \_$ is defined by

$$\iota_+ +_{\mathsf{Uni}} \iota_+ = \sigma_+\ \iota_+\ (\_ \mapsto 2)$$
$$(\delta\sigma\ c\ A\ B) +_{\mathsf{Uni}} (\delta\sigma\ d\ A'\ B') = \delta\sigma\ (c +_{\mathsf{Uni}} d)\ ([A, A'] \circ (c +_{\mathsf{Info}} d))\ ([B, B'] \circ (c +_{\mathsf{Info}} d))$$

simultaneously with a map $(c +_{\mathsf{Info}} d) : \mathsf{Info}^+\ (c +_{\mathsf{Uni}} d) \to \mathsf{Info}^+\ c + \mathsf{Info}^+\ d$, whose definition is similar. Note that we did not need to consider the definition of e.g. $\iota_+ +_{\mathsf{Uni}} (\delta\sigma\ c\ A\ B)$ as these summands cannot possibly have the same length.

▶ **Lemma 17.** *For all $c, d : \mathsf{UF}^+\ D\ E\ n$ and $Z : \mathsf{Fam}\ D$ we have $[\![\ c +_+ d\ ]\!]^+\ Z \cong [\![\ c\ ]\!]^+\ Z + [\![\ d\ ]\!]^+\ Z$, where the right hand side is a coproduct of families.*

Putting everything together, we have:

▶ **Theorem 18.** *Let $D, E : \mathsf{Set}_1$. Define $\_ +_{\mathsf{UF}} \_ : \mathsf{UF}\ D\ E \to \mathsf{UF}\ D\ E \to \mathsf{UF}\ D\ E$ by $c +_{\mathsf{UF}} d = \mathsf{forget}\ (\mathsf{canon}^+\ c +_+ \mathsf{canon}^+\ d)$. Then $[\![\ c +_{\mathsf{UF}} d\ ]\!]\ Z \cong [\![\ c\ ]\!]\ Z + [\![\ d\ ]\!]\ Z$.*

3.4 Composition of uniform Codes

Recall Section 2.2, where composition of DS codes followed from a power operation which – because of the dependency arising in its attempted construction – we could not define. Fortunately, in UF, a power operator is definable! Composition is here – as for DS – facilitated by a conjunction of the power operation and a bind operator. A full bind operation for UF is not definable since the grafting of uniform trees into a uniform tree may not be uniform since the trees may differ in height (i.e. UF is not a monad). However, for composition, it suffices to graft trees of the *same* height. Define $- \gg\!=[- \longrightarrow -] : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to$

$\mathsf{Set}) \to \mathsf{Uni}\ D \to \mathsf{Uni}\ D$, together with $(c \gg\!=\![E \longrightarrow d])_{\mathsf{Info}} : \mathsf{Info}\ (c \gg\!=\![E \longrightarrow d]) \to \big(\Sigma x : \mathsf{Info}\ c\big)(E\,x \to \mathsf{Info}\ d)$, which explains the meaning of $c \gg\!=\![E \longrightarrow d]$ at the level of Info. We write $\gg\!=_{\mathsf{Info},0}$ and $\gg\!=_{\mathsf{Info},1}$ for the first and second projection of $(c \gg\!=\![E \longrightarrow d])_{\mathsf{Info}}$ respectively, inferring the other arguments from context:

$$
\begin{aligned}
&c \gg\!=\![E \longrightarrow \iota_{\mathsf{UF}}] = c\\
&c \gg\!=\![E \longrightarrow \sigma_{\mathsf{UF}}\ d\ A]\\
&\qquad = \sigma_{\mathsf{UF}}\ (c \gg\!=\![E \longrightarrow d])(\gamma \mapsto \big(e : E(\gg\!=_{\mathsf{Info},0}\ \gamma)\big) \to A(\gg\!=_{\mathsf{Info},1}\ \gamma\, e))\\
&c \gg\!=\![E \longrightarrow \delta_{\mathsf{UF}}\ d\ A] =\\
&\qquad = \delta_{\mathsf{UF}}\ (c \gg\!=\![E \longrightarrow d])(\gamma \mapsto \big(\Sigma e : E(\gg\!=_{\mathsf{Info},0}\ \gamma)\big)A(\gg\!=_{\mathsf{Info},1}\ \gamma\, e))
\end{aligned}
$$

$$
\begin{aligned}
&(c \gg\!=\![E \longrightarrow \iota_{\mathsf{UF}}])_{\mathsf{Info}}\ x = (x, (\_ \mapsto \star))\\
&(c \gg\!=\![E \longrightarrow \sigma_{\mathsf{UF}}\ d\ A])_{\mathsf{Info}}\ (x, g) = (\gg\!=_{\mathsf{Info},0}\ \ x, e \mapsto (\gg\!=_{\mathsf{Info},0}\ \ x\ e, g\ e))\\
&(c \gg\!=\![E \longrightarrow \delta_{\mathsf{UF}}\ d\ A])_{\mathsf{Info}}\ (x, g) = (\gg\!=_{\mathsf{Info},0}\ \ x, e \mapsto (\gg\!=_{\mathsf{Info},0}\ \ x\ e, (a \mapsto g\ (e, a)))
\end{aligned}
$$

This definition is validated by the following proposition:

▶ **Proposition 19.** *There is an equivalence*

$$[\![\ c \gg\!=\![E \longrightarrow d], (d \gg\!=\![E \longrightarrow d])_{\mathsf{Info}}\ ]\!] \cong ([\![\ c, \mathsf{id}\ ]\!]) \gg\!=_{\mathsf{Fam}} (e \mapsto ((E\ e) \longrightarrow_{\mathsf{Fam}} [\![\ d, \mathsf{id}\ ]\!]))$$

▶ Remark. While is not possible to derive a bind operator from $\_ \gg\!=\![\_ \longrightarrow \_]$, we do obtain a power operator with the right universal property by

$$A \longrightarrow (c, f) \coloneqq (\iota_{\mathsf{UF}} \gg\!=\![(\_ \mapsto A) \longrightarrow c], (\gamma \mapsto f \circ \gg\!=_{\mathsf{Info},1}))\ .$$

(This fact will not be needed in the proof of composition in Theorem 20.)

We can now define composition for UF codes in a fashion similar to Theorem 8, except that we separate the action of the first component of a code and take care of the second component in a second step:

▶ **Theorem 20.** *The operations*

$$
\begin{aligned}
&\_ \bullet_{\mathsf{Uni}} \_ : \mathsf{Uni}\ D \to \mathsf{UF}\ C\ D \to \mathsf{Uni}\ C\\
&(\_ \bullet_{\mathsf{Info}} \_) : \big(c : \mathsf{Uni}\ D\big) \to \big(R : \mathsf{UF}\ C\ D\big) \to \mathsf{Info}\ (c \bullet_{\mathsf{Uni}} R) \to \mathsf{Info}\ c
\end{aligned}
$$

simultaneously defined by

$$
\begin{aligned}
\iota_{\mathsf{UF}} \bullet_{\mathsf{Uni}} R &= \iota_{\mathsf{UF}}\\
(\sigma_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Uni}} R &= \sigma_{\mathsf{UF}}\ (c \bullet_{\mathsf{Uni}} R)\ (A \circ (c \bullet_{\mathsf{Info}} R))\\
(\delta_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Uni}} (d, \beta) &= (c \bullet_{\mathsf{Uni}} (d, \beta)) \gg\!=\![(A \circ (c \bullet_{\mathsf{Info}} (d, \beta))) \longrightarrow d]\\
(\iota_{\mathsf{UF}} \bullet_{\mathsf{Info}} R)\ x &= x\\
((\sigma_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Info}} R)\ (x, y) &= ((c \bullet_{\mathsf{Info}} R)\ x, y)\\
((\delta_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Info}} (d, \beta))\ x &= ((c \bullet_{\mathsf{Info}} (d, \beta))\ (\gg\!=_{\mathsf{Info},0}\ \ x, \beta \circ (\gg\!=_{\mathsf{Info},1}\ \ x)))
\end{aligned}
$$

make $\_ \bullet \_ : \mathsf{UF}\ D\ E \to \mathsf{UF}\ C\ D \to \mathsf{UF}\ C\ E$ *a composition operation for* UF *codes, where*

$$(c, \alpha) \bullet (d, \beta) = (c \bullet_{\mathsf{Uni}} (d, \beta), \alpha \circ (c \bullet_{\mathsf{Info}} (d, \beta)))\ .$$

▶ **Example 21.** If we compose c_{2W} from Example 11 with the "upgraded" code $c_{\mathsf{W}\,\mathbb{N}\,\mathsf{Fin}}$ from Example 10, we get a code for a universe where each constructor now takes a list of inductive arguments, with decoding the product of the decodings. Up to an isomorphism relating coproducts of compositions with compositions of coproducts, the resulting code is $c_{\mathsf{2W}} \bullet c_{\mathsf{W}\,\mathbb{N}\,\mathsf{Fin}} \cong c_{2,\mathsf{UF}} +_{\mathsf{UF}} c'_{W,\mathsf{UF}}$, where $c_{2,\mathsf{UF}}$ is as before, and

$$\begin{aligned} c'_{W,\mathsf{UF}} = (\delta_{\mathsf{UF}}\ (\sigma_{\mathsf{UF}}\ c_{\mathsf{W}\,\mathbb{N}\,\mathsf{Fin}}\ ((\star, n, Y) &\mapsto ((x : \mathsf{Fin}\,n) \to Y\,x) \to \mathbb{N})) \\ ((\star, n, Y, e) &\mapsto (\Sigma y : (x : \mathsf{Fin}\,n) \to Y\,x)\mathsf{Fin}\,(e\,y)), \\ ((\star, n, Y, e, B) &\mapsto (\Sigma y : (x : \mathsf{Fin}\,n) \to Y\,x)(w : \mathsf{Fin}\,(e\,y)) \to B\,(y, w))) \ . \end{aligned}$$

4 Polynomial Codes PN for Inductive-Recursive Definitions

We saw in Section 2.2 that composition for Dybjer-Setzer codes requires a power operator. However, simply adding a code for powers means that $\mathsf{DS}\ D\ \_$ is no longer a monad, and the bind operation was crucial for constructing composition. Hence, further adjustments are required. Following this line of thought results in a system including sums and type-indexed products. For this reason, we call it *polynomial inductive-recursive definitions*, and denote it by PN. It was originally invented by the second author in order to make induction-recursion resemble the descriptions of datatypes in Chapman et al. [3]. Just like uniform codes, polynomial codes are presented as a two-level definition which itself is an inductive-recursive definition:

▶ **Definition 22.** Let $D, E : \mathsf{Set}_1$. The large type $\mathsf{PN}\ D\ E : \mathsf{Set}_1$ of *polynomial codes for induction-recursion* is defined by $\mathsf{PN}\ D\ E \coloneqq (\Sigma c : \mathsf{Poly}\,\mathsf{D})(\mathsf{Info}\,\mathsf{c} \to E)$, where $\mathsf{Poly}\,\mathsf{D} : \mathsf{Set}_1$ and $\mathsf{Info} : \mathsf{Poly}\,\mathsf{D} \to \mathsf{Set}_1$ are mutually defined by

$$\begin{array}{rll} \mathsf{id}_{\mathsf{PN}} : \mathsf{Poly}\,\mathsf{D} & & \mathsf{Info}\,\mathsf{id}_{\mathsf{PN}} = D \\ \mathsf{con} : (A : \mathsf{Set}) \to \mathsf{Poly}\,\mathsf{D} & & \mathsf{Info}\,(\mathsf{con}\,\mathsf{A}) = A \\ \mathsf{sig} : (S : \mathsf{Poly}\,\mathsf{D}) \to (\mathsf{Info}\,\mathsf{S} \to \mathsf{Poly}\,\mathsf{D}) \to \mathsf{Poly}\,\mathsf{D} & & \mathsf{Info}\,(\mathsf{sig}\,\mathsf{S}\,\mathsf{F}) = (\Sigma x : \mathsf{Info}\,\mathsf{S})(\mathsf{Info}\,(\mathsf{F}\,\mathsf{x})) \\ \mathsf{pi} : (A : \mathsf{Set}) \to (A \to \mathsf{Poly}\,\mathsf{D}) \to \mathsf{Poly}\,\mathsf{D} & & \mathsf{Info}\,(\mathsf{pi}\,\mathsf{A}\,\mathsf{F}) = (x : A) \to \mathsf{Info}\,(\mathsf{F}\,\mathsf{x}) \end{array}$$

Warning: polynomial codes should not be confused with polynomial functors [9, 10]! We use the same name Info as in uniform codes for the function computing the information represented by a code. The code $\mathsf{id}_{\mathsf{PN}}$ represents the identity functor, $\mathsf{con}\ A$ the functor constantly returning index type A, $\mathsf{sig}\ S\ F$ represents a dependent coproduct of functors, and $\mathsf{pi}\ A\ F$ represents an A-indexed dependent product of functors. Observe that $\mathsf{PN}\ D\ \_$ is again, like $\mathsf{UF}\ D\ \_$, functorial by function composition.

▶ **Example 23** (W-types, again)**.** We revisit Examples 2 and 10. For $S : \mathsf{Set}$, $P : S \to \mathsf{Set}$ the polynomial code for the W-type $\mathsf{W}\ S\ P$ is $(c_{\mathsf{W}\ S\ P,\mathsf{PN}}, \_ \mapsto \star) : \mathsf{PN}\ 1\ 1$ where $c_{\mathsf{W}\ S\ P,\mathsf{PN}} = \mathsf{sig}\ (\mathsf{con}\ S)(s \mapsto\ \ \mathsf{pi}\ (P\ s)\ (\_ \mapsto \mathsf{id}_{\mathsf{PN}}))$. Again this can be upgraded to a PN Set Set code, applying $T : U \to \mathsf{Set}$ everywhere in the structure, by replacing the trivial map $(\_ \mapsto \star) : \mathsf{Info}\,\mathsf{c}_{\mathsf{W}\ \mathsf{S}\ \mathsf{P},\mathsf{PN}} \to 1$ by the map $((s, Y) \mapsto (c : P(s)) \to Y\,x) : \mathsf{Info}\,\mathsf{c}_{\mathsf{W}\ \mathsf{S}\ \mathsf{P},\mathsf{PN}} \to \mathsf{Set}$.

▶ **Example 24** (A universe closed under W-types, again)**.** We also revisit Example 3 again. A polynomial code $(c_{\mathsf{2W},\mathsf{PN}}, \alpha) : \mathsf{PN}\ \mathsf{Set}\ \mathsf{Set}$ for a universe containing 2, closed under W-types is given by $c_{\mathsf{2W},\mathsf{PN}} : \mathsf{Poly}\ \mathsf{Set}$ and $\alpha_{\mathsf{2W},\mathsf{PN}} : \mathsf{Info}\,\mathsf{c}_{\mathsf{2W},\mathsf{PN}} \to \mathsf{Set}$ where

$$c_{\mathsf{2W},\mathsf{PN}} = \mathsf{sig}\ (\mathsf{con}\ \{\mathsf{two}, \mathsf{w}\})(\mathsf{two} \mapsto \mathsf{con}\ 1; \mathsf{w} \mapsto \mathsf{sig}\ \mathsf{id}_{\mathsf{PN}}\ (X \mapsto \mathsf{pi}\ X\ (\_ \mapsto \mathsf{id}_{\mathsf{PN}})))$$

and $\alpha_{\mathsf{2W},\mathsf{PN}}$ is defined by $\alpha_{\mathsf{2W},\mathsf{PN}}(\mathsf{two}, x) = 2$ and $\alpha_{\mathsf{2W},\mathsf{PN}}(\mathsf{w}, (A, B)) = \mathsf{W}\ S\ P$.

▶ Remark. One obtains a weaker system by replacing the pi code by a code $\mathsf{pow} : \mathsf{Set} \to \mathsf{Poly}\ D \to \mathsf{Poly}\ D$ with $\mathsf{Info}\,(\mathsf{pow}\,\mathsf{A}\,\mathsf{c}) = A \to \mathsf{Info}\,\mathsf{c}$. In the full system, such a code can be defined by $\mathsf{pow}\ A\ c \coloneqq \mathsf{pi}\ A\ (\_ \mapsto c)$. The weaker system also enjoys composition, and the embedding of Dybjer-Setzer codes in Section 4.1 factors through the system with powers only. Semantically, the stronger system is just as easy to handle (see Theorem 27 below).

Polynomial codes in $\mathsf{PN}\ D\ E$ decode to functors $\mathsf{Fam}\ D \to \mathsf{Fam}\ E$ in the following way:

▶ **Definition 25.** Let $c : \mathsf{Poly}\ D$ and $\alpha : \mathsf{Info}\,\mathsf{c} \to E$. The polynomial code $(c, \alpha) : \mathsf{PN}\ D\ E$ induces a functor $[\![\ c, \alpha\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$ by $[\![\ c, \alpha\ ]\!]\ Z = \mathsf{Fam}(\alpha)\ ([\![\ c\ ]\!]_0\ Z, [\![\ c\ ]\!]_{\mathsf{info}}\ Z)$ where $[\![\ c\ ]\!]_0 : \mathsf{Fam}\ D \to \mathsf{Set}$ and $[\![\ c\ ]\!]_{\mathsf{info}} : (X : \mathsf{Fam}\ D) \to [\![\ c\ ]\!]_0\ X \to \mathsf{Info}\,\mathsf{c}$ are simultaneously defined by induction on c:

$$[\![\ \mathsf{id_{PN}}\ ]\!]_0\ (U,T) = U \qquad [\![\ \mathsf{id_{PN}}\ ]\!]_{\mathsf{info}}\ (U,T)\ x = T\,x$$
$$[\![\ \mathsf{con}\ A\ ]\!]_0\ X = A \qquad [\![\ \mathsf{con}\ A\ ]\!]_{\mathsf{info}}\ X\ a = a$$
$$[\![\ \mathsf{sig}\ S\ F\ ]\!]_0\ (U,T) = \big(\Sigma s : [\![\ S\ ]\!]_0\ (U,T)\big)([\![\ F([\![\ S\ ]\!]_{\mathsf{info}}\ (U,T)\ s)\ ]\!]_0\ (U,T))$$
$$[\![\ \mathsf{sig}\ S\ F\ ]\!]_{\mathsf{info}}\ (U,T)\ (s,x) = ([\![\ S\ ]\!]_{\mathsf{info}}\ (U,T)\ s, [\![\ F\ ([\![\ S\ ]\!]_{\mathsf{info}}\ (U,T)\ s)\ ]\!]_{\mathsf{info}}\ (U,T)\ x)$$
$$[\![\ \mathsf{pi}\ A\ F\ ]\!]_0\ X = \big(x : A\big) \to [\![\ Fx\ ]\!]_0\ X \qquad [\![\ \mathsf{pi}\ A\ F\ ]\!]_{\mathsf{info}}\ X\ g = (a \mapsto [\![\ (Fa)\ ]\!]_{\mathsf{info}}\ X\ (g\ a))$$

▶ **Example 26.** Decoding $c_{\mathsf{W}\ S\ P,\mathsf{PN}}$ from Example 23, we get

$$[\![\ c_{\mathsf{W}\ S\ P,\mathsf{PN}}\ ]\!]_0\ (U,T) = \big(\Sigma s : S\big)(P(s) \to U)$$

this time matching the domain of the W-type constructor sup strictly. Similarly decoding $(c_{\mathsf{2W},\mathsf{PN}}, \alpha_{\mathsf{2W},\mathsf{PN}})$ from Example 24 we again get the same result as in Example 6.

Since we did not exhibit PN as a subsystem of DS, we cannot rely on Dybjer and Setzer's proof of soundness, i.e. that initial algebras of the corresponding functors exist in their model. We can, however, extend their proof to polynomial codes[2]:

▶ **Theorem 27.** *Working in ZFC, assume the existence of a Mahlo cardinal* M *and a 1-inaccessible cardinal* I *above it. Then there is a set-theoretic model of Martin-Löf Type Theory +* PN *where types* $A : \mathsf{Set}$ *are interpreted as sets in* V_{M} *and large types* $D : \mathsf{Set}_1$ *are interpreted as sets in* V_{I} *(here* V_α *is the cumulative hierarchy). In this model, all functors* $[\![\ c\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ D$ *arising from polynomial codes* $c : \mathsf{PN}\ D\ D$ *have initial algebras.*

Note that the existence of large cardinals is only needed for the soundness proof, and not for working within the theory itself. The same situation applies to Dybjer and Setzer's DS.

4.1 Embedding of DS into PN

▶ **Proposition 28.** *The map* $\mathsf{DStoPN} : \mathsf{DS}\ D\ E \to \mathsf{PN}\ D\ E$ *given by* $\mathsf{DStoPN}\,c = (\mathsf{toP}\,c, \mathsf{tol}\,c)$ *where* $\mathsf{toP} : \mathsf{DS}\ D\ E \to \mathsf{Poly}\,\mathsf{D}$ *and* $\mathsf{tol} : \big(c : \mathsf{DS}\ D\ E\big) \to \mathsf{Info}\,(\mathsf{toP}\,\mathsf{c}) \to E$ *are defined by*

$$\mathsf{toP}(\iota\ e) = \mathsf{con}\,1 \qquad \mathsf{tol}(\iota\ e)\star = e$$
$$\mathsf{toP}(\sigma\ A\ f) = \mathsf{sig}\,(\mathsf{con}A)\,(\mathsf{toP} \circ f) \qquad \mathsf{tol}(\sigma\ A\ f)\,(a,x) = \mathsf{tol}\,(f\ a)\,x$$
$$\mathsf{toP}(\delta\ A\ F) = \mathsf{sig}\,(\mathsf{pi}\,A\,(\_ \mapsto \mathsf{id_{PN}}))\,(\mathsf{toP} \circ F) \qquad \mathsf{tol}(\delta\ A\ F)\,(g,x) = \mathsf{tol}\,(F\ g)\,x$$

is semantics-preserving.

We conjecture that this embedding is strict, i.e. that there is a code $c : \mathsf{PN}\ D\ E$ with $[\![\ c\ ]\!] \not\simeq [\![\ \mathsf{DStoPN}\ d\ ]\!]$ for every $d : \mathsf{DS}\ D\ E$ for some $D, E : \mathsf{Set}_1$.

[2] We require a little bit more from the metatheory: Dybjer and Setzer [8] require I to be 0-inaccessible only. But existence of I is a mild assumption compared to the existence of M.

4.2 Composition of Polynomial Codes

Composition for PN codes can be defined following the same pattern as in Proposition 8, where we constructed composition for DS codes using the assumption of a power operation, and the fact that DS is a monad. The system PN has a power operation using the pi constructor, and is a monad thanks to the sig constructor:

▶ **Proposition 29.** *For each* $D : \mathsf{Set}_1$, $\mathsf{PN}\ D$ *is a monad, i.e. there are terms* $\eta_{\mathsf{PN}} : E \to \mathsf{PN}\ D\ E$ *and* $\mu_{\mathsf{PN}} : \mathsf{PN}\ D\ (\mathsf{PN}\ D\ E) \to \mathsf{PN}\ D\ E$ *satisfying the monad laws. Furthermore, let* $(U,T) : \mathsf{Fam}\ D$. *Then* $[\![\ \eta_{\mathsf{PN}}(e)\ ]\!](U,T) = \eta_{\mathsf{Fam}}(e)$ *for every* $e : E$ *and , and* $[\![\ \mu_{\mathsf{PN}}(c)\ ]\!](U,T) = \mu_{\mathsf{Fam}}(\mathsf{Fam}([\![\ -\ ]\!](U,T))([\![\ c\ ]\!](U,T)))$ *for every* $c : \mathsf{PN}\ D\ (\mathsf{PN}\ D\ E)$.

Proof. We define $\eta_{\mathsf{PN}}(e) = (\mathsf{con}\ 1, \_ \mapsto e)$ and $\mu_{\mathsf{PN}}(c,\alpha) = (\mathsf{sig}\ c\ (\mathsf{fst} \circ \alpha), (x,y) \mapsto \mathsf{snd}\ (\alpha\ x)\ y)$. The equations in terms of the monad structure on Fam holds on the nose. ◀

Using the monad structure, we can define a "dependent bind" operation

$$\begin{aligned} \_ \gg\!=_{\mathsf{PN}} \_ &: \mathsf{PN}\ C\ D \to ((x : D) \to \mathsf{PN}\ C\ (E\,x)) \to \mathsf{PN}\ C\ ((\Sigma x : D)(E\,x)) \\ c \gg\!=_{\mathsf{PN}} h &= \mu_{\mathsf{PN}}(\mathsf{PN}(x \mapsto \mathsf{PN}(y \mapsto (x,y)))\ (h\,x)\ c) \end{aligned}$$

We also note that the pi constructor can be packaged up into the following "dependent power" operation for $S : \mathsf{Set}$ and $E : A \to \mathsf{Set}_1$:

$$\begin{aligned} \pi_{\mathsf{PN}} A &: (a : A) \to \mathsf{PN}\ D\ (E\,a) \to \mathsf{PN}\ D\ ((a : A) \to (E\,a)) \\ \pi_{\mathsf{PN}}\ A\ f &= (\mathsf{pi}\ A\ (\mathsf{fst} \circ f), (g \mapsto (a \mapsto \mathsf{snd}\,(f a)\ (g a)))) \end{aligned}$$

Using these ingredients, we can now define composition of PN codes:

▶ **Theorem 30.** *For* $c : \mathsf{Poly}\ D$ *and* $\alpha : \mathsf{Info}\,c \to E$ *and* $R : \mathsf{PN}\ C\ D$, *define* $(c,\alpha) \bullet R = \mathsf{PN}(\alpha)\,(c/R) : \mathsf{PN}\ C\ E$, *where* $\_/\_ : (c : \mathsf{Poly}\ E) \to \mathsf{PN}\ D\ E \to \mathsf{PN}\ D(\mathsf{Info}\,c)$ *is defined by*

$$\begin{aligned} \mathsf{id}_{\mathsf{PN}}/R &= R & (\mathsf{sig}\ c\ f)/R &= (c/R) \gg\!=_{\mathsf{PN}} (p \mapsto (f\,p)/R) \\ (\mathsf{con}\ A)/R &= (\mathsf{con} A, \mathsf{id}) & (\mathsf{pi}\ A\ f)/R &= \pi_{\mathsf{PN}}\ A\ (a \mapsto (f a)/R) \end{aligned}$$

Then $[\![\ R \bullet Q\ ]\!]\,(U,T) \cong [\![\ R\ ]\!]\,([\![\ Q\ ]\!]\,(U,T))$.

▶ **Example 31.** Let us compose $c_{\mathsf{2W,PN}}$ from Example 24 with the "upgraded" code $c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin,PN}}$ from Example 23. This time we get the code $\mathsf{sig}\ (\mathsf{con}\ \{\mathsf{two}, \mathsf{w}\})\ f$, where $f\ \mathsf{two} = \mathsf{con}\ 1$ and $f\ \mathsf{w} = \mathsf{sig}\ c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin,PN}}\ ((n,Y) \mapsto \mathsf{pi}\ ((x : \mathsf{Fin}\,n) \to (Y\,x))\ (\_ \mapsto c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin,PN}}))$.

5 Conclusions

Inductive-recursive definitions arise as initial algebras of endofunctors on Fam D, but the question of exactly which functors does not have a canonical answer. Dybjer and Setzer [7] gave one axiomatisation DS, which was adequate in the sense that it covered all examples "in the wild", and all functors represented in it could be shown to have initial algebras (in a sufficiently strong metatheory). We have presented two alternative axiomatisations UF and PN that retain these properties, but in addition are closed under composition. This opens up the field to find the *optimal* axiomatisation of inductive-recursive definitions. As a start, we hope to show in future work that both inclusions $\mathsf{UF} \hookrightarrow \mathsf{DS} \hookrightarrow \mathsf{PN}$ are strict.

Acknowledgements. We would like to thank Peter Hancock and Anton Setzer for inspiration and interesting discussions, and the reviewers for their suggestions and comments.

References

1 Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Containers: Constructing strictly positive types. *TCS*, 342(1):3 – 27, 2005.

2 Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris. Indexed containers. *Journal Functional Programming*, 25, 2015.

3 James Chapman, Pierre-Évariste Dagand, Conor McBride, and Peter Morris. The gentle art of levitation. In *ICFP 2010*, pages 3–14, 2010.

4 Peter Dybjer. Inductive families. *Formal aspects of computing*, 6(4):440–465, 1994.

5 Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2), 2000.

6 Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In *TLCA*, pages 129–146. Springer Verlag, 1999.

7 Peter Dybjer and Anton Setzer. Induction–recursion and initial algebras. *Annals of Pure and Applied Logic*, 124(1-3):1–47, 2003.

8 Peter Dybjer and Anton Setzer. Indexed induction–recursion. *Journal of logic and algebraic programming*, 66(1):1–49, 2006.

9 Nicola Gambino and Martin Hyland. Wellfounded trees and dependent polynomial functors. In *Types for Proofs and Programs*, pages 210–225, 2004.

10 Nicola Gambino and Joachim Kock. Polynomial functors and polynomial monads. *Mathematical Proceedings of the Cambridge Philosophical Society*, 154:153–192, 2013.

11 Neil Ghani and Peter Hancock. Containers, monads and induction recursion. *Mathematical Structures in Computer Science*, 26(1):89–113, 2016.

12 Peter Hancock. Private communication.

13 Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975.

14 Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1984.

15 Robert Pollack. Dependently typed records in type theory. *Formal Aspects of Computing*, 13(3):386–402, 2002.

One-Dimensional Logic over Trees*

Emanuel Kieroński[1] and Antti Kuusisto[2]

1 **University of Wrocław, Poland**
kiero@cs.uni.wroc.pl
2 **University of Bremen, Germany**
kuusisto@uni-bremen.de

Abstract

A one-dimensional fragment of first-order logic is obtained by restricting quantification to blocks of existential quantifiers that leave at most one variable free. This fragment contains two-variable logic, and it is known that over words both formalisms have the same complexity and expressive power. Here we investigate the one-dimensional fragment over trees. We consider unranked unordered trees accessible by one or both of the descendant and child relations, as well as ordered trees equipped additionally with sibling relations. We show that over unordered trees the satisfiability problem is ExpSpace-complete when only the descendant relation is available and 2-ExpTime-complete with both the descendant and child or with only the child relation. Over ordered trees the problem remains 2-ExpTime-complete. Regarding expressivity, we show that over ordered trees and over unordered trees accessible by both the descendant and child the one-dimensional fragment is equivalent to the two-variable fragment with counting quantifiers.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases satisfiability, expressivity, trees, fragments of first-order logic

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.64

1 Introduction

One-dimensional fragment of first-order logic, F_1, is obtained by restricting quantification to blocks of existential quantifiers that leave at most one variable free (as the logic is closed under negation and boolean operations one may also use blocks of universal quantifiers). It is not difficult to show that over general relational structures the satisfiability problem for F_1 is undecidable [8]. In such situations, there are two standard ways of regaining decidability. One can either try to impose some additional restrictions on the syntax of the considered logic or to restrict attention to some specific classes of structures. Both approaches have been tried in the context of F_1.

A nice syntactic restriction of F_1 which turns out to be decidable over general structures is called a *uniform* one-dimensional fragment, UF_1. It was introduced in [8] as a generalization of the two-variable fragment of first-order logic, FO^2, to contexts with relations of arity higher than two, e.g., databases. The readers interested in this variant are referred to [8], [11], [12] and a survey [14] which also reveals some connections with description logics.

Let us turn to the restricted classes of structures. There are two important first-choice options, well motivated in various areas of computer science, namely the class of words and the class of trees. F_1 over words and ω-words is investigated in [10]. The satisfiability

* E.K. was supported by the Polish National Science Centre grant No. 2016/21/B/ST6/01444. A.K. was supported by the ERC grant 647289 CODA.

problem is shown to be NExpTime-complete, exactly as in the case of FO^2 [7]. Moreover, over words F_1 and FO^2 turn out to share the same expressive power. The advantage of F_1 over FO^2 is that the former allows us to express some properties in a more natural way (and seems to be more succinct, which is however not formally proved). There are a few other related formalisms over words, worth mentioning here. In [7] it is shown that FO^2 is expressively equivalent to unary temporal logic, UTL, i.e., temporal logic with four navigational operators: *next state*, *somewhere in the future*, *previous state*, *somewhere in the past*. FO^2, however, is exponentially more succinct than UTL. The satisfiability problem for UTL is PSpace-complete. An extension of FO^2 by counting quantifiers, C^2, is shown to be NExpTime-complete over words in [5]. In fact, it is not difficult to observe that over words C^2 has the same expressive power as plain FO^2 (but, again, the former is more succinct). Another interesting extension of FO^2 over words, this time significantly increasing its expressive power, is an extension by the *between* predicate recently studied by in [13]. Its satisfiability problem is ExpSpace-complete.

Turning now to the class of trees, both FO^2 and C^2 retain a reasonable complexity, namely their satisfiability problems over trees are ExpSpace-complete. See [2] for the analysis of FO^2 over trees and [1] for its extension covering C^2. Regarding the expressive power, the situation depends on the type of the trees considered. In the case of *unordered* trees FO^2 cannot count and is less expressive than C^2. Over *ordered* trees both formalisms are equally expressive [1] and share the expressiveness with the navigational core of XPath temporal logic (cf. [15]). The importance of FO^2 and C^2 over trees is also justified by the fact that they are located close to the border between elementary and non-elementary, e.g., adding the third variable makes the satisfiability problem very hard: still decidable, but as shown in [18] necessarily with a non-elementary complexity.

In this paper we investigate the computational complexity and the expressive power of F_1 over trees. We consider finite unranked trees accessible by a few navigational signatures: just the descendant relation; just the child relation; both the descendant and the child relations; and, finally, the descendant relation, the child relation plus the next-sibling and following-sibling relations. Concerning the complexity of satisfiability, it depends on whether the child relation is present or not. With the child relation the satisfiability problem is 2-ExpTime-complete, and without it is ExpSpace-complete. To show the complexity results we perform some surgery on models leading to small model properties, and then design algorithms searching for such appropriate small models. Technically, we extend the approach from [4] used there in the context of FO^2. Roughly speaking we appropriately abstract the information about a node by its *profile* (an analogous notion is called *a full type* in [4]) and then we either contract nodes with the same profiles, or delete some nodes whose sufficiently many profiles are realized in a (fragment) of a model. Worth mentioning is that an orthogonal extension of the method from [4] is used in [1] in the context of C^2. In both cases the challenge is to carefully tune the notion of a profile (full type) in order to get the optimal complexity. Regarding expressivity, we argue that over ordered trees with all of the four navigational relations we consider, F_1 is expressively equivalent to C^2 and FO^2. We also show that over unordered trees equipped with both the descendant and the child relation F_1 is still equivalent to C^2 (but this time the latter is known to be more expressive than FO^2). While the former equivalence is rather easy to see (though slightly awkward to formally show), the latter is less obvious and more difficult to prove. In our expressivity studies we do not consider the cases of unordered trees accessible by only one of the descendant and the child relations. However, we conjecture that also under these scenarios F_1 is equivalent to C^2. We leave the related investigations for the full version of this paper.

In the case of trees, as in the case of F_1 over words, the advantage of F_1 over FO^2 and C^2 is that it allows to specify some properties in a more natural and elegant way. If we want to say that a tree contains some (especially not fully specified) pattern, consisting of more than two elements, we can just quantify an appropriate number of positions, and say how they should be labelled and related to each other. Expressing the same in FO^2 (if possible), and usually also in C^2, will very likely require some heavy recycling of the two available variables and a careful navigation over trees. Let us just recall a simple example from [10], which in fact does not even use the navigational relations. The formula $\exists xyz \bigwedge_{i=1}^{n}(P_i(x) \vee P_i(y) \wedge P_i(z))$ says that there are three points which together ensure that each unary properties $P_1, \ldots, P_n$ appears in a model. A reader is asked to check that complicated and long formulas arise when we try to express the same in FO^2 or even in C^2 over (ordered or unordered) trees.

The rest of the paper is organized as follows. In Section 2 we define the logic and structures we are interested in and introduce some tools which will be then used in the following sections. In Section 3 we perform some surgery on trees, which then, in Section 4, allows us to establish the exact complexity bounds under all the considered navigational scenarios. In Section 5 we consider expressivity issues, relating F_1 over trees with C^2, FO^2 and GF^2, and finally in Section 6 we conclude the paper.

2 Preliminaries

2.1 Trees and logics

We work with signatures of the form $\tau = \tau_0 \cup \tau_{bin}$, where τ_0 is a set of unary symbols and $\tau_{bin} \subseteq \{\downarrow, \downarrow_+, \rightarrow, \rightarrow^+\}$ is a set of *navigational* binary symbols. Over such signatures we consider the *one-dimensional* fragment of first-order logic F_1, that is the relational fragment in which quantification is restricted to blocks of existential quantifiers that leave at most one variable free. Formally, F_1 over relational signature τ and some countably infinite set of variables Var is the smallest set such that:

- $R\bar{x} \in \mathrm{F}_1$ for all $R \in \tau$ and all tuples $\bar{x}$ of variables from Var of the appropriate length,
- $x = y \in \mathrm{F}_1$ for all variables $x, y \in Var$,
- F_1 is closed under $\vee$ and $\neg$,
- if φ is an F_1 formula with free variables $x_0, \ldots, x_k$ then formulas $\exists x_0, \ldots, x_k \varphi$ and $\exists x_1, \ldots, x_k \varphi$ belong to F_1.

As usually, we can use standard abbreviations for other Boolean operations, like $\wedge, \rightarrow, \top$, etc., as well as for universal quantification. The length of a formula φ is measured in a natural way, and denoted $\|\varphi\|$. The *width* of a formula is the maximum of the numbers of free variables in its subformulas.

For a given formula φ we denote by $\tau_0(\varphi)$ the set of unary symbols that appear in φ. We write $\mathrm{F}_1[\tau_{bin}]$ to denote that the only binary symbols that are allowed are those from τ_{bin}.

We are interested in finite unranked tree structures, in which the interpretation of symbols from τ_{bin} is fixed: if available in the signature, $\downarrow$ is interpreted as the child relation, $\rightarrow$ as the right sibling relation, and $\downarrow_+$ and $\rightarrow^+$ as their respective transitive closures. If at least one of $\rightarrow$, $\rightarrow^+$ is interpreted in a tree then we say that this tree is *ordered*; in the opposite case we say that the tree is *unordered*. In this paper we investigate the four navigational signatures, namely, in the case of unordered trees, we consider accessing them only by the descendant relation ($\mathrm{F}_1[\downarrow_+]$), only by the child relation ($\mathrm{F}_1[\downarrow]$), and by both of them ($\mathrm{F}_1[\downarrow_+, \downarrow]$); in the case of ordered trees we consider just the full signature ($\mathrm{F}_1[\downarrow, \downarrow_+, \rightarrow, \rightarrow^+]$).

We use symbol $\mathfrak{T}$ (possibly with sub- or superscripts) to denote tree structures. For a given tree $\mathfrak{T}$ we denote by T its universe. If a is a node of $\mathfrak{T}$ then we denote by $\mathfrak{T}_a^{\downarrow}$ the subtree

of $\mathfrak{T}$ rooted at a, by $\mathfrak{T}_a^{\uparrow}$ the tree obtained from $\mathfrak{T}$ by removing all subtrees rooted at the children of a. Additionally, in the case of ordered trees we denote by $\mathfrak{T}_a^{\leftarrow}$ the substructure (which usually is not a tree) of $\mathfrak{T}$ generated by the nodes of subtrees rooted at a and all its left siblings (nodes a' such that $\mathfrak{T} \models a' \rightarrow^+ a$), and, symmetrically, we denote by $\mathfrak{T}_a^{\rightarrow}$ the substructure of $\mathfrak{T}$ generated by the nodes of subtrees rooted at a and all its right siblings.

2.2 Normal form

We adapt here the well known Scott normal form for FO^2 [16] to our purposes. We say that an $\mathrm{F}_1[\tau_{bin}]$ formula φ is in *normal form* if φ has the following shape:

$$\bigwedge_{1 \leq i \leq m_\exists} \forall y_0 \exists y_1 \dots y_{k_i} \varphi_i^\exists \wedge \bigwedge_{1 \leq i \leq m_\forall} \forall x_1 \dots x_{l_i} \varphi_i^\forall, \tag{1}$$

where $\varphi_i^\exists = \varphi_i^\exists(y_0, y_1, \dots, y_{k_i})$ and $\varphi_i^\forall = \varphi_i^\forall(x_1, \dots, x_{l_i})$ are quantifier-free. Please note that the width of φ is the maximum of the set $\{k_i + 1\}_{1 \leq i \leq m_\exists} \cup \{l_j\}_{1 \leq j \leq m_\forall}$. The following fact can be proved in a standard fashion, see, e.g., [6] for a more detailed exposition of the technique.

▶ **Lemma 1.** *For every $\mathrm{F}_1[\tau_{bin}]$ formula φ, one can compute in polynomial time an $\mathrm{F}_1[\tau_{bin}]$ formula φ' in normal form (over the signature extended by some fresh unary symbols) such that for trees of size (number of nodes) equal at least to the width of φ: (i) any model of φ can be expanded to a model of φ' by appropriately interpreting fresh unary symbols; (ii) any model of φ' restricted to the signature of φ is a model of φ.*

Proof. (Sketch) We successively replace innermost subformulas ψ of φ of the form $\exists y_1, \dots, y_k$ $\varphi(y_0, y_1, \dots, y_k)$ by atoms $P_\psi(y_0)$, where P_ψ is a fresh unary symbol, and axiomatize P_ψ using two normal form conjuncts: $\forall y_0 \exists y_1, \dots, y_k (P_\psi(y_0) \rightarrow \varphi(y_0, y_1, \dots, y_k))$ and $\forall y_0, y_1, \dots, y_k$ $(\neg\varphi(y_0, y_1, \dots, y_k) \vee P_\psi(y_0))$. ◀

Lemma 1 allows us, when dealing with satisfiability or when analysing the size and shape of models, to restrict attention to normal form formulas (models of size smaller than the width of the considered formula can be easily treated separately).

2.3 Types and profiles

In this subsection we prepare some notions useful in the rest of this paper.

2.3.1 Types

For $k \in \mathbb{N} \setminus \{0\}$ a *k-type* (or a *type of size k*) π over a signature $\tau = \tau_0 \cup \tau_{bin}$ is a set of literals over variables $x_1, \dots, x_k$ (often identified with a conjunction of its elements) such that

- for each $P \in \tau_0$ and $1 \leq i \leq k$ either Px_i or $\neg Px_i$ belongs to π
- for each $\rightleftharpoons \in \tau_{bin}$ and $1 \leq i, j, \leq k$, $i \neq j$ either $x_i \rightleftharpoons x_j$ or $\neg x_i \rightleftharpoons x_j$ belongs to π
- for each $1 \leq i < j \leq k$ the inequality $x_i \neq x_j$ belongs to π
- π is satisfiable in a tree, i.e., there exists a tree $\mathfrak{T}$ containing nodes $a_1, \dots, a_k$ such that $\mathfrak{T} \models \pi(a_1, \dots, a_k)$

In this paper we will only be interested in k-types over signatures containing $\downarrow_+$. If for some variable x_i and all $j \neq i$ we have that $x_i \downarrow_+ x_j \in \pi$ then we call x_i the *root* of π. If for some variable x_i and all $j \neq i$ either $x_j \downarrow_+ x_i \in \pi$ or $x_i \downarrow_+ x_j \notin \pi$ and $x_j \downarrow_+ x_i \notin \pi$ then we call x_i a *leaf* of π. Additionally for signatures containing horizontal relations: If for some variable x_i

and all $j \neq i$ either $x_i \rightarrow^+ x_j \in \pi$ or there is h such that $x_i \rightarrow^+ x_h, x_h \downarrow_+ x_j \in \pi$ then we call x_i the *leftmost element* of π. Analogously we define the *rightmost element* of π.

Note that a k-type may have at most one root, one leftmost element and one rightmost element, but many leaves. A *type* is a k-type for some $k \geq 1$.

We say that a tuple of distinct nodes $a_1 \ldots, a_k$ of a tree $\mathfrak{T}$ *realizes* a k-type π if $\mathfrak{T} \models \pi[a_1, \ldots, a_k]$. In this case we write $\text{type}^{\mathfrak{T}}(a_1, \ldots, a_k) = \pi$.

For a given k-type π and a sequence of variables $x_{i_1}, \ldots, x_{i_k}$ we denote by $\pi(x_{i_1}, \ldots, x_{x_k})$ the result of the simultaneous substitution $x_j \leftarrow x_{i_j}$ in π. Note that in this operation $i_1, \ldots, i_k$ is not required to be a permutation of $1, \ldots, k$.

2.3.2 Subtypes

Observe that a 1-type is completely determined by a subset of τ_0. We denote by $\pi \restriction x_i$ the 1-type obtained by restricting π to literals over x_i and then replacing in them x_i by x_1. More generally, for distinct indices $i_1, \ldots, i_l \in \{1, \ldots, k\}$ we denote by $\pi \restriction [x_{i_1} \ldots x_{i_l}]$ the l-type obtained by restricting π to literals over $x_{i_1}, \ldots, x_{i_l}$, and then replacing in them x_{i_j} by x_j (for $j = 1, \ldots, l$). We say that $\pi \restriction [x_{i_1} \ldots x_{i_l}]$ is a *subtype* of π; if $i_1 = 1$ then such a subtype is called an *initial subtype* of π. Initial subtypes may be formed with $l = k$ and are called *rearrangement* of π in this case. We say that a set $\mathcal{C}$ of types is *closed under initial subtypes* if for any k-type $\pi \in \mathcal{C}$ and any distinct $i_2, \ldots, i_l \in \{2, \ldots, k\}$, we have $\pi \restriction [x_1, x_{i_2}, \ldots, x_{i_l}] \in \mathcal{C}$.

2.3.3 Profiles

Profiles are intended to abstract the information about a node in a tree. Namely, they say what are the types of tuples (of some bounded size) containing the given element. For convenience we will separately store the types of tuples built of the nodes *above* and *below* the given element.

A *k-profile* over a signature τ (containing $\downarrow_+$) is a tuple $(\alpha, \mathcal{A}, \mathcal{B})$ such that:

- α is a 1-type,
- $\mathcal{A}$ is a set of types closed under initial subtypes, such that for all $\pi \in \mathcal{A}$: (i) π is of size at most k; (ii) $\pi \restriction x_1 = \alpha$ and (iii) x_1 is a leaf of π,
- $\mathcal{B}$ is a set of types closed under initial subtypes, such that for all $\pi \in \mathcal{B}$: (i) π is of size at most k; (ii) $\pi \restriction x_1 = \alpha$ and (iii) x_1 is the root of π.

Given a profile θ we will sometimes refer to its components as $\theta.\alpha$, $\theta.\mathcal{A}$ and $\theta.\mathcal{B}$.

In the case of the signature $\{\downarrow, \downarrow_+, \rightarrow, \rightarrow^+\}$ we also consider *horizontal* k-profiles, i.e., tuples of the form $(\alpha, \mathcal{A}, \mathcal{B}, \mathcal{A}_L, \mathcal{A}_R)$, extending k-profiles in such a way that:

- $\mathcal{A}_L$ is a set of types closed under the initial subtypes, such that for all $\pi \in \mathcal{A}_L$: (i) π is of size at most k, (ii) $\pi \restriction x_1 = \alpha$ and (iii) x_1 is the leftmost element of π,
- $\mathcal{A}_R$ is a set of types closed under the initial subtypes, such that for all $\pi \in \mathcal{A}_L$: (i) π is of size at most k, (ii) $\pi \restriction x_1 = \alpha$ and (iii) x_1 is the rightmost element of π,

By simple calculations we get:

▶ **Claim 2.** *For any navigational signature τ_{bin} we have:*

(i) *The number of k-types over $\tau = \tau_0 \cup \tau_{bin}$ is bounded exponentially in $|\tau_0|$ and k. In particular, there are $2^{|\tau_0|}$ 1-types.*

(ii) *The number of k-profiles and horizontal k-profiles over $\tau_0 \cup \tau_{bin}$ are bounded doubly exponentially in $|\tau_0|$ and k.*

We denote by $\mathrm{prof}_k^{\mathfrak{T}}(a)$ the k-profile *realized* by a in $\mathfrak{T}$, i.e., the profile $(\alpha, \mathcal{A}, \mathcal{B})$ such that:

- α is the 1-type of a,
- $\mathcal{A}$ is the set of types of size at most k realized by tuples $a_1, a_2, \ldots, a_l \in T_a^{\uparrow}$ such that $a_1 = a$,
- $\mathcal{B}$ is the set of types of size at most k realized by tuples $a_1, a_2, \ldots, a_l \in T_a^{\downarrow}$ such that $a_1 = a$.

An element $a \in T$ *realizes* a horizontal k-profile $(\alpha, \mathcal{A}, \mathcal{B}, \mathcal{A}_L, \mathcal{A}_R)$ if it realizes $(\alpha, \mathcal{A}, \mathcal{B})$ and

- $\mathcal{A}_L$ is the set of types of size at most k realized by tuples $a_1, a_2, \ldots, a_l \in T_a^{\leftarrow}$ such that $a_1 = a$,
- $\mathcal{A}_R$ is the set of types of size at most k realized by tuples $a_1, a_2, \ldots, a_l \in T_a^{\rightarrow}$ such that $a_1 = a$.

Note that in realized horizontal profiles it is also the case that both $\mathcal{A}_L$ and $\mathcal{A}_R$ are subsets of $\mathcal{A}$. Also all of $\mathcal{A}, \mathcal{B}, \mathcal{A}_L, \mathcal{A}_R$ are closed under initial subsets.

In the sequel, whenever a formula $\varphi \in \mathrm{F}_1[\tau_{bin}]$ is fixed we silently assume that all k-types, k-profiles, horizontal-k-profiles and all structures considered are over the signature $\tau_{bin} \cup \tau_0(\varphi)$.

Let us consider a k-profile $\theta = (\alpha, \mathcal{A}, \mathcal{B})$. We say that an l-type π $(1 \leq l \leq k)$ is *implicit* in θ if π is a member of $\mathcal{A}$ or $\mathcal{B}$ or if there are l_1-type $\pi_1 \in \mathcal{A}$, l_2-type $\pi_2 \in \mathcal{B}$, $l_1 + l_2 - 1 = k$, such that π is the unique l-type containing $\pi_1 \cup \pi_2(x_1, x_{l_1+1}, x_{l_1+2}, \ldots, x_{l_1+l_2-1})$, Note that if for some $2 \leq i \leq l_1$ we have $x_i{\downarrow_+}x_1 \in \pi_1$ then $x_i{\downarrow_+}x_j \in \pi$ for all $j \geq l_1$ and if $x_i{\downarrow_+}x_1 \notin \pi$ then also $x_j{\downarrow_+}x_1 \notin \pi$ for all $j \geq l_1$. Intuitively, an l-type π is implicit in θ if in any tree in which θ is realized by a node a there is a tuple of nodes starting with a realizing π.

Let φ be a normal form formula of width n. Given an n-profile θ one can easily see if its any realization in any tree $\mathfrak{T}$ have all the witnesses required by $\forall\exists...\exists$ conjuncts of φ, and if each tuple of nodes of $\mathfrak{T}$ containing a cannot violate any universal conjunct of φ. Formally, we say that an n-profile θ is *φ-admissible* if

1. for every conjunct of φ of the form $\forall y_0 \exists y_1 \ldots y_{k_i} \varphi_i^{\exists}(y_0, \ldots, y_{k_i})$ there is a k-type π $(k \leq k_i)$ implicit in θ and a function $h : \{y_1, \ldots, y_{k_i}\} \to \{x_1, \ldots, x_{l_i}\}$, such that $\pi \models \varphi_i^{\exists}[y_0/x_1, y_1/h(y_1), \ldots, y_{l_i}/h(y_{k_i})]$.
2. for every conjunct of φ of the form $\forall y_1 \ldots y_{l_i} \varphi_i^{\forall}(y_1, \ldots, y_{l_i})$ any k-type $(k \leq l_i)$ implicit in θ and any function $h : \{y_1, \ldots, y_{l_i}\} \to \{x_1, \ldots, x_k\}$ having x_1 in its image we have $\pi \models \varphi_i^{\forall}[y_1/h(y_1), \ldots, y_1/h(y_1), \ldots, y_{l_i}/h(y_{k_i})]$

It is then straightforward to see the following.

▶ **Lemma 3.** *Let φ be a normal form formula of width n. Then $\mathfrak{T} \models \varphi$ iff all n-profiles realized in $\mathfrak{T}$ are φ-admissible.*

In our decision procedures we will sometimes check admissibility of profiles. Since the number of types implicit in a profile is bounded polynomially this can be done (relatively) easily.

▶ **Claim 4.** *Given a normal form F_1 formula φ of width n and an n-profile θ it can be checked in nondeterministic polynomial time (in $\|\varphi\|$ and $|\theta|$) whether θ is φ-admissible.*

In the next section we perform some surgery on models of a normal form formula φ. Namely, we remove some nodes and sometimes change the connections among the remaining nodes. Observing that the n-profiles of the surviving nodes are not changed we will then be able to conclude (thanks to Lemma 3) that the resulting trees are still models of φ.

3 Pruning trees

We now show that when looking for models of a formula φ one can restrict attention to models with bounded depth and degree. We obtain an exponential bound on the depth in the case of signatures not containing $\downarrow$, and a doubly exponential bound when $\downarrow$ is present. The bound on the degree is exponential for unordered trees and doubly exponential for ordered trees. We remark that all the above bounds are essentially optimal.

3.1 Bounded paths

The crucial observation here is that one can remove the fragment of a model between two nodes having the same profile.

▶ **Lemma 5.** *Let τ_{bin} be any of the navigational signatures $\{\downarrow\}$, $\{\downarrow_+\}$, $\{\downarrow,\downarrow_+\}$, $\{\downarrow,\downarrow_+,\rightarrow,\rightarrow^+\}$. Let φ be a normal form $\mathrm{F}_1[\tau_{bin}]$ formula of width n. Let $\mathfrak{T} \models \varphi$ and let $a, b \in T$ be two nodes such that $\mathfrak{T} \models a\downarrow_+ b$ and $\mathrm{prof}_n^{\mathfrak{T}}(a) = \mathrm{prof}_n^{\mathfrak{T}}(b)$. Let $\mathfrak{T}'$ be the tree obtained from $\mathfrak{T}$ by replacing the subtree rooted at a by the subtree rooted at b. Then for any $c \in T'$ we have that $\mathrm{prof}_n^{\mathfrak{T}'}(c) = \mathrm{prof}_n^{\mathfrak{T}}(c)$. In consequence $\mathfrak{T}' \models \varphi$.*

Proof. Consider the case when c belongs to $\mathfrak{T}'^{\downarrow}_b$ (possibly $c = b$). Since $\mathfrak{T}'^{\downarrow}_b = \mathfrak{T}^{\downarrow}_b$ then in particular $\mathfrak{T}'^{\downarrow}_c = \mathfrak{T}^{\downarrow}_c$ and it is clear that $\mathrm{prof}_n^{\mathfrak{T}'}(c).\mathcal{B} = \mathrm{prof}_n^{\mathfrak{T}}(c).\mathcal{B}$.

To see that $\mathrm{prof}_n^{\mathfrak{T}'}(c).\mathcal{A} \subseteq \mathrm{prof}_n^{\mathfrak{T}}(c).\mathcal{A}$ consider any $\pi \in \mathrm{prof}_n^{\mathfrak{T}'}(c).\mathcal{A}$. Take a realization of π in $\mathfrak{T}'$ starting with c. Let $c, b_1, \ldots, b_k, a_1, \ldots, a_l$ be a list of all nodes of this realization, such that $b_1, \ldots, b_k \in T'^{\downarrow}_b$ and $a_1, \ldots, a_l \notin T'^{\downarrow}_b$. Let $\pi_0 = \mathrm{type}^{\mathfrak{T}'}(c, b_1, \ldots, b_k, a_1, \ldots, a_l)$. Note that π_0 is a rearrangement of π. Let $\pi'_0 = \mathrm{type}^{\mathfrak{T}'}(b, a_1, \ldots, a_l)$. Note that $\pi'_0 = \mathrm{type}^{\mathfrak{T}}(a, a_1, \ldots, a_l)$ and thus $\pi'_0 \in \mathrm{prof}_n^{\mathfrak{T}}(a).\mathcal{A} = \mathrm{prof}_n^{\mathfrak{T}}(b).\mathcal{A}$. It follows that π'_0 is realized in $\mathfrak{T}$ by $b, a'_1, \ldots, a'_l$ for some $a'_1, \ldots, a'_l \in T^{\uparrow}_b$. Observe that $\mathrm{type}^{\mathfrak{T}}(c, b_1, \ldots, b_k, a'_1, \ldots, a'_l) = \pi_0$ and thus $\pi_0 \in \mathrm{prof}_n^{\mathfrak{T}}(c).\mathcal{A}$. As $\mathrm{prof}_n^{\mathfrak{T}}(c).\mathcal{A}$ is closed under initial subtypes (and thus also rearrangements) it follows that $\pi \in \mathrm{prof}_n^{\mathfrak{T}}(c).\mathcal{A}$

For the opposite direction, consider now any $\pi \in \mathrm{prof}_n^{\mathfrak{T}}(c).\mathcal{A}$. Take a realization of π in $\mathfrak{T}$ starting with c. Let $c, b_1, \ldots, b_k, a_1, \ldots, a_l$ be a list of all nodes of this realization such that $b_1, \ldots, b_k \in T^{\downarrow}_b$ and $a_1, \ldots, a_l \notin T^{\downarrow}_b$. Let $\pi_0 = \mathrm{type}^{\mathfrak{T}}(c, b_1, \ldots, b_k, a_1, \ldots, a_l)$ Let $\pi'_0 = \mathrm{type}^{\mathfrak{T}}(b, a_1, \ldots, a_l)$. Note that $\pi'_0 \in \mathrm{prof}_n^{\mathfrak{T}}(b).\mathcal{A} = \mathrm{prof}_n^{\mathfrak{T}}(a).\mathcal{A}$ and thus $a, a'_1, \ldots, a'_l$ realize π'_0 in $\mathfrak{T}'$ for some $a'_1, \ldots, a'_l$. Now $\mathrm{type}^{\mathfrak{T}'}(b, a'_1, \ldots, a'_l) = \pi'_0$ and $\mathrm{type}^{\mathfrak{T}'}(c, b_1, \ldots, b_k, a'_1, \ldots, a'_l) = \pi_0$. Thus $\pi_0 \in \mathrm{prof}_n^{\mathfrak{T}}(c).\mathcal{A}$ and since π is a rearrangement of π_0 then also $\pi \in \mathrm{prof}_n^{\mathfrak{T}}(c).\mathcal{A}$.

The case when $c \in T'^{\uparrow}_b$ can be analysed analogously. Since we have shown that all profiles of nodes in $\mathfrak{T}'$ are realized in $\mathfrak{T}$ it follows by Lemma 3 that $\mathfrak{T}' \models \varphi$. ◀

Having proved Lemma 5 we can now obtain the desired bounds.

▶ **Corollary 6.**

(i) *Every satisfiable $\mathrm{F}_1[\downarrow]$, $\mathrm{F}_1[\downarrow,\downarrow_+]$ or $\mathrm{F}_1[\downarrow,\downarrow_+,\rightarrow,\rightarrow^+]$ normal form formula φ has a model whose vertical root-to-leaf paths are bounded doubly exponentially in $\|\varphi\|$ by a fixed function $\mathfrak{f}_d$.*

(ii) *Every satisfiable $\mathrm{F}_1[\downarrow_+]$ normal form formula φ has a model whose vertical root-to-leaf paths are bounded exponentially in $\|\varphi\|$ by a fixed function $\mathfrak{f}_s$.*

Proof. Take a model $\mathfrak{T} \models \varphi$. If there are nodes a, b meeting conditions of Lemma 5 then replace the subtree rooted at a by the subtree rooted at b. Repeat this operation until all root-to-leaf paths realize only distinct n-profiles. Let $\mathfrak{T}^*$ be the eventually obtained tree. By Lemma 5 we have $\mathfrak{T}^* \models \varphi$. To see (i) just recall that by Claim 2 (ii) the number of distinct

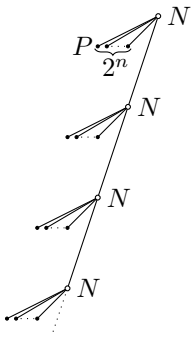


Figure 1 Enforcing doubly exponential path in $\mathrm{F}_1[\downarrow]$.

n-profiles over $\{\downarrow\}$, $\{\downarrow, \downarrow_+\}$ or $\{\downarrow, \downarrow_+, \rightarrow, \rightarrow^+\}$ is bounded doubly exponentially in $\tau_0(\varphi)$ and n and thus also in $\|\varphi\|$. For (ii) take any vertical root-to-leaf path p of $\mathfrak{T}^*$ and consider any 1-type α realized on this path. Let $a_1, \ldots, a_k$ be the list of all nodes of p realizing α, $\mathfrak{T}^* \models a_i \downarrow_+ a_j$ for $i < j$. Let $(\alpha, \mathcal{A}_i, \mathcal{B}_i)$ be the n-profile (over $\{\downarrow_+\}$) of a_i for $1 \leq i \leq k$. We observe that for $i < j$ we have $\mathcal{A}_i \subseteq \mathcal{A}_j$ and, $\mathcal{B}_i \supseteq \mathcal{B}_j$. Let us explain the former of these two inclusions. Take any k-type $\pi \in \mathcal{A}_i$ and let $a_i, b_2, \ldots, b_k$ be its realization. Note that the nodes $b_2, \ldots, b_k$ are related by $\downarrow_+$ to a_j precisely as to a_i. Thus $a_j, b_2, \ldots, b_k$ realizes π and thus $\pi \in \mathcal{A}_j$. The latter inclusion can be shown analogously. Recall that by Claim 2 (i) $|\mathcal{A}_i|$ and $|\mathcal{B}_i|$ are bounded exponentially. Thus when moving along $a_1, \ldots, a_k$ each of the components $\mathcal{A}_i$ and $\mathcal{B}_i$ can change at most exponentially many times, and in consequence, k is bounded exponentially. Finally, noting that the number of 1-types is also bounded exponentially we get the desired bound. ◀

The bounds in the both parts of the above corollary are essentially optimal. Exponentially long paths over $\{\downarrow_+\}$ can be easily enforced even in FO^2 by organizing, by means of unary predicates $P_0, \ldots, P_{n-1}$, a binary counter counting from 0 to $2^n - 1$ and requiring each node storing a value smaller than $2^n - 1$ to have a descendant storing the value greater by one. Let us see that the presence of $\downarrow$ allows to simply enforce doubly-exponential paths. We use unary predicates $N, P, P_0, \ldots, P_{n-1}, Q$. See Fig. 1. The intended long path is the path of elements in N. Every element in N is going to have 2^n children marked by P, each of which has a *local position* in the range $[0, 2^n - 1]$ encoded by means of $P_0, \ldots, P_{n-1}$. Reading the truth-values of Q as binary digits we can assume that the collection of the P-children of a node in N encodes its *global position* in the tree in the range $[0, 2^{2^n} - 1]$ (the i-th bit of this global position is 1 iff at the element at local position i the value of Q is true). It is then possible to say that each node in n whose global position is smaller than $2^{2^n} - 1$ has a child in N with the global position greater by 1. We skip here the details.

3.2 Bounded degree

Our next aim is to show that also the degree of nodes can be bounded.

▶ **Lemma 7.**

(i) *Let φ be a normal form $\mathrm{F}_1[\downarrow, \downarrow_+, \rightarrow, \rightarrow^+]$ formula. Let $\mathfrak{T} \models \varphi$. Then there exists a tree $\mathfrak{T}^* \models \varphi$ obtained by removing some subtrees from $\mathfrak{T}$ (and appropriately repairing the sibling relations), in which the degree of every node is bounded doubly exponentially in $\|\varphi\|$ by a fixed function $\mathfrak{f}'_d$.*

(ii) *Let φ be a normal form $\mathrm{F}_1[\downarrow]$, $\mathrm{F}_1[\downarrow_+]$ or $\mathrm{F}_1[\downarrow, \downarrow_+]$. Let $\mathfrak{T} \models \varphi$. Then there exists a tree $\mathfrak{T}^* \models \varphi$ obtained by removing some subtrees from $\mathfrak{T}$, in which the degree of every node is bounded exponentially in $\|\varphi\|$ by a fixed function $\mathfrak{f}'_s$.*

Proof. Let n be the width of φ.

(i) Consider any node $a \in T$. Let $a_1, \ldots, a_k$ be all the the children of a, listed from left to right. If for some $i < j$ the horizontal n-profiles of a_i and a_j are equal (note that $i > 1$ in this case) then we remove all the subtrees rooted at $a_i, \ldots, a_{j-1}$ and join a_{i-1} with a_j by $\rightarrow$. By arguments similar to those from the proof of Lemma 5 we can show that the profiles of the surviving elements of T do not change. Repeating this process as long as possible we eventually obtain a tree in which the number of children of a is bounded doubly exponentially (by the number of distinct horizontal n-profiles). We then repeat the process successively for all the nodes of $\mathfrak{T}$.

(ii) The $\mathcal{B}$ components of profiles do not behave monotonically along horizontal paths (as they do along vertical paths), thus we cannot use horizontal k-profiles to get the desired exponential bound on their length. We proceed in a slightly different manner. Consider any node $a \in T$. Let $\bar{a} = a_1, \ldots, a_k$ be the list of the children of a. For $1 \leq i \leq k$ let $types(a_i)$ be the set of types of size at most n realized in $\mathfrak{T}^{\downarrow}_{a_i}$ by tuples whose first element is a_i. For each type $\pi \in \bigcup_{i=1}^{k} types(a_i)$ mark n nodes in $\bar{a}$ such that $\pi \in types(a_i)$ (or all such nodes if there are less than n of them). This way we mark at most exponentially many children of a. Let us remove all the subtrees rooted at unmarked nodes from $\bar{a}$ and denote the obtained tree $\mathfrak{T}'$. We claim the the n-profiles of all the elements surviving the surgery do not change. Consider any $c \in T'$. Noting that the elements of $\mathfrak{T}'$ are related to each other by the navigational predicates $\downarrow_+, \downarrow$ exactly as they are related in $\mathfrak{T}$ we see that $\mathrm{prof}^{\mathfrak{T}'}_n(c).\mathcal{A} \subseteq \mathrm{prof}^{\mathfrak{T}}_n(c).\mathcal{A}$ and $\mathrm{prof}^{\mathfrak{T}'}_n(c).\mathcal{B} \subseteq \mathrm{prof}^{\mathfrak{T}}_n(c).\mathcal{B}$.

For $\supseteq$ we distinguish two case: the one in which c is in $\mathfrak{T}'^{\uparrow}_a$, and the other in which it is not. Let us sketch the arguments for the latter (the former is similar). Since c retains its subtree from $\mathfrak{T}$ it is clear that $\mathrm{prof}^{\mathfrak{T}}_n(c).\mathcal{B} \subseteq \mathrm{prof}^{\mathfrak{T}'}_n(c).\mathcal{B}$. To see that $\mathrm{prof}^{\mathfrak{T}}_n(c).\mathcal{A} \subseteq \mathrm{prof}^{\mathfrak{T}'}_n(c).\mathcal{A}$ take any $\pi \in \mathrm{prof}^{\mathfrak{T}}_n(c).\mathcal{A}$ and its any realization $c, b_1, \ldots, b_l \in T$. Split $b_1, \ldots, b_l$ into the disjoint tuples of nodes: let the first tuple $\bar{b}_0$ contain the nodes from $\mathfrak{T}^{\uparrow}_a$ and the other tuples $\bar{b}_1, \ldots, \bar{b}_s$ the nodes from the subtrees rooted at distinct nodes from $\bar{a}$ (note that $s < n$). The tuple $\bar{b}_0$ is retained in $\mathfrak{T}'$; the other tuples may be deleted, but due to our strategy of marking important nodes in $\bar{a}$ there exists a $1{-}1$ function returning for a tuple $\bar{b}_i$ a node in $\bar{a}$ surviving the surgery in whose subtree a tuple $\bar{b}'_i$ of type equal to the type of $\bar{b}_i$ exists. Using the elements $c, \bar{b}_0, \bar{b}'_1, \ldots, \bar{b}'_s$ we can now form a realization of π in $\mathfrak{T}'$ (starting from c). Thus $\pi \in \mathrm{prof}^{\mathfrak{T}'}_n(c).\mathcal{A}$, which finishes the argument.

Again, repeating the described process for all nodes we eventually obtain a tree $\mathfrak{T}^*$ in which the degree of every node is bounded exponentially and the n-profiles of all nodes remain as in $\mathfrak{T}$, and thus are φ-admissible. In effect $\mathfrak{T}^* \models \varphi$. ◀

The bounds on the degree of nodes in Lemma 7 are essentially optimal. In particular a doubly exponential chain of siblings can be enforced by means of $\rightarrow$ and $\downarrow$ (or $\rightarrow$ and $\downarrow_+$) similarly to a doubly exponential vertical root-to-leaf path: every element of the chain is required to have 2^n children storing the binary digits of a doubly exponential counter; the next sibling of a node a is forced to store the counter value greater by one than the value stored at a.

4 Complexity of satisfiability

In this section, using the results from Section 3 we establish the precise complexity of the satisfiability problem in all of the scenarios we consider.

4.1 Only descendant relation

Let us start with the observation that in the case when only the descendant relation is present in the navigational signature the complexity of F_1 is equal to the complexity of FO^2 and C^2.

▶ **Theorem 8.** *The satisfiability problem for* $\mathrm{F}_1[\downarrow_+]$ *is* ExpSpace*-complete.*

The lower bound is inherited from $\mathrm{FO}^2[\downarrow_+]$, [2], which in turn refers to ExpSpace-hardness of the so-called one-way two-variable guarded fragment, [9]. For the upper bound we design an alternating exponential time procedure. The result then follows from the well known fact that AExpTime=ExpSpace, [3]. The procedure first guesses the profile of the root and then guesses the profiles of its children, checking if the information recorded in the profiles is locally consistent, and if each guessed profile is φ-admissible. Further, it works in a loop, universally moving to one of the children, guessing profiles of its children and proceeding similarly.

Algorithm 1: Procedure F_1`[↓+]-sat-test`

Input: an $\mathrm{F}_1[\downarrow_+]$ normal form formula φ
- let n be the width of φ
- let $maxdepth := \mathfrak{f}_s(\|\varphi\|)$; let $maxdegree := \mathfrak{f}'_s(\|\varphi\|)$; % *cf. Cor. 6 and Lem. 7*
- let $level := 0$;
- **guess** an n-profile θ such that $\theta.\mathcal{A} = \{\alpha\}$; % *root*
- **while** $level < maxdepth$ **do**
  - if θ is not φ-admissible then **reject**
  - **guess** an integer $0 \leq k \leq maxdegree$; % *the number of children*
  - for $1 \leq i \leq k$ **guess** a profile θ_i;
  - if not *locally-consistent*$(\theta, \theta_1, \ldots, \theta_k)$ then **reject**;
  - if $\theta.\mathcal{B} = \{\alpha\}$ then **accept** % *a leaf reached; it must be* $k = 0$
  - $level := level + 1$;
  - **universally choose** $1 \leq i \leq k$; let $\theta := \theta_i$;
- **endwhile**
- **reject**
- **endprocedure**

The function *locally-consistent* checks whether, from a local point of view, a tree may have a node realizing an n-profile θ whose children realize n-profiles $\theta_1, \ldots, \theta_k$. It checks if $\theta.\mathcal{B}$ is the set of types which can be obtained, informally speaking, by putting $\theta.\alpha$ on top of a disjoint union of types taken from distinct $\theta_i.\mathcal{B}$ (possibly with their roots removed); and, analogously, verifies some natural conditions on the $\mathcal{A}$-components of θ and θ_i-s. We skip the details of this function.

▶ **Lemma 9.** *Procedure* F_1`[↓+]-sat-test` *accepts its input* φ *iff* φ *is satisfiable.*

Proof. Assume that φ is satisfiable. By Corollary 6 (ii) and Lemma 7 (ii) there exists a small model $\mathfrak{T} \models \varphi$. The procedure accepts φ by making all its guesses in accordance to $\mathfrak{T}$, i.e., in the first step it sets θ to be equal to the n-profile of the root of $\mathfrak{T}$ and then in each step it sets θ_i to be the n-profile of the i-th child of the previously considered element.

In the opposite direction, from an accepting (tree-)run t of the procedure we can naturally construct a tree structure $\mathfrak{T}_t$, with 1-types of elements as guessed during the execution. Our procedure guesses actually not only 1-types but full n-profiles of nodes. The function *locally-consistent* guarantees that the n-profiles of nodes $\mathfrak{T}_t$ are indeed as guessed. To see this, we first prove by induction on the height of nodes that the $\mathcal{B}$ components of profiles are

as required: the base step for leaves holds due to the acceptance condition of the procedure; then we move towards the root using the conditions of the function *locally-consistent*. For the $\mathcal{A}$-components we proceed by induction on the depth of nodes: the base case holds for the root by the requirement on the first profile from the procedure; then we move down the tree using the conditions of the function *locally-consistent* and the already proved fact that the $\mathcal{B}$-component are as required.

Since the procedure checks if each of the guessed n-profiles is φ-admissible, then by Lemma 3 we have that $\mathfrak{T}_t \models \varphi$. ◀

To finish the proof of Thm. 8 it remains to note that the values of *maxdepth* and *maxdegree* ensure that the algorithm works indeed in (alternating) exponential time.

4.2 Descendant and child

We first note that when the child relation $\downarrow$ is available in the signature then, in contrast to the case of FO^2 and C^2 the satisfiability problem becomes 2-ExpTime-hard. This can be shown by a simple adaptation of the 2-ExpTime-hardness proof for the unary negation fragment, UNFO, [17], which works in particular for UNFO[$\downarrow$] over trees. Actually some two-dimensional formulas appear in that proof, but their usage is not crucial and can be easily avoided.

▶ **Theorem 10.** *The satisfiability problem for* $\mathrm{F}_1[\downarrow]$ *is 2-*ExpTime*-hard.*

A matching upper bound for $\mathrm{F}_1[\downarrow, \downarrow_+]$ is easy to obtain using the tools we have already developed.

▶ **Theorem 11.** *The satisfiability problem for* $\mathrm{F}_1[\downarrow, \downarrow_+]$ *is 2-*ExpTime*-complete.*

Proof. The lower bound follows from Thm. 10. For the upper bound we just modify the procedure F_1`[`$\downarrow_+$`]-sat-test` from Section 4.1 in a natural way. By Corollary 6 (i) and Lemma 7 (i) we know that satisfiable formulas have models with doubly exponentially bounded paths and exponentially bounded degree of nodes. Thus we just change the initial value of *maxdepth* to $\mathfrak{f}_d(\|\varphi\|)$ and adjust the function *locally-consistent* by taking into account the presence of $\downarrow$ relation. The obtained procedure works in alternating exponential space. Since AExpSpace = 2-ExpTime [3] we get the desired result. ◀

4.3 Ordered trees

We conclude this section observing that the satisfiability problem of F_1 over ordered trees remains in 2-ExpTime.

▶ **Theorem 12.** *The satisfiability problem for* $\mathrm{F}_1[\downarrow, \downarrow_+, \rightarrow, \rightarrow^+]$ *is 2-*ExpTime*-complete.*

Proof. The lower bound follows from Thm. 10. For the upper bound we need another modification of the procedure F_1`[`$\downarrow_+$`]-sat-test`. As we want to fit in alternating exponential space we cannot this time allow ourselves for guessing at once all the children of a node (as there may be doubly exponentially many of them). Instead we start by guessing the leftmost one, and then move to the right until we reach the rightmost one. During this horizontal walk, at each node a we actually make a universal choice between continuing the walk to the right and moving down to the first child of a. Further, instead of n-profiles of nodes we guess horizontal-n-profiles of the form $(\alpha, \mathcal{A}, \mathcal{B}, \mathcal{A}_L, \mathcal{A}_R)$. The function `locally-consistent` takes this into account; it can be naturally designed to ensure that the the nodes guessed in an

accepting tree-run of the procedure indeed have the declared horizontal profiles. Note in particular that knowing the components $\mathcal{A}_L$, $\mathcal{A}_R$ and $\mathcal{B}$ of the horizontal profile of any child of a one can compute the $\mathcal{B}$ component of the profile of a. We leave the technical details of the required adaptation for the full version of the paper. ◀

5 Expressivity

We show that the expressive power of $\mathrm{F}_1[\tau_{bin}]$ is equal to the expressive power of $\mathrm{C}^2[\tau_{bin}]$ in the case of ordered trees, $\tau_{bin} = \{\downarrow, \downarrow_+, \rightarrow, \rightarrow^+\}$, and in the case of unordered trees accessible by both the descendant and the child relations, $\tau_{bin} = \{\downarrow, \downarrow_+\}$. Translation from C^2 to F_1 is easy. Consider, e.g., a subformula of the form $\exists^{\geq k} y \psi(x, y)$ and note that it can be written as $\exists y_1, \ldots, y_k(\bigwedge_{i \neq j} y_i \neq y_j \wedge \bigwedge_i \psi(x, y_i))$. Regarding the opposite direction, in the case of the ordered trees the equivalence is not very surprising: the power of the full navigational signature allows scanning trees in an ordered way and express the existence of patterns described by F_1 subformulas by a reuse of two variables. Actually, one can even directly translate F_1 to FO^2 without counting in this case. The equivalence over unordered trees is slightly harder and less obvious. The proof of the following theorem will be given in the full version of this paper.

▶ **Theorem 13.** *For any* $\mathrm{F}_1[\downarrow, \downarrow_+]$ *(*$\mathrm{F}_1[\downarrow, \downarrow_+, \rightarrow, \rightarrow^+]$*) sentence* φ *there is a* $\mathrm{C}^2[\downarrow, \downarrow_+]$ *(*$\mathrm{C}^2[\downarrow, \downarrow_+, \rightarrow, \rightarrow^+]$*) sentence* φ^* *such that for any tree* $\mathfrak{T}$ *we have* $\mathfrak{T} \models \varphi$ *iff* $\mathfrak{T} \models \varphi^*$*, and vice versa.*

Let us also collect here the results comparing the expressive power over trees of F_1, C^2 and two other two-variable logics: the two-variable guarded fragment, GF^2, and plain FO^2. By $A \prec B$ we denote that A is strictly less expressive than B (on the level of sentences), and by $A \equiv B$ we denote that A and B have the same expressive power.

▶ **Corollary 14.**
(i) *Over* $\tau = \{\downarrow, \downarrow_+, \rightarrow, \rightarrow^+\}$ *we have* $\mathrm{GF}^2[\tau] \equiv \mathrm{FO}^2[\tau] \equiv \mathrm{C}^2[\tau] \equiv \mathrm{F}_1[\tau]$.
(ii) *If* $\tau = \{\downarrow, \downarrow_+\}$ *then* $\mathrm{GF}^2[\tau] \prec \mathrm{FO}^2[\tau] \prec \mathrm{C}^2[\tau] \equiv \mathrm{F}_1[\tau]$.

Regarding (i): the translation of FO^2 to GF^2 can be done similarly to the translation of FO^2 to a variant of Core XPath in [15], equivalence of C^2 and FO^2 is shown in [1] and of C^2 and F_1 in Thm. 13. Regarding (ii): Let us assume that the signature contains no unary predicates and for $i \in \mathbb{N}$ let $\mathfrak{T}_i$ denote the tree consisting just of a root and its i children. The C^2 formula $\exists x \exists^{\geq 3} y\ x \downarrow_+ y$ distinguishes $\mathfrak{T}_3$ and $\mathfrak{T}_2$, while the FO^2 formula $\exists xy(\neg x \downarrow_+ y \wedge \neg y \downarrow_+ x \wedge x \neq y)$ distinguishes $\mathfrak{T}_2$ and $\mathfrak{T}_1$. It is not difficult to see that FO^2 cannot distinguish between $\mathfrak{T}_3$ and $\mathfrak{T}_2$ (use a simple 2-pebble game argument, cf. [1]) and that GF^2 cannot distinguish between $\mathfrak{T}_2$ and $\mathfrak{T}_1$ (use bisimulations).

6 Conclusion

We established the computational complexity of F_1 over unordered trees, showing its ExpSpace-completeness over trees accessible only by the descendant relation and 2-ExpTime-completeness in the presence of the child relation. The 2-ExpTime-upper bound holds also for ordered trees equipped additionally with the next-sibling and following-sibling relations. Deriving the complexities for the remaining combinations of the considered navigational symbols (e.g., $\mathrm{F}_1[\downarrow_+, \rightarrow]$) is not difficult, and we will do it in the full version of this paper.

We also proved that under two of the considered navigational scenarios F_1 is expressively equivalent to C^2. Extending the expressive power comparison between F_1 and C^2 (and other logics) to the signatures containing just one of $\downarrow$, $\downarrow_+$ is left as a future work.

Another interesting open problem is to formally compare the succinctness of F_1 versus FO^2 and C^2 over trees (and over words).

We worked with unranked trees, but it is not difficult to adapt all the results for the case of ranked trees. In particular the counters in the 2-ExpTime-lower bound proof (Thm. 10) can be organized as binary subtrees instead of horizontal chains of siblings.

References

1 Bartosz Bednarczyk, Witold Charatonik, and Emanuel Kieronski. Extending two-variable logic on trees. In *Computer Science Logic*, pages 11:1–11:20, 2017.

2 Saguy Benaim, Michael Benedikt, Witold Charatonik, Emanuel Kieronski, Rastislav Lenhardt, Filip Mazowiecki, and James Worrell. Complexity of two-variable logic on finite trees. *ACM Trans. Comput. Log.*, 17(4):32:1–32:38, 2016.

3 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

4 Witold Charatonik, Emanuel Kieronski, and Filip Mazowiecki. Satisfiability of the two-variable fragment of first-order logic over trees. *CoRR*, abs/1304.7204, 2013.

5 Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and a linear order. In *Computer Science Logic*, pages 631–647, 2015.

6 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

7 Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002. doi:10.1006/inco.2001.2953.

8 Lauri Hella and Antti Kuusisto. One-dimensional fragment of first-order logic. In *Advances in Modal Logic 10*, pages 274–293, 2014.

9 Emanuel Kieronski. On the complexity of the two-variable guarded fragment with transitive guards. *Inf. Comput.*, 204(11):1663–1703, 2006.

10 Emanuel Kieronski. One-dimensional logic over words. In *Computer Science Logic*, pages 38:1–38:15, 2016.

11 Emanuel Kieronski and Antti Kuusisto. Complexity and expressivity of uniform one-dimensional fragment with equality. In *Mathematical Foundations of Computer Science, Part I*, pages 365–376, 2014.

12 Emanuel Kieronski and Antti Kuusisto. Uniform one-dimensional fragments with one equivalence relation. In *Computer Science Logic*, pages 597–615, 2015.

13 Andreas Krebs, Kamal Lodaya, Paritosh Pandya, and Howard Straubing. Two-variable logic with a between predicate. In *Logic in Computer Science*, 2016.

14 Antti Kuusisto. On the uniform one-dimensional fragment. In *Proceedings of Description Logic Workshop*, 2016.

15 Maarten Marx. First order paths in ordered trees. In *International Conference on Database Theory*, pages 114–128, 2005.

16 Dana Scott. A decision method for validity of sentences in two variables. *Journal Symbolic Logic*, 27:477, 1962.

17 Luc Segoufin and Balder ten Cate. Unary negation. *Logical Methods in Computer Science*, 9(3), 2013.

18 Larry J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, MIT, Cambridge, Massachusetts, USA, 1974.

An Improved FPT Algorithm for the Flip Distance Problem*

Shaohua Li[1], Qilong Feng[2], Xiangzhong Meng[3], and Jianxin Wang[4]

1 **School of Information Science and Engineering, Central South University, Changsha, Hunan, P. R. China**
2 **School of Information Science and Engineering, Central South University, Changsha, Hunan, P. R. China**
3 **School of Information Science and Engineering, Central South University, Changsha, Hunan, P. R. China**
4 **School of Information Science and Engineering, Central South University, Changsha, Hunan, P. R. China**
jxwang@mail.csu.edu.cn

Abstract

Given a set $\mathcal{P}$ of points in the Euclidean plane and two triangulations of $\mathcal{P}$, the flip distance between these two triangulations is the minimum number of flips required to transform one triangulation into the other. The Parameterized Flip Distance problem is to decide if the flip distance between two given triangulations is equal to a given integer k. The previous best FPT algorithm runs in time $O^*(k \cdot c^k)$ ($c \leq 2 \times 14^{11}$), where each step has fourteen possible choices, and the length of the action sequence is bounded by $11k$. By applying the backtracking strategy and analyzing the underlying property of the flip sequence, each step of our algorithm has only five possible choices. Based on an auxiliary graph G, we prove that the length of the action sequence for our algorithm is bounded by $2|G|$. As a result, we present an FPT algorithm running in time $O^*(k \cdot 32^k)$.

1998 ACM Subject Classification F.2.2 Geometrical Problems and Computations, G.2.1 Combinatorial Algorithms

Keywords and phrases triangulation, flip distance, FPT algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.65

1 Introduction

Given a set $\mathcal{P}$ of n points in the Euclidean plane, a *triangulation* of $\mathcal{P}$ is a maximal planar subdivision whose vertex set is $\mathcal{P}$ [10]. A *flip* operation to one diagonal e of a convex quadrilateral in a triangulation is to remove e and insert the other diagonal into this quadrilateral. Note that if the quadrilateral associated with e is not convex, the flip operation is not allowed. The *flip distance* between two triangulations is the minimum number of flips required to transform one triangulation into the other.

Triangulations play an important role in computational geometry, which are applied in areas such as computer-aided geometric design and numerical analysis [11, 13, 21].

Given a point set $\mathcal{P}$ in the Euclidean plane, we can construct a graph $G_T(\mathcal{P})$ in which every triangulation of $\mathcal{P}$ is represented by a vertex, and two vertices are adjacent if their

* This work is supported in part by the National Natural Science Foundation of China under Grants (61420106009, 61232001, 61472449, 61672536, 61572414).

corresponding triangulations can be transformed into each other through one flip operation. $G_T(\mathcal{P})$ is called the *triangulations graph* of $\mathcal{P}$. Properties of the triangulations graph are studied in the literature. Aichholzer et al. [1] showed that the lower bound of the number of vertices of $G_T(\mathcal{P})$ is $\Omega(2.33^n)$. Lawson and Charles [17] showed that the diameter of $G_T(\mathcal{P})$ is $O(n^2)$. Hurtado et al. [14] proved that the bound is tight. Since $G_T(\mathcal{P})$ is connected [17], any two triangulations of $\mathcal{P}$ can be transformed into each other through a certain number of flips.

The Flip Distance problem consists in computing the flip distance between two triangulations of $\mathcal{P}$, which was proved to be NP-complete by Lubiw and Pathak [18]. Pilz [20] showed that the Flip Distance problem is APX-hard. Aichholzer et al. [2] proved that the Flip Distance problem is NP-complete on triangulations of simple polygons. However, the complexity of the Flip Distance problem on triangulations of convex polygons has been open for many years, which is equivalent to the problem of computing the rotation distance between two rooted binary trees [23].

The Parameterized Flip Distance problem is: given two triangulations of a set of points in the plane and an integer k, deciding if the flip distance between these two triangulations is equal to k. For the Parameterized Flip Distance problem on triangulations of a convex polygon, Lucas [19] gave a kernel of size $2k$ and an $O^*(k^k)$-time algorithm. Kanj and Xia [15] studied the Parameterized Flip Distance problem on triangulations of a set of points in the plane, and presented an $O^*(k \cdot c^k)$-time algorithm ($c \leq 2 \cdot 14^{11}$), which applies to triangulations of general polygonal regions (even with holes or points inside it).

In this paper, we exploit the structure of the Parameterized Flip Distance problem further. At first, we give a nondeterministic construction process to illustrate our idea. The nondeterministic construction process contains only two types of actions, which are the *moving action* as well as the *flipping and backing action*. There are 4 choices for the moving action and one choice for the flipping and backing action. Given two triangulations and a parameter k, we prove that either there exists a sequence of actions of length at most $2k$, following which we can transform one triangulation into the other, or we can reject this instance. Thus we get an improved $O^*(k \cdot 32^k)$-time FPT algorithm, which also applies to triangulations of general polygonal regions (even with holes or points inside it).

2 Preliminaries

In a triangulation T, a flip operation f to an edge e that is the diagonal of a convex quadrilateral $\mathcal{Q}$ is to delete e and insert the other diagonal e' into $\mathcal{Q}$. We define e as the *underlying edge* of f, denoted by $\varepsilon(f)$, and e' as the *resulting edge* of f, denoted by $\varphi(f)$. (For consistency and clarity, we continue to use some symbols and definitions from [15]). Note that if e is not a diagonal of any convex quadrilateral in the triangulation, flipping e is not allowed. Suppose that we perform a flip operation f on a triangulation T_1 and get a new triangulation T_2. We say f transforms T_1 into T_2. T_1 is called an *underlying triangulation* of f, and T_2 is called a *resulting triangulation* of f. Given a set $\mathcal{P}$ of n points in the Euclidean plane, let T_{start} and T_{end} be two triangulations of $\mathcal{P}$, in which T_{start} is the initial triangulation and T_{end} is the objective triangulation. Let $F = \langle f_1, f_2, ..., f_r \rangle$ be a sequence of flips, and $\langle T_0, T_1, ..., T_r \rangle$ be a sequence of triangulations of $\mathcal{P}$ in which $T_0 = T_{start}$ and $T_r = T_{end}$. If T_{i-1} is an underlying triangulation of f_i, and T_i is a resulting triangulation of f_i for each $i = 1, 2, ..., r$, we say F transforms T_{start} into T_{end}, or F is a *valid sequence*, denoted by $T_{start} \xrightarrow{F} T_{end}$. The flip distance between T_{start} and T_{end} is the length of a shortest valid flip sequence.

Now we give the formal definition of the Parameterized Flip Distance problem.

Parameterized Flip Distance
Input: Two triangulations T_{start} and T_{end} of $\mathcal{P}$ and an integer k.
Question: Decide if the flip distance between T_{start} and T_{end} is equal to k.

The triangulation on which we are performing a flip operation is called the *current triangulation*. An edge e which belongs to the current triangulation but does not belong to T_{end} is called a *necessary edge* in the current triangulation. It is easy to see that for any necessary edge e, there must exist a flip operation f in a valid sequence such that $e = \varepsilon(f)$. Otherwise, we cannot get the objective triangulation T_{end}.

For a directed graph D, a maximal connected component of its underlying graph is called a *weakly connected component* of D. We define the size of an undirected tree as the number of its vertices.

A *parameterized problem* is a decision problem for which every instance is of the form (x, k), where x is the input instance and $k \in \mathbb{N}$ is the parameter. A parameterized problem is *fixed-parameter tractable* (FPT) if it can be solved by an algorithm (*FPT algorithm*) in $O(f(k)|x|^{O(1)})$ time, where $f(k)$ is a computable function of k. In addition to computational geometry, parameterized problems in other areas such as graph theory [8, 9, 12], computational biology [3, 22] and MAX-SAT [6] are also studied extensively. For a further introduction to parameterized algorithms, readers could refer to [4, 7].

3 The Improved Algorithm for the Parameterized Flip Distance Problem

Given T_{start} and T_{end}, let $F = \langle f_1, f_2, ..., f_r \rangle$ be a valid sequence, that is, $T_{start} \xrightarrow{F} T_{end}$. Definition 1 defines the adjacency of two flips in F.

▶ **Definition 1.** [15] Let f_i and f_j be two flips in F $(1 \leq i < j \leq r)$. We define that flip f_j is adjacent to flip f_i, denoted by $f_i \to f_j$, if the following two conditions are satisfied:
(1) either $\varphi(f_i) = \varepsilon(f_j)$, or $\varphi(f_i)$ and $\varepsilon(f_j)$ share a triangle in triangulation T_{j-1};
(2) $\varphi(f_i)$ is not flipped between f_i and f_j, that is, there does not exist a flip f_p in F, where $i < p < j$, such that $\varphi(f_i) = \varepsilon(f_p)$.

By Definition 1, we can construct a directed acyclic graph (DAG), denoted by D_F. Every node in D_F represents a flip operation of F, and there is an arc from f_i to f_j if f_j is adjacent to f_i. For convenience, we label the nodes in D_F using labels of the corresponding flip operations. In other words, we can see a node in D_F as a flip operation and vice versa.

The following lemma shows that any topological sorting of D_F is a valid sequence.

▶ **Lemma 2.** *[15] Let T_0 and T_r be two triangulations and $F = \langle f_1, f_2, ..., f_r \rangle$ be a sequence of flips such that $T_0 \xrightarrow{F} T_r$. Let $\pi(F)$ be a permutation of the flips in F such that $\pi(F)$ is a topological sorting of D_F. Then $\pi(F)$ is a valid sequence of flips such that $T_0 \xrightarrow{\pi(F)} T_r$.*

Lemma 2 ensures that if we repeatedly remove a source node from D_F and flip the underlying edge of this node until D_F becomes empty, we can get a valid sequence and the objective triangulation T_{end}.

Here we introduce the definition of a walk.

▶ **Definition 3.** [16] A *walk* in a triangulation T (starting from an edge $e \in T$) is a sequence of edges of T beginning with e in which any two consecutive edges share a triangle in T.

According to Lemma 2, if there is a valid sequence F for the input instance, any topological sorting of D_F is also a valid sequence for the given instance. The difficulty is that F is unknown. In order to find the topological sorting of D_F, the algorithm in [15] takes a nondeterministic walk to find an edge e which is the underlying edge of a source node, flips this edge (removing the corresponding node from D_F), nondeterministically walks to an edge which shares a triangle with e and recursively searches for an edge corresponding to a source node. They deal with weakly connected components of D_F one after one (refer to Corollary 4 in [15]), that is, their algorithm tries to find a solution F in which all flips belonging to the same weakly connected component of D_F appear consecutively. In order to keep searching procedure within the current weakly connected component, they use a stack to preserve the nodes (defined as *connecting point* in [15]) whose removal separates the current weakly connected component into small weakly connected components. When removing all nodes of a small component, their algorithm jumps to the connecting point at the top of the stack and moves to deal with another small component.

We observe that it is not necessary to remove all nodes of a weakly connected component before dealing with other weakly connected components, that is, our algorithm may find a solution F in which the nodes belonging to the same weakly connected components appear dispersedly. Thus we do not need a stack to preserve connecting points. Instead of five types of actions in [15], we only need two types, that is, moving action as well as flipping and backtracking action (see Section 3.2). Moreover, every time we find a source node, we remove the node, flip the underlying edge and backtrack instead of searching for the next node in four directions, thus reducing the number of choices for the actions. Another contribution of this paper is that we construct an auxiliary graph G and prove that G is a forest. Since there is a bijection between nondeterministic actions and nodes as well as edges of G, we prove that there exists a sequence of actions of length at most $2|D_F|$, which is smaller than $11|D_F|$ in [15]. In addition, we make some optimization on the strategy of finding the objective sequence. As a result, we improve the running time of the algorithm from $O^*(k \cdot c^k)$ where $c \leq 2 \cdot 14^{11}$ to $O^*(k \cdot 32^k)$.

3.1 Nondeterministic construction process

Now we give a description of our nondeterministic construction process **NDTRV** (see Fig. 1). The construction is nondeterministic as it always guesses the optimal choice correctly when running. The actual deterministic algorithm enumerates all possible choices to simulate the nondeterministic actions (see Fig. 3). Readers could refer to [5] as an example of nondeterministic algorithm. We present this construction process in order to depict the idea behind our deterministic algorithm clearly and vividly.

Let T_{start} be the initial triangulation, and T_{end} be the objective triangulation. Suppose that F is a shortest valid sequence, that is, F has the shortest length among all valid sequences. Let D_F be the DAG constructed after F according to Definition 1. **NDTRV** traverses D_F, removes the vertices of D_F in a topologically-sorted order and transforms T_{start} into T_{end}. Although D_F is unknown, for further analysis, we assume that **NDTRV** can remove and copy nodes in D_F so that it can construct an auxiliary undirected graph G and a list L based on D_F during the traversal. In later analysis we show that G is a forest, and there is a bijection between actions of **NDTRV** and nodes as well as edges of G. Obviously G and L are unknown as well. We just show that if a shortest valid sequence F exists, then D_F exists. So do G, L and Q. We can see D_F and G as conceptual or dummy graphs. We construct G instead of analysing a subgraph of D_F because one move action (see Section 3.2) of **NDTRV** may correspond to one or two edges in D_F (see Fig. 2), while there is a one-to-one correspondence between move actions and edges in G .

At the beginning of an iteration, **NDTRV** picks a necessary edge $e = \varepsilon(f_h)$ arbitrarily and nondeterministically guesses a walk W to find the underlying edge of a source node f_s. Lemma 5 shows that there exists such a walk W whose length is bounded by the length of a directed path B from f_s to f_h, and every edge e' in W is the underlying edge of some flip f' on B. **NDTRV** uses L to preserve a sequence of nodes $\Gamma = \langle f_s = v_1, ..., f_h = v_\ell \rangle$ on B, whose underlying edges are in W. Simultaneously **NDTRV** constructs a path S by copying all nodes in Γ as well as adding an undirected edge between the copy of v_i and v_{i+1} for $i = 1, ..., \ell$. S is defined as a *searching path*. The node f_h is called a *starting node*. If a starting node is precisely a source node in D_F, the searching path consists only of the copy of this starting node. When finding $\varepsilon(f_s)$, **NDTRV** removes f_s from D_F, flips $\varepsilon(f_s)$ and moves back to the previous edge $\varepsilon(v_2)$ of $\varepsilon(f_s)$ in W. If v_2 becomes a source node of D_F, **NDTRV** removes v_2 from D_F, flips $\varepsilon(v_2)$ and moves back to the previous edge $\varepsilon(v_3)$. **NDTRV** repeats the above operations until finding a node v_i in Γ which is not a source node in D_F. Then **NDTRV** uses v_i as a new starting node, and recursively guesses a walk nondeterministically from $\varepsilon(v_i)$ to find another edge which is the underlying edge of a source node as above. **NDTRV** performs these operations until the initial starting node f_h becomes a source node in D_F. Finally **NDTRV** removes f_h and flips $\varepsilon(f_h)$, terminating this iteration. **NDTRV** repeats the above iteration until T_{start} is transformed into T_{end}. We give the formal presentation of **NDTRV** in Fig. 1 and an example in Fig. 2.

3.2 Actions of the construction

Our construction process contains two types of actions operating on triangulations. The edge which the algorithm is operating on is called *the current edge*. The current triangulation is denoted by $T_{current}$.

- **(i)** Move to one edge that shares a triangle with the current edge in $T_{current}$. We formalize it as $(move, e_1 \mapsto e_2)$, where e_1 is the current edge and e_2 shares a triangle with e_1.
- **(ii)** Flip the current edge and move back to the previous edge of the current edge in W. We formalize it as $(f, e_4 \mapsto e_3)$, where f is the flip performed on the current edge, e_4 equals $\varphi(f)$ and e_3 is the previous edge of $\varepsilon(f)$ in the current walk W.

Since there are four edges that share a triangle with the current edge, there are at most four directions for an action of type (i). However, there is only one choice for an action of type (ii).

3.3 The sequence of actions

The following theorem is the main theorem for the deterministic algorithm **FLIPDT**, which bounds the length of the sequence of actions by $2|V(D_F)|$.

▶ **Theorem 4.** *There exists a sequence of actions of length at most $2|V(D_F)|$ following which we can perform a sequence of flips F' of length $|V(D_F)|$, starting from a necessary edge in T_{start}, such that F' is a topological sorting of D_F.*

In order to prove theorem 4, we need to introduce some lemmas. We will give the proof for Theorem 4 at the end of Section 3.3.

Lemma 5 shows the existence of a length-bounded walk to find an edge which is the underlying edge of a source node.

▶ **Lemma 5.** *[16] Suppose that a sequence of flips F^- is performed such that every time we flip an edge, we delete the corresponding source node in the DAG resulting from preceding*

NDTRV(T_{start}, T_{end}; D_F)
Input: the initial triangulation T_{start} and objective triangulation T_{end}. Assuming F is a shortest sequence, D_F is the corresponding DAG according to Definition 1.
/*G is an auxiliary undirected graph */
/*L is a list keeping track of searching paths for backtracking */
/*Q is a list preserving the sequence of nondeterministic actions */
/* $T_{current}$ is the current triangulation*/

a. Let $V(G)$ and $E(G)$ be empty sets, L and Q be empty lists;
b. $T_{current} = T_{start}$;
c. **While** $T_{current} \neq T_{end}$ **do**
c.1. Pick a necessary edge $e = \varepsilon(f_h)$ in $T_{current}$ arbitrarily;
c.2. Add a copy of f_h into G;
c.3. Add f_h into L;
c.4. **TrackTree**($T_{current}$, e, D_F, G, L, Q);

TrackTree($T_{current}$, $\varepsilon(f_h)$, D_F, G, L, Q) /*construct searching paths starting from $\varepsilon(f_h)$*/
1. Nondeterministically guess a walk in $T_{current}$ from $\varepsilon(f_h)$ to find $\varepsilon(f_s)$ according to Lemma 5, let $\Gamma = \langle f_s = v_1, ..., f_h = v_\ell \rangle$, where f_s is a source node in D_F, be a sequence of nodes on the backbone B whose underlying edges are in the walk W such that $\varepsilon(v_i)$ and $\varepsilon(v_{i+1})$ are consecutive in W for $i = 1, ..., \ell - 1$;
2. Add a copy of v_1,...,$v_{\ell-1}$ into G respectively;
3. Connect the copies of v_1,...,v_ℓ in G into a path;
4. Append $v_{\ell-1}$,...,v_1 to L; /*record current searching path*/
5. Append $(move, \varepsilon(v_\ell) \mapsto \varepsilon(v_{\ell-1}))$,...,$(move, \varepsilon(v_2) \mapsto \varepsilon(v_1))$ to Q; /*record actions*/
6. Remove $f_s = v_1$ from L;
7. Remove $f_s = v_1$ from D_F;
8. Flip $\varepsilon(f_s)$ in $T_{current}$ and move back to $\varepsilon(v_2)$;
9. Append $(f_s, \varphi(v_1) \mapsto \varepsilon(v_2))$ to Q; /*record actions*/
10. **For** $i = 2$ **to** ℓ **do**
10.1 Nondeterministically guess if v_i is a source node in D_F;
10.2 **If** v_i is a source node of D_F **then** /*flip and move back*/
10.2.1 Remove v_i from L;
10.2.2 Remove v_i from D_F;
10.2.3 Flip $\varepsilon(v_i)$ in $T_{current}$ and move back to $\varepsilon(v_{i+1})$;
10.2.4 Append $(v_i, \varphi(v_i) \mapsto \varepsilon(v_{i+1}))$ to Q;
10.3 **Else** /*construct searching paths from v_i*/
10.3.1 **TrackTree**($T_{current}$, $\varepsilon(v_i)$, D_F, G, L, Q);

Figure 1 Nondeterministic construction **NDTRV**

deleting operations. Let f_h be a node in the remaining DAG such that $\varepsilon(f_h)$ is an edge in the triangulation T resulting from performing the sequence of flips F^-. There is a source node f_s in the remaining DAG satisfying:

(1) *There is a walk W in T from $\varepsilon(f_h)$ to $\varepsilon(f_s)$.*

(2) *There is a directed path B from f_s to f_h in the remaining DAG that we refer to as the backbone of the DAG.*

(3) *The length of W is at most that of B.*

(4) *Any edge in W is the underlying edge of a flip in B, that is, $W = \langle \varepsilon(v_1), ..., \varepsilon(v_\ell) \rangle$, where $v_1 = f_s, ..., v_\ell = f_h$ are nodes in B and there is a directed path B_i from v_i to v_{i+1} for $i = 1, ..., \ell - 1$ such that $B_i \subset B$. A searching path is an undirected path constructed by copying $v_1, ..., v_\ell$ and connecting the copies of $v_1, ..., v_\ell$ into a path.*

▶ **Lemma 6.** ***NDTRV*** *transforms T_{start} into T_{end} with the minimum number of flips and stops in polynomial time if it correctly guesses every moving and flipping action.*

Proof. Suppose that F is a shortest valid sequence. According to Lemma 5, every edge

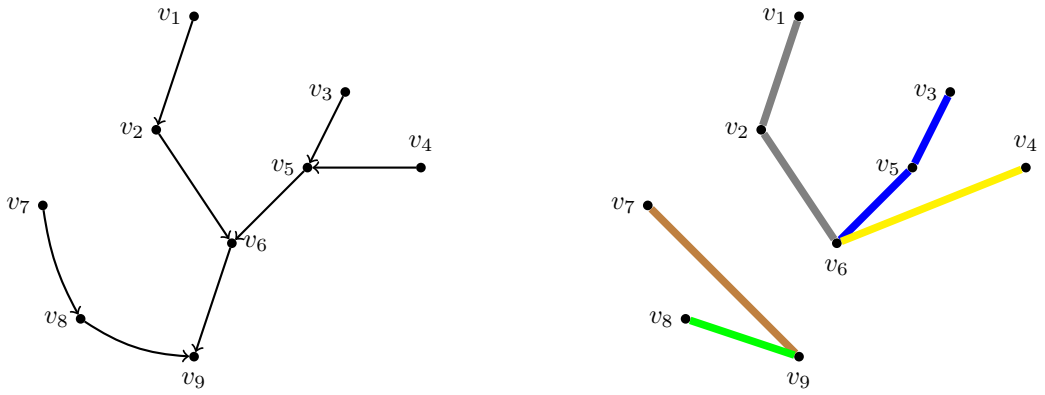


Figure 2 An example for constructing G. The graph on the left is the DAG. The graph on the right is the auxiliary graph G constructed by **NDTRV**. $\varepsilon(v_6)$ is the first chosen necessary edge, and $\varepsilon(v_9)$ is the second one. Edges with the same color belong to the same searching path. Searching paths in the same connected component are constructed in the same iteration.

flipped in **NDTRV** is the underlying edge of a source node in the remaining graph of D_F, and every node removed from the remaining graph of D_F in **NDTRV** is a source node. If $T_{current}$ is equal to T_{end} but D_F is not empty, then there exists a valid sequence F' which is shorter than F, contradicting that F is a shortest valid sequence. Thus **NDTRV** traverses D_F, removes all nodes of D_F in a topologically-sorted order and transforms T_{start} into T_{end} with the minimum number of flips by Lemma 2. Since the diameter of a transformations graph $G_T(\mathcal{P})$ is $O(n^2)$ [17], **NDTRV** stops in polynomial time. ◀

NDTRV constructs G, Q and L during its execution. Lemma 7, Lemma 8 and Lemma 9 show some properties of G, Q, L and D_F.

▶ **Lemma 7.** *At the end of **NDTRV**, the graph G consists of all searching paths constructed during the execution of **NDTRV**. Moreover, the list Q contains a sequence of nondeterministic actions starting from a necessary edge in T_{start} following which we can perform a sequence of flips F' of length $|V(D_F)|$ such that F' is a topological sorting of D_F.*

Proof. From the construction of G, we see that one node is added into G when it is in some searching path. Moreover, we add an edge between two nodes in G if and only if they are adjacent in a searching path. It follows that G consists of all searching paths constructed during the execution of **NDTRV**.

By the proof of Lemma 6, **NDTRV** removes the nodes of D_F in a topologically-sorted order. Since Q records every action of **NDTRV**, if we perform the actions in Q, we get a valid sequence F'. This completes the proof. ◀

▶ **Lemma 8.** *The following statements are true:*

(1) *At the end of any iteration of **NDTRV**, all nodes whose copies are in the searching paths constructed in this iteration are removed from D_F, and L becomes empty.*

(2) *During every step of **NDTRV**, there exists a directed path in the remaining graph of D_F passing through every node in L from the last node of L to the first node of L.*

(3) *All searching paths in G are edge-disjoint. Any two searching paths which belong to two different iterations respectively are node-disjoint. Searching paths constructed in each iteration form a tree respectively, namely a track tree. Moreover, G is a forest.*

Proof. Suppose we are in the first iteration. We name the searching paths in this iteration $S_1, ..., S_m$ by the order they are constructed. Suppose now we have constructed searching paths $S_1, ..., S_i$ and $i < m$. We prove that statements (2) and (3) hold for the first iteration by induction on i. When $i = 1$, the statement is true because there is only one searching path S_1 forming a tree, and there is a directed path in the remaining graph of D_F passing through every node in L from the last to the first according to Lemma 5. Assume that the statements are true for any $i < m$, that is, $S_1, ..., S_i$ are edge-disjoint, and they form a tree. Moreover, there exists a directed path P in the remaining graph of D_F passing through every node in L from the last to the first (inductive hypothesis). The next searching path we will construct is S_{i+1}. Note that before constructing S_{i+1}, we may remove the last node of L repeatedly from the remaining graph of D_F and L if it is a source node in the remaining graph of D_F. Since every node we removed was the beginning node of the directed path P, there was still a directed path passing through every node in L from the last to the first. Let $f_{start,i+1}$ be the starting node whose copy is in S_{i+1}, which means that $f_{start,i+1}$ is the last node in L, and $f_{start,i+1}$ is not a source node in the remaining graph of D_F. By the inductive hypothesis and the analysis above, there is a directed path P_1 from $f_{start,i+1}$ to f_h which is the first node of L. Let f'_s be a source node in the remaining graph of D_F such that the copy of f'_s is in S_{i+1}. By Lemma 5, there is a directed path P_2 in the remaining graph of D_F passing through the nodes whose copies are in S_{i+1} from f'_s to $f_{start,i+1}$. We argue that P_1 has only one common node with P_2, which is exactly $f_{start,i+1}$. Otherwise, there is a directed cycle induced by some of the nodes on P_1 and P_2 in the remaining graph of D_F, contradicting the acyclicity of D_F. It follows that there is a directed path in the remaining graph of D_F passing through every node in L from the last node of L to the first node of L. Since S_{i+1} cannot contain any copies of the nodes that have been removed from the remaining graph of D_F and L, S_{i+1} has only one common node with the searching paths S_1,...,S_i, namely the copy of $f_{start,i+1}$. It is implied that a node of D_F cannot be added to L more than once for the same reason. As a result, S_1,...,S_i and S_{i+1} are edge-disjoint, and S_1,...,S_{i+1} form a tree. The tree formed by S_1,...,S_m is a track tree. This completes the inductive proof.

We prove statement (1) in the first iteration by contradiction. Suppose that there exists one or several nodes left in L at the end of an iteration, and w is the last one in L. Suppose w was added to L when searching path S_p was constructed and **TrackTree**(v_p) was called. Let u be the node appended to L on its heel. Such a node u existed since w was not a source node then. Otherwise w was removed, resulting in a contradiction as one node of D_F cannot be added to L more than once (according to the proof in the above paragraph). Since w is the last one in L at the end of this iteration, u was removed, and **NDTRV** moved back to $\varepsilon(w)$. According to **NDTRV**, w has never become a source node in D_F. Otherwise, w has been removed from L, leading to a contradiction since one node of D_F cannot be added to L more than once. Thus **TrackTree**(w) was called. The terminal condition of **TrackTree**(w) is that w becomes a source node in D_F, and w is removed from L and D_F. This implies that this iteration does not end, a contradiction. It follows that L becomes empty at the end of the first iteration. According to **NDTRV**, all nodes whose copies are in the searching paths constructed in the first iteration are removed from D_F.

We prove that statements (1), (2) and (3) hold for the whole procedure by induction. We have proved that they are true for the first iteration. Suppose that there are t iterations in the procedure, and statements (1), (2) and (3) hold for the first j iterations for any $1 \leq j < t$. By the inductive hypothesis, at the end of the j-th iteration, L is empty. A node x whose copy is in the searching paths constructed in the first j iterations can never appear in L in the

$(j+1)$-th iteration because it has been removed from D_F. It follows that the searching paths constructed in the $(j+1)$-th iteration are node-disjoint with the searching paths constructed in the first j iterations. Thus they are edge-disjoint. Since L is empty, it is not difficult to see that the proof for the first iteration also holds for the $(j+1)$-th iteration. It follows that the searching paths constructed in the $(j+1)$-th iteration form a tree that is node disjoint with other j trees belonging to the first j iterations respectively, that is, these trees form a forest all together. By Lemma 7, G consists of all searching paths constructed during the execution of **NDTRV**. It follows that G is a forest. This completes the inductive proof for the whole procedure. ◀

▶ **Lemma 9.** *$|V(G)|$ is equal to $|V(D_F)|$.*

Proof. According to **NDTRV**, a node v is added to L if and only if its copy is added to G, and v is removed from L if and only if v is removed from D_F. By the proof in Lemma 8, no node in D_F can be added to L more than once. By the proof of Lemma 6, all nodes of D_F are removed by **NDTRV**. It follows that all nodes in D_F have exactly one copy in G. Thus $|V(G)| = |V(D_F)|$. ◀

We give the proof of Theorem 4 below.

Proof. (Theorem 4) In the procedure of constructing G in Fig. 1, we construct a list Q which consists of actions of type (i) and (ii). We claim that Q is exactly the sequence satisfying the requirement of this theorem. By Lemma 7, the number of actions of type (ii) in Q is exactly $|V(D_F)| = |V(G)|$. **NDTRV** adds an action of type (i) to Q if and only if it adds an an edge to $E(G)$. Moreover, Q and $E(G)$ are both empty at the beginning of **NDTRV**. It follows that there is a one-to-one correspondence between actions of type (i) in Q and edges in G. By Lemma 8, G is a forest. As a result, $|E(G)| \leq |V(G)|$, and the length of Q is bounded by $|E(G)| + |V(G)| \leq 2|V(G)| = 2|V(D_F)|$. ◀

3.4 The deterministic algorithm

Now we are ready to give the deterministic algorithm **FLIPDT** for the Parameterized Flip Distance problem. The specific algorithm is presented in Fig. 3. As mentioned above, we assume that **NDTRV** is always able to guess the optimal choice correctly. In fact, **FLIPDT** achieves this by trying all possible sequences of actions and partitions of k. At the top level, **FLIPDT** branches into all partitions of k, namely $(k_1, ..., k_t)$ satisfying $k_1 + ... + k_t = k$ and $k_1, ..., k_t \geq 1$, in which k_i $(i = 1, ..., t)$ equals the size of the track tree A_i constructed during the i-th iteration.

Suppose that **FLIPDT** is under some partition $(k_1, ..., k_t)$. Let $T^0_{iteration} = T_{start}$. **FLIPDT** permutates all necessary edges in T_{start} in the lexicographical order, and the ordering is denoted by O_{lex}. Here we number the given points of $\mathcal{P}$ in the Euclidean plane from 1 to n arbitrarily and label one edge by a tuple consisting of two numbers of its endpoints (the smaller number is ahead of the other one). Thus we can order the edges lexicographically. **FLIPDT** performs t iterations. At the beginning of the i-th iteration, $i = 1, ..., t$, we denote the current triangulation by $T^{i-1}_{iteration}$. For $i = 1, ..., t$, $T^i_{iteration}$ is also the triangulation resulting from the execution of the i-th iteration. At the beginning of the i-th iteration $(i = 1, ..., t)$, **FLIPDT** repeatedly picks the next edge in O_{lex} until finding a necessary edge e belonging to $T^{i-1}_{iteration}$ (just pick the first edge in O_{lex} in the first iteration). Note that one edge in O_{lex} may not be a necessary edge anymore with respect to $T^{i-1}_{iteration}$. Moreover, if **FLIPDT** reaches the end of O_{lex} but does not find a necessary edge belonging

FLIPDT(T_{start}, T_{end}, k)
Input: two triangulations T_{start} and T_{end} of a point set $\mathcal{P}$ in the Euclidean plane and an integer k.
Output: return YES if there exists a sequence of flips of length k that transforms T_{start} into T_{end}; otherwise return NO.

1. **For** each partition $(k_1, ..., k_t)$ of k satisfying $k_1 + k_2 + ... + k_t = k$ and $k_1, ..., k_t \geq 1$ **do**
1.1 Order all necessary edges in T_{start} lexicographically and denote this ordering by O_{lex};
1.2 **FDSearch**(T_{start},1,$(k_1, ..., k_t)$); /*iteration 1 *distributed with* k_1 */
2. **Return** NO;

FDSearch(T,i,$(k_1, ..., k_t)$) /*the concrete branching procedure*/
Input: a triangulation T, an integer i denoting that the algorithm is at the i-th iteration and a partition $(k_1, ..., k_t)$ of k.
Output: return YES if the instance is accepted.

1. Repeatedly pick the next edge in O_{lex} until finding a necessary edge e with respect to T and T_{end};
2. **If** it reaches the end of O_{lex} but finds no necessary edge in T **then**
2.1 Update O_{lex} by permutating all necessary edges in $T^{i-1}_{iteration}$ in lexicographical order, and pick the first edge e in O_{lex};
3. **For** each possible sequence of actions seq_i of length $2k_i - 1$ **do**
3.1 T' = **Transform**(T,seq_i,e);
3.2 **If** $i < t$ **then** /*continue to the next iteration distributed with* k_{i+1} */
3.2.1 **FDSearch**(T',$i + 1$,$(k_1, ..., k_t)$);
3.3 **Else if** $i = t$ and $T' = T_{end}$ **then** /*compare* T' *with* T_{end} */
3.3.1 **Return** YES;

Transform(T,s,e) /*subprocess for transforming triangulations*/
Input: a triangulation T, a sequence of actions s and a starting edges e.
Output: a new triangulation T'.

1. Perform a sequence of actions s starting from e in T, getting a new triangulation T';
2. **Return** T';

Figure 3 The deterministic algorithm for the Flip Distance problem

to $T^{i-1}_{iteration}$, it needs to update O_{lex} by clearing O_{lex} and permutating all necessary edges in $T^{i-1}_{iteration}$ lexicographically, and choose the first edge in the updated ordering O_{lex}. Then **FLIPDT** branches into every possible sequence of actions seq_i of length $2k_i - 1$. For each enumeration of seq_i, **FLIPDT** performs the actions of seq_i on $T^{i-1}_{iteration}$ starting from e and gets a new triangulation $T^{i}_{iteration}$. For every triangulation $T^{i}_{iteration}$ resulting from seq_i, **FLIPDT** performs the $(i + 1)$-th iteration on $T^{i}_{iteration}$. **FLIPDT** proceeds as above from the first iteration to the last iteration. When **FLIPDT** finishes the last iteration, it judges if the resulting triangulation $T^{t}_{iteration}$ is equal to T_{end}. If they are equal, the input instance is a yes-instance. Otherwise, **FLIPDT** rejects this case and proceeds.

Now we analyse how to enumerate all possible sequences of length $2k_i - 1$. By the proof of Lemma 8 and Theorem 4, the searching paths constructed during each iteration form a track tree in which a node corresponds to an action of type (ii) while an edge corresponds to an action of type (i). It follows that the number of actions of type (ii) is k_i, and the number of actions of type (i) is $k_i - 1$ since the number of nodes equals the number of edges plus one in a tree. According to **NDTRV**, the last action γ in seq_i must be of type (ii), and in any prefix of $seq_i - \gamma$ the number of actions of type (i) must not be less than that of type (ii). Thus **FLIPDT** only needs to enumerate all sequences of length $2k_i - 1$ satisfying the above constraints.

The following theorem proves the correctness of the algorithm **FLIPDT**.

▶ **Theorem 10.** *Let* (T_{start}, T_{end}, k) *be an input instance.* ***FLIPDT*** *is correct and runs in time* $O^*(k \cdot 32^k)$.

Proof. Suppose that (T_{start}, T_{end}, k) is a yes-instance. There must exist a sequence of flips F of length k such that $T_{start} \xrightarrow{F} T_{end}$. Thus D_F exists according to Definition 1. By **NDTRV**, Lemma 7 and Lemma 8, there exists an undirected graph G consisting of a set of node-disjoint track trees $A_1, ..., A_t$. Moreover, Theorem 4 shows that there exists a sequence of actions Q following which we can perform all flips of D_F in a topologically-sorted order. Due to **NDTRV**, Q consists of several subsequences $seq_1, ..., seq_t$, in which seq_i is constructed in the i-th iteration and corresponds to the track tree A_i for $i = 1, ..., t$. Supposing the size of A_i is λ_i for $i = 1, ..., t$ satisfying $\lambda_1 + ... + \lambda_t = k$, seq_i contains λ_i actions of type (ii) corresponding to the nodes of A_i as well as $\lambda_i - 1$ actions of type (i) corresponding to the edges of A_i. **FLIPDT** guesses the size of every track tree by enumerating all possible partitions of k into $(k_1, ..., k_t)$ such that $k_1 + ... + k_t = k$ and $k_1, ..., k_t \geq 1$. We say that k_i is distributed to the i-th iteration or the distribution for the i-th iteration is k_i for $i = 1, ..., t$.

We claim that **FLIPDT** is able to perform a sequence Σ of actions which correctly guesses every subsequence $seq_1, ..., seq_t$ of the objective sequence Q, that is, Σ is a concatenation of $seq_1, ..., seq_t$. Suppose that **FLIPDT** has completed i iterations. We prove this claim by induction on i. At the first iteration, **FLIPDT** starts by picking the first necessary edge e_1 in list O_{lex}. In the first iteration of constructing Q, **NDTRV** starts by picking an arbitrary necessary edge. Without loss of generality, it chooses e_1 and construct seq_1 starting from e_1. The length of seq_1 is $2\lambda_1 - 1$. Since **FLIPDT** tries every distribution in $\{1, ..., k\}$ for the first iteration and $1 \leq \lambda_1 \leq k$, there is a correct guess of the distribution equal to λ_1 for this iteration. Under this correct guess, **FLIPDT** tries all possible sequences of actions of length $2\lambda_1 - 1$ starting from e_1. It follows that **FLIPDT** is able to perform a sequence that is equals to seq_1 in the first iteration resulting in a triangulation T_1.

Suppose that the claim is true for any first i iterations $(1 \leq i < t)$. That is, under some guess for the partition of k, $\lambda_1, ..., \lambda_i$ are distributed to the first i iterations respectively. Moreover, **FLIPDT** has completed i iterations and performed a sequence of actions $seq_{concat,i}$, which is equal to the concatenation of $seq_1, ..., seq_i$, resulting in a triangulation T_i. Based on T_i and $seq_{concat,i}$, **FLIPDT** is ready to perform the $(i+1)$-th iteration. Suppose that **FLIPDT** picks e_{i+1} from O_{lex}. Let us see the construction of Q in **NDTRV**. Suppose **NDTRV** has constructed the first i track trees $A_1, ..., A_i$, and it is ready to begin a new iteration by arbitrarily picking a necessary edge in the current triangulation. Since **FLIPDT** correctly guessed and performed the first i subsequences of Q, T_i is exactly equal to the current triangulation in **NDTRV**. Thus e_{i+1} is a candidate edge belonging to the set of all selectable necessary edges for **NDTRV** in this iteration. Without loss of generality, it chooses e_{i+1} and constructs seq_{i+1} of length $2\lambda_{i+1} - 1$ starting from e_{i+1}. Since the sizes of $A_1, ..., A_i$ are $\lambda_1, ..., \lambda_i$ respectively, we get that $1 \leq \lambda_{i+1} \leq k - (\lambda_1 + ... + \lambda_i)$. We argue that **FLIPDT** is able to perform a sequence that is equal to the concatenation of $seq_1, ..., seq_{i+1}$. Since the edges in O_{lex} are ordered lexicographically and **FLIPDT** chooses necessary edges in a fixed manner, **FLIPDT** is sure to choose e_{i+1} to begin the $(i+1)$-th iteration for every guessed sequence in which the first i subsequences are equal to seq_1,...,seq_i respectively. Thus **FLIPDT** actually tries every distribution in $\{1, ..., k - (\lambda_1 + ... + \lambda_i)\}$ for the $(i+1)$-th iteration starting from e_{i+1} based on T_i and $seq_{concat,i}$. It follows that there is a correct guess of distribution for the $(i+1)$-th iteration which is equal to λ_{i+1}. Under this correct guess of distribution, **FLIPDT** tries all possible sequences of length $2\lambda_{i+1} - 1$ starting from e_{i+1} on T_i based on $seq_{concat,i}$, ensuring that one of them is equal to seq_{i+1}. It follows that the claim is true for the first $i+1$ iterations. This completes the inductive proof for the claim.

If (T_{start}, T_{end}, k) is a yes-instance, the action sequence Q of length at most $2k$ exists and the deterministic algorithm can find such a sequence. Otherwise, there is no valid sequence F of length k. Thus there is no such action sequence Q. As a result, **FLIPDT** returns NO. It is proved that **FLIPDT** decides the given instance (T_{start}, T_{end}, k) correctly.

Finding and ordering all necessary edges in T_{start} takes $O(n+k\log k)$ time, and **FLIPDT** may update the ordering O_{lex} at the beginning of each iteration. The number of partitions of k is known as the composition number of k, which is 2^{k-1}. Under each partition $(k_1, ..., k_t)$ of k and for each k_i, $i = 1, ..., t$, we enumerate all possible subsequences of actions in which there are k_i actions of type (ii) and $k_i - 1$ actions of type (i). It follows that the number of all possible subsequences is bounded by $\binom{2(k_i-1)}{k_i-1} \times 4^{k_i-1} = O^*(16^{k_i})$ since there are four choices for action (i) and one choice for action (ii). Here we use Stirling's approximation $n! \approx \sqrt{2\pi n}(n/e)^n$ and get that $\binom{2(k_i-1)}{k_i-1} = O^*(4^{k_i})$. It follows that there are $O^*(16^{k_1}) \times O^*(16^{k_2}) \times ... \times O^*(16^{k_t}) = O^*(16^k)$ cases under each partition. Since for each case we can perform the sequence of actions in $O(k)$ time, and the resulting triangulation can be compared to T_{end} in $O(k)$ time, the running time of the whole algorithm is bounded by $O^*(k \cdot 2^{k-1} \cdot (n + k\log k) + k \cdot 2^{k-1} \cdot 16^k) = O^*(k \cdot 32^k)$.

According to the definition of the Flip Distance problem, we need to decide if we can find a shorter valid flip sequence for the given triangulations T_{start} and T_{end}. This is achieved by calling **FLIPDT** on each instance (T_{start}, T_{end}, k') for $k' = 0, ..., k$. The running time is bounded by $\sum_{k'=0}^{k} O^*(k' \cdot 32^{k'}) = O^*(k \cdot 32^k)$. ◀

4 Conclusion

In this paper we presented an FPT algorithm running in time $O^*(k \cdot 32^k)$ for the Parameterized Flip Distance problem, improving the previous $O^*(k \cdot c^k)$-time ($c \leq 2 \times 14^{11}$) FPT algorithm by Kanj and Xia [15]. An important related problem is computing the flip distance between triangulations of a convex polygon, whose traditional complexity is still unknown. Although our algorithm can be applied to the case of convex polygon, it seems that an $O(c^k)$ algorithm with smaller c for this case probably exists due to its more restrictive geometric property. In addition, whether there exists a polynomial kernel for the Parameterized Flip Distance problem is also an attractive problem.

References

1 Oswin Aichholzer, Ferran Hurtado, and Marc Noy. A lower bound on the number of triangulations of planar point sets. *Computational Geometry*, 29(2):135–145, 2004. doi:10.1016/j.comgeo.2004.02.003.

2 Oswin Aichholzer, Wolfgang Mulzer, and Alexander Pilz. Flip distance between triangulations of a simple polygon is NP-complete. *Discrete & Computational Geometry*, 54(2):368–389, 2015. doi:10.1007/s00454-015-9709-7.

3 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, Michael T. Hallett, and Harold T. Wareham. Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences*, 11(1):49–57, 1995. doi:10.1093/bioinformatics/11.1.49.

4 Jianer Chen. Parameterized computation and complexity: A new approach dealing with np-hardness. *J. Comput. Sci. Technol.*, 20(1):18–37, 2005. doi:10.1007/s11390-005-0003-7.

5 Jianer Chen, Donald K. Friesen, Weijia Jia, and Iyad A. Kanj. Using nondeterminism to design efficient deterministic algorithms. *Algorithmica*, 40(2):83–97, 2004. doi:10.1007/s00453-004-1096-z.

6 Jianer Chen, Chao Xu, and Jianxin Wang. Dealing with 4-variables by resolution: An improved maxsat algorithm. *Theor. Comput. Sci.*, 670:33–44, 2017. doi:10.1016/j.tcs.2017.01.020.

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

8 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science, FOCS, Palm Springs, USA*, pages 150–159, 2011. doi:10.1109/FOCS.2011.23.

9 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM J. Discrete Math.*, 27(1):290–309, 2013. doi:10.1137/110843071.

10 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008.

11 Gerald E. Farin. *Curves and surfaces for computer-aided geometric design - a practical guide (4. ed.)*. Computer science and scientific computing. Academic Press, 1997.

12 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, Portland, USA*, pages 142–151, 2014. doi:10.1137/1.9781611973402.10.

13 Bernd Hamann. Modeling contours of trivariatc data. *Mathematical Modeling and Numerical Analysis*, 26:51–75, 1992.

14 Ferran Hurtado, Marc Noy, and Jorge Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, 1999. doi:10.1007/PL00009464.

15 Iyad A. Kanj and Ge Xia. Flip Distance is in FPT time $O(n + k \cdot c^k)$. In *proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science, STACS, Garching, Germany*, pages 500–512, 2015. doi:10.4230/LIPIcs.STACS.2015.500.

16 Iyad A. Kanj and Ge Xia. Computing the flip distance between triangulations. *to appear in Discrete & Computational Geometry*, 2017.

17 Charles L. Lawson. Transforming triangulations. *Discrete Mathematics*, 3(4):365–372, 1972. doi:10.1016/0012-365X(72)90093-3.

18 Anna Lubiw and Vinayak Pathak. Flip distance between two triangulations of a point set is NP-complete. *Computational Geometry*, 49:17–23, 2015. doi:10.1016/j.comgeo.2014.11.001.

19 Joan M. Lucas. An improved kernel size for rotation distance in binary trees. *Information Processing Letters*, 110(12-13):481–484, 2010. doi:10.1016/j.ipl.2010.04.022.

20 Alexander Pilz. Flip distance between triangulations of a planar point set is APX-hard. *Computational Geometry*, 47(5):589–604, 2014. doi:10.1016/j.comgeo.2014.01.001.

21 Larry L. Schumaker. Triangulations in CAGD. *IEEE Computer Graphics and Applications*, 13(1):47–52, 1993. doi:10.1109/38.180117.

22 Feng Shi, Jianxin Wang, Yufei Yang, Qilong Feng, Weilong Li, and Jianer Chen. A fixed-parameter algorithm for the maximum agreement forest problem on multifurcating trees. *SCIENCE CHINA Information Sciences*, 59(1):1–14, 2016. doi:10.1007/s11432-015-5355-1.

23 Daniel D. Sleator, Robert E. Tarjan, and William P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, STOC, Berkeley, USA*, pages 122–135, 1986. doi:10.1145/12130.12143.

Reversible Kleene Lattices*

Paul Brunet

University College London, United Kingdom
paul@brunet-zamansky.fr

Abstract

We investigate the equational theory of reversible Kleene lattices, that is algebras of languages with the regular operations (union, composition and Kleene star), together with the intersection and mirror image. Building on results by Andréka, Mikulás and Németi from 2011, we construct the free representation of this algebra. We then provide an automaton model to compare representations. These automata are adapted from Petri automata, which we introduced with Pous in 2015 to tackle a similar problem for algebras of binary relations. This allows us to show that testing the validity of equations in this algebra is decidable, and in fact ExpSpace-complete.

1998 ACM Subject Classification F.4.3 Formal Languages, F.1.1 Models of Computation, F.3.2 Semantics of Programming Languages.

Keywords and phrases Kleene algebra, Automata, Petri nets, Decidability, Complexity.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.66

1 Introduction

We are interested in algebras of languages, equipped with the constants empty language (0), unit language (1, the language containing only the empty word), the binary operations of union (+), intersection (∩), and concatenation (·), and the unary operations of Kleene star ($\_^\star$) and mirror image, also called converse, ($\_^\smile$). We call these algebras *reversible Kleene lattices*. Given a finite set of variables X, and two terms e, f built from variables and the above operations, we say that the equation $e = f$ (respectively inequation $e \leq f$) is *valid* if the corresponding equality (resp. containment) holds universally. A *free representation* is a set $\mathcal{M}$ together with a map h from terms to elements of $\mathcal{M}$ such that $e = f$ is valid if and only if h maps e and f to the same element of $\mathcal{M}$.

It is well known that to any term over this syntax, one can associate a regular language, and that comparing regular languages is decidable. In fact, the problem of comparing regular expressions with intersection with respect to regular language equivalence is ExpSpace-complete [12]. The difference with the work presented here is that we are considering equations which are stable under substitution. Formally, this means that we do not interpret the letter a as the singleton language $\{a\}$, but rather as a universally quantified variable ranging over all languages. What is remarkable however is that testing the validity of equations in reversible Kleene lattices is still an ExpSpace-complete problem, as we show in this paper. Several fragments of this algebra have been studied:

Kleene algebra (KA) [9]: if we restricts ourselves to the operators of regular expressions (0, 1, +, ·, and $\_^\star$), then the free representation is the set of regular languages, with the usual definition of the language of an expression. Testing the validity of equations in KA is thus a PSpace-complete problem [19, 14].

* An extended version of this abstract is available at https://hal.archives-ouvertes.fr/hal-01474911, [5].

42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017).
Editors: Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin; Article No. 66; pp. 66:1–66:14

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

Kleene algebra with converse (KAC) [3]: if we add to KA the converse operation, then the free representation consists of regular expressions over a duplicated alphabet, with a letter a' denoting the converse of the letter a. The associated decision problem is still in PSpace.

Identity-free Kleene lattices (KL^-) [1]: this algebra stems from the operators 0, +, $\cdot$, $\cap$ and $\_^+$, where the latter is the non-zero iteration. Andréka Mikulás and Németi studied this fragment, and showed that the free representation of this algebra consists of languages of series-parallel graphs, downward closed with respect to some graph preorder. We reformulated their results with Pous [6], and introduced a new class of automata, called Petri automata, able to recognise these languages of graphs. In [6] we provided a decision procedure to compare these automata, thus yielding an ExpSpace decision procedure for the equational theory of this algebra. It is in fact ExpSpace-complete, thanks to some simple adaptation of a result by Fürer [12].

The present work is then an extension of identity-free Kleene lattices, by adding unit and mirror image. The addition of mirror image is fairly simple, relying mainly on ideas from [3]. However, the seemingly small addition of 1 yields some complications. In fact, in [1, 6] there is a free representation of Kleene allegories, an algebra over the same signature as reversible Kleene lattices, but whose intended model is binary relations rather than languages. In this context, adding 1 means moving from *series-parallel* graphs to graphs of tree-width 2, which might have cycles. This is a significant problem for automata based decision procedures.

In the context of languages, adding 1 yields other problems. However, the free representation we get for reversible Kleene lattices remains more tractable than that of Kleene allegories. In particular we do not create cycles in series parallel graphs, but rather have to collect additional information. Let us illustrate the kind of reasoning we develop to study these algebras with the following inequation: $c \cdot (1 \cap a) \leq a \cdot c$. On the left hand side (LHS), the term $1 \cap a$ appears. This term is either equal to 1 if the empty word belongs to language a, or 0 otherwise. In the first case, the LHS is equal to c and we have $1 \leq a$, meaning that $c = 1 \cdot c \leq a \cdot c$. In the second case the LHS is equal to 0, which is contained in $a \cdot c$ as well. The key observation here is that the second case does not really matter: in a term build out of concatenations, intersections, converse, variables and units, if 0 appears somewhere then the term will always evaluate to 0 and thus be contained in any other term. The free representations we develop for union-free terms consist of pairs of a representation of a 1-free term and a set of language variables which are assumed to contain the empty word. This allows us to make the reasoning we used above automatic.

Following an approach similar to [6], we construct in Section 2 the free representation of reversible Kleene lattices, and introducing a new Petri net-based automata model we show in Section 3 that testing the validity of equations is a decidable problem, and in fact an ExpSpace-complete one. We conclude and list some perspectives in Section 4.

Basic definitions and notations

For a pair $p = \langle x, y \rangle$, we denote by $\pi_1(p) = x$ the first projection, and by $\pi_2(p) = y$ the second projection. The set of functions from a set A to a set B is written $A \to B$, and the set of partial functions from A to B is written $A \rightharpoonup B$. The number of elements of a finite set A is written $|A|$. The empty word is denoted by ε, and the set of words over the alphabet Σ is $\Sigma^\star$. If $w = x_1 \dots x_n$ is a word of length n, $w[i, j]$ is the word $x_i \dots x_j$ if $i \leqslant j$, and undefined otherwise. If f is a function from some set X to $\{0, \dots, n\}$, x, y are elements of X, and if $f(x) \leqslant f(y)$, we use the notation $w^f[x, y]$ for the word $w[f(x), f(y)]$.

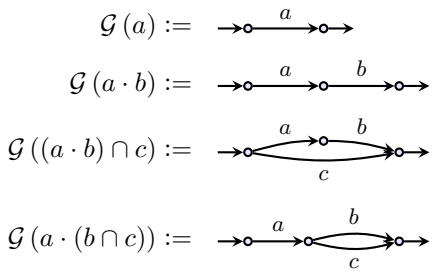


Figure 1 Graphs associated to terms.

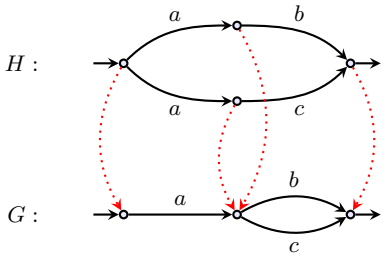


Figure 2 Graph homomorphism.

Let X be a finite set of variables, we define $\dot{A} := A \cup \{a^\smile \mid a \in A\}$ for every subset $A \subseteq X$. The set $\dot{X}$ is called the duplicated alphabet, we let α, β range over $\dot{X}$. Expressions over X are given by the following grammar:

$$e, f ::= 0 \mid 1 \mid x \mid e^\smile \mid e + f \mid e \cdot f \mid e \cap f \mid e^\star. \qquad (x \in X)$$

The set of expressions over X is written $\mathcal{E}\langle X \rangle$. The size of an expression e, written $|e|$, is its number of symbols, *i.e.* the number of vertices in its syntax tree. Most of the time, we will implicitly assume that the converse operator only appears as $x^\smile$, with $x \in X$. This is not restrictive, as every expression can be transformed linearly such that this property holds. Given a second alphabet Σ, an interpretation is a map $\sigma : X \to \mathcal{P}(\Sigma^\star)$ which associates to every variable a a language $\sigma(a)$. This map can be uniquely extended to a homomorphism $\widehat{\sigma} : \mathcal{E}\langle X \rangle \to \mathcal{P}(\Sigma^\star)$ defined inductively:

$$\widehat{\sigma}(0) = \emptyset \qquad \widehat{\sigma}(1) = \{\varepsilon\} \qquad \widehat{\sigma}(a) = \sigma(a) \qquad \widehat{\sigma}(e^\smile) = \widehat{\sigma}(e)^\smile = \{x_n \dots x_1 \mid x_1 \dots x_n \in \widehat{\sigma}(e)\}$$

$$\widehat{\sigma}(e + f) = \widehat{\sigma}(e) \cup \widehat{\sigma}(f) \qquad \widehat{\sigma}(e \cdot f) = \widehat{\sigma}(e) \cdot \widehat{\sigma}(f) \qquad \widehat{\sigma}(e \cap f) = \widehat{\sigma}(e) \cap \widehat{\sigma}(f)$$

$$\widehat{\sigma}(e^\star) = \widehat{\sigma}(e)^\star = \{w_1 \dots w_n \mid w_i \in \widehat{\sigma}(e)\}.$$

We say that $e = f$ (respectively $e \leq f$) is valid, and write $\mathrm{Lang} \models e = f$ (resp. $\mathrm{Lang} \models e \leq f$), when for every interpretation σ, we have $\widehat{\sigma}(e) = \widehat{\sigma}(f)$ (resp. $\widehat{\sigma}(e) \subseteq \widehat{\sigma}(f)$).

It is interesting to note that as decision problems, the validity of equations and that of inequations are equivalent. Indeed, the following equivalences hold:

$$\mathrm{Lang} \models e = f \Leftrightarrow \mathrm{Lang} \models e \leq f \wedge \mathrm{Lang} \models f \leq e \qquad \mathrm{Lang} \models e \leq f \Leftrightarrow \mathrm{Lang} \models e + f = f.$$

2 The free representation of reversible Kleene lattices

2.1 Intuitions

First introduced in the context of relation algebra [2, 11], directed labelled 2-pointed graphs can be used to describe the algebra of languages over the signature $\langle \cdot, \cap \rangle$. First, we associate to every term u over this signature such a graph $\mathcal{G}(u)$. See Figure 1 for examples. The set of such graphs is equipped with a preorder: G is smaller than H if there is a graph homomorphism from H to G. Such a homomorphism is illustrated in Figure 2.

Already, this gives us a clue as to the (in)equational theory: the inequation $u \leq v$ is valid if and only if $\mathcal{G}(u)$ is smaller than $\mathcal{G}(v)$. Moving to identity-free Kleene lattices, *i.e.* to

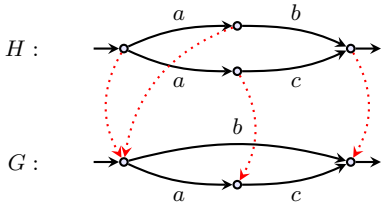


Figure 3 Weak graph morphism: $\langle G, \{a\}\rangle \blacktriangleleft \langle H, \emptyset\rangle$

the signature $\langle 0, +, \cdot, \cap, \_^+\rangle$, we associate to every term e a set $\mathcal{G}(e)$ of such graphs, and then take its downward closure $\mathcal{G}(e)\downarrow$ with respect to the preorder. This yields the free representation of this algebra: the equation $e = f$ is valid if and only if $\mathcal{G}(e)\downarrow = \mathcal{G}(f)\downarrow$.

To move from identity free Kleene lattices to reversible Kleene lattices, two steps are necessary: we need to add the converse, and to add the constant 1. The first step is somewhat straightforward, thanks to results by Ésik et al. [3]: they showed that the free representation of the algebra of languages with the regular operations together with converse is simply the set of regular languages over a duplicated alphabet, where we add for every letter a new letter representing its converse. This approach works well in our setting, by considering graphs labelled with the duplicated alphabet.

For the second step, we draw our inspiration from Lemma 3.4 in [1], that established that every term in $\mathcal{E}\langle X\rangle$ is equivalent to a finite sum of terms of the form $(1 \cap a \cap b \ldots) \cdot e$, where $a, b, \cdots \in X$ are letters, and 1 does not appear in e. For every interpretation $\sigma : X \to \mathcal{P}(\Sigma^\star)$, if there is some variable $x \in \{a, b, \ldots\}$ such that $\varepsilon \notin \sigma(x)$, then the interpretation of $1 \cap a \cap b \cap \ldots$ is $\emptyset$. Otherwise, if the empty word is in the interpretation of each of the $a, b, \ldots$, then the interpretation is $\{\varepsilon\}$. If we now look at the interpretation of the whole term, this means that:

$$\widehat{\sigma}((1 \cap a \cap b \cap \ldots) \cdot e) = \begin{cases} \widehat{\sigma}(e) & \text{if } \forall x \in \{a, b, \ldots\}, \varepsilon \in \sigma(x); \\ \emptyset & \text{otherwise.} \end{cases}$$

Consider now an inequation $f_1 \leq f_2$, where $f_i = (1 \cap a_{i,1} \cap \cdots \cap a_{i,n_i}) \cdot e_i$ for $i \in \{1, 2\}$. If there exists a variable $x \in \{a_{2,1}, \ldots, a_{2,n_2}\} \setminus \{a_{1,1}, \ldots, a_{1,n_1}\}$, we can build an interpretation σ such that (1) $\varepsilon \notin \sigma(x)$ and (2) $\widehat{\sigma}(f_1) \neq \emptyset$. The first condition ensures that the image of f_2 will be $\emptyset$, hence the inequation is not valid. Thus for the inequation to be valid, we need that $\{a_{2,1}, \ldots, a_{2,n_2}\} \subseteq \{a_{1,1}, \ldots, a_{1,n_1}\}$. Furthermore, for every σ such that there is an $a_{1,i}$ whose interpretation does not contain the empty word, the image of f_1 will be $\emptyset$, which is trivially contained in the image of the f_2. We reach the following equivalence: $f_1 \leq f_2$ is valid if and only if (1) $\{a_{2,1}, \ldots, a_{2,n_2}\} \subseteq \{a_{1,1}, \ldots, a_{1,n_1}\}$ and (2) for every interpretation σ such that the empty word is in the interpretation of every $a_{1,j}$, we have $\widehat{\sigma}(e_1) \subseteq \widehat{\sigma}(e_2)$. This means that we need to compare 1-free expressions under the assumption that certain variables contain the empty word.

This is the intuitions behind what we call weak graphs. Weak graphs are pairs of a graph and a set of test variables. They are equipped with a preorder relation $\blacktriangleleft$, which relates $\langle G, A\rangle$ and $\langle H, B\rangle$ if $B \subseteq A$ and there is a map φ from H to G such that every edge labelled outside of A is preserved, but edges labelled with tests in A are either preserved or contracted. Such a map is shown in Figure 3.

We then have theorems similar to those for identity free Kleene lattices and series parallel graphs, in the sense that for every pair of terms u, v over the syntax $\langle \cdot, \cap, 1, \_^\smile\rangle$, if we denote by $\mathcal{WG}(u), \mathcal{WG}(v)$ their associated weak graphs, $u \leq v$ is valid if and only if

$\mathcal{WG}(u) \blacktriangleleft \mathcal{WG}(v)$. Furthermore, if we associate to every expression e in $\mathcal{E}\langle X\rangle$ a downwards closed set of weak graphs ${}^{\blacktriangleleft}[\![e]\!]$, the equation $e = f$ is valid if and only if ${}^{\blacktriangleleft}[\![e]\!] = {}^{\blacktriangleleft}[\![f]\!]$.

2.2 Weak terms

We define the following two sets of terms over the alphabet X:
Ground terms: $u, v \in GT\langle X\rangle ::= 1 \mid a \mid a^{\smile} \mid u \cdot v \mid u \cap v$.
Simple ground terms: $u, v \in GT^{-}\langle X\rangle ::= a \mid a^{\smile} \mid u \cdot v \mid u \cap v$.

We call the variables of the term u, and write $\mathit{var}(u)$, the set of variables $a \in X$ such that a or $a^{\smile}$ appears in u. We call *weak terms* the elements of the set $(GT^{-}\langle X\rangle \cup \{1\}) \times \mathcal{P}(X)$, that is simple ground terms or 1 indexed with a set of test variables. The set of weak terms is written $WT\langle X\rangle$. This set is equipped with two products, denoted by $\bullet$ and $\|$, defined by:

$$1_A \bullet 1_B := 1_{A\cup B} \qquad 1_A \bullet u_B = u_A \bullet 1_B := u_{A\cup B} \qquad u_A \bullet v_B := (u \cdot v)_{A\cup B}$$

$$1_A \parallel 1_B := 1_{A\cup B} \qquad 1_A \parallel u_B = u_A \parallel 1_B := 1_{A\cup B\cup \mathit{var}(u)} \qquad u_A \parallel v_B := (u \cap v)_{A\cup B}.$$

Given an interpretation $\sigma : X \to \mathcal{P}(\Sigma^{\star})$, the interpretation $\tilde{\sigma}(u_A)$ of the weak term u_A is either $\emptyset$ if $\exists a \in A : \varepsilon \notin \sigma(a)$, or $\widehat{\sigma}(u)$ otherwise. We define a translation τ from ground terms to weak terms:

$$\tau(u \cdot v) := \tau(u) \bullet \tau(v) \qquad \tau(u \cap v) := \tau(u) \parallel \tau(v) \qquad \forall u \in \{1\} \cup \dot{X},\ \tau(u) := u_{\emptyset}.$$

This translation is faithful, in the sense that the following holds:

▶ **Lemma 1.** $\forall u \in GT\langle X\rangle, \forall \sigma : X \to \mathcal{P}(\Sigma^{\star}),\ \widehat{\sigma}(u) = \tilde{\sigma} \circ \tau(u)$.

Proof (Sketch). The proof relies on the fact that for every pair of weak terms x, y we have:

$$\tilde{\sigma}(x \bullet y) = \tilde{\sigma}(x) \cdot \tilde{\sigma}(y) \qquad \tilde{\sigma}(x \parallel y) = \tilde{\sigma}(x) \cap \tilde{\sigma}(y).$$

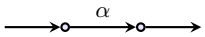
We then conclude by a simple induction on u. For concision, the full proof is omitted here. ◀

We can also define a converse translation $\kappa : WT\langle X\rangle \to GT\langle X\rangle$ which associate to a weak term $u_{\{a_1,\dots,a_n\}}$ the ground term $(1 \cap a_1 \cap \cdots \cap a_n) \cdot u$. It is immediate to check that for every term $x \in WT\langle X\rangle$ and every interpretation σ we have $\tilde{\sigma}(x) = \widehat{\sigma} \circ \kappa(x)$.

2.3 Weak graphs

A *graph* G in our setting is a tuple $\langle V_G, E_G, i_G, o_G\rangle$, where V_G is a finite set of vertices, $E_G \subseteq V_G \times \dot{X} \times V_G$ is a set of labelled and directed edges, and $i_G, o_G \in V_G$ are two vertices, called the input and output of the graph. *Term graphs* must further be series parallel[23], i_G must be the unique source vertex (*i.e.* with no incoming edge), and o_G the unique sink vertex (*i.e.* with no outgoing edge). We let G, H range over graphs. Term graphs can be sequentially composed, by identifying the output of the first graph with the input of the second one, or composed in parallel, by identifying the inputs of both graphs and identifying theirs outputs. These two compositions are respectively denoted by ; and |. The set $\ell(G)$ of labels of a graph G is defined as the set of letters $a \in X$ such that there is an edge in E_G labelled with either a or $a^{\smile}$.

The graph of a simple ground term u, written $\mathcal{G}(u)$, is a term graph defined inductively:

$\mathcal{G}(\alpha) :=$ 

$\qquad \mathcal{G}(u \cdot v) := \mathcal{G}(u) ; \mathcal{G}(v) \qquad \mathcal{G}(u \cap v) := \mathcal{G}(u) \mid \mathcal{G}(v).$

We define the graph $\not\Vdash$ as $\longrightarrow\!\!\circ\!\!\longrightarrow$. Notice that it is not a term graph, as it is not series parallel. We call *weak graph* a pair whose left part is either a term graph or $\not\Vdash$, and whose right part is a set of test variables. We denote the weak graph $\langle G, A\rangle$ by G_A. For every weak term x we associate a weak graph $\mathcal{WG}(x)$ as one would expect:

$$\mathcal{WG}(1_A) := \not\Vdash_A \qquad\qquad \mathcal{WG}(u_A) := \mathcal{G}(u)_A .$$

The weak graph G_A is smaller than H_B, written $G_A \blacktriangleleft H_B$, if $B \subseteq A$ and there exists a function $\varphi : V_H \to V_G$ such that $\varphi(i_H) = i_G$, $\varphi(o_H) = o_G$, and for every edge $\langle x, \alpha, y\rangle$ in E_H, either $\langle \varphi(x), \alpha, \varphi(y)\rangle \in E_G$ or $\alpha \in \dot{A}$ and $\varphi(x) = \varphi(y)$. The relation $\blacktriangleleft$ is a preorder. We will show in the next section that for any two ground terms u and v, the following holds:

$$\mathrm{Lang} \models u \leq v \Leftrightarrow \mathcal{WG}(\tau(u)) \blacktriangleleft \mathcal{WG}(\tau(v)) .$$

The first important lemma is the following. It generalises [1, Lemma 2.5] by including 1 and $\_^\smile$, thus moving from series parallel graphs to weak graphs.

▶ **Lemma 2.** *$\forall u \in WT\langle X\rangle$, there exists a word w_u and an interpretation σ_u such that for every $v \in WT\langle X\rangle$, $w_u \in \tilde{\sigma}_u(v) \Leftrightarrow \mathcal{WG}(u) \blacktriangleleft \mathcal{WG}(v)$.*

Proof. Let $\mathcal{WG}(u) = \langle\langle V_u, E_u, i_u, o_u\rangle, A\rangle$. Let $\mu : V_u \to \{1, \dots, |V_u|\}$ be a bijective map such that $\langle x, \alpha, y\rangle \in E_u \Rightarrow \mu(x) < \mu(y)$[1]. In particular, $\mu(i_u) = 1$ and $\mu(o_u) = |V_u|$. Let $n = 2 \times (|V_u| - 1)$, and Σ_u an alphabet composed of n distinct letters $x_1, \dots, x_n$.

We define $w_u = x_1 x_2 \dots x_n$, and $f : V_u \to \{0, \dots, n\}$ such that $f(x) = 2(\mu(x) - 1)$. Notice that $f(i_u) = 0$ and $f(o_u) = n$. We now define σ_u:

$$\sigma_u(a) := \begin{cases} \left\{w_u^f[x,y] \mid \langle x, a, y\rangle \in E_u\right\} \cup \left\{w_u^f[x,y]^\smile \;\middle|\; \langle x, a^\smile, y\rangle \in E_u\right\} \cup \{\varepsilon\} & \text{if } a \in A \\ \left\{w_u^f[x,y] \mid \langle x, a, y\rangle \in E_u\right\} \cup \left\{w_u^f[x,y]^\smile \;\middle|\; \langle x, a^\smile, y\rangle \in E_u\right\} & \text{if } a \notin A \end{cases}$$

Notice that for every $\alpha \in \dot{X}$, $\varepsilon \in \widehat{\sigma}_u(\alpha) \Leftrightarrow \alpha \in \dot{A}$, and that if $x \neq y$ then $w_u^f[x,y] \in \widehat{\sigma}_u(\alpha)$ if and only if $\langle x, \alpha, y\rangle \in E_u$.

Let $v = t_B \in WT\langle X\rangle$. First, suppose that $\exists a \in B \setminus A$. We know that $\varepsilon \notin \sigma_u(a)$, meaning that $\tilde{\sigma}_u(v) = \emptyset$. We also know by definition of $\blacktriangleleft$ that $\mathcal{WG}(u) \not\blacktriangleleft \mathcal{WG}(v)$. Thus the equivalence holds, as both sides are false. In the following, we thus assume that $B \subseteq A$.

If $t = 1$, then $\tilde{\sigma}_u(v) = \{\varepsilon\}$. This means that $w_u \in \tilde{\sigma}_u(v)$ if and only if $w_u = \varepsilon$. By definition, this is equivalent to $n = 0$, which is again equivalent to $|V_u| = 1$ thus to $u = 1_A$. We conclude this case by noticing that the only graph G such that $G_A \blacktriangleleft \not\Vdash_B$ is $\not\Vdash$ itself.

The other case is when t is a simple ground term. Then an induction much like in the proof of [1, Lemma 2.5] allows to conclude. We omit this part of the proof here. ◀

The other important lemma is a generalisation of [1, Lemma 2.3]. It will allow us to factor any interpretation of u through the weak graph $\mathcal{WG}(u)$.

▶ **Lemma 3.** *For every simple ground term u, every interpretation $\sigma : X \to \mathcal{P}(\Sigma^\star)$, and every word $w \in \Sigma^\star$ of length n:*

$$w \in \widehat{\sigma}(u) \Leftrightarrow \exists \varphi : V_u \to \{0, \dots, n\} : \begin{cases} \varphi(i_u) = 0 \wedge \varphi(o_u) = n \\ \langle x, \alpha, y\rangle \in E_u \Rightarrow w^\varphi[x,y] \in \widehat{\sigma}(\alpha) . \end{cases}$$

It can be proved by a simple induction on u; for concision, we omit this proof.

[1] Remember that both term graphs and $\not\Vdash$ are directed acyclic graphs.

2.4 Freeness results

We can now establish our first freeness result:

▶ **Theorem 4.** $\forall x, y \in GT\langle X\rangle$, $\mathcal{WG}(\tau(x)) \blacktriangleleft \mathcal{WG}(\tau(y)) \Leftrightarrow \mathrm{Lang} \models x \leq y$.

Proof. The statement of the theorem is equivalent to the following, thanks in part to Lemma 1: $\forall x, y \in WT\langle X\rangle, \mathcal{WG}(x) \blacktriangleleft \mathcal{WG}(y) \Leftrightarrow \forall \Sigma, \forall \sigma : X \to \mathcal{P}(\Sigma^\star), \tilde{\sigma}(x) \subseteq \tilde{\sigma}(y)$. We let $x = u_A$ and $y = v_B$, and proceed to prove both implications.

Suppose $\mathcal{WG}(x) \blacktriangleleft \mathcal{WG}(y)$, let σ be an interpretation, and w a word of length n. The case of 1 being trivial, we consider here the case where both u and v are simple ground terms. Assume $w \in \tilde{\sigma}(x)$, then we need to prove that $w \in \tilde{\sigma}(y)$. First notice that because $\tilde{\sigma}(x) \neq \emptyset$ it must be the case that $\forall a \in A, \varepsilon \in \sigma(a)$. By Lemma 3, we have a function $\varphi : V_u \to \{0, \ldots, n\}$ such that $\varphi(i_u) = 0$, $\varphi(o_u) = n$, and $\langle x, \alpha, y\rangle \in E_u \Rightarrow w^\varphi[x, y] \in \widehat{\sigma}(\alpha)$. By definition of $\blacktriangleleft$, we also have a function $\psi : V_v \to V_u$ such that $\psi(i_v) = i_u$, $\psi(o_v) = o_u$, and for every edge $\langle x, \alpha, y\rangle$ in E_v, either $\langle \psi(x), \alpha, \psi(y)\rangle \in E_u$ or $\alpha \in \dot{A}$ and $\psi(x) = \psi(y)$. We define $\Phi = \varphi \circ \psi$. Now we may check that $\Phi(i_v) = \varphi(i_u) = 0$; $\Phi(o_v) = \varphi(o_v) = n$; and if $\langle x, \alpha, y\rangle \in E_v$, then either

- $\langle \psi(x), \alpha, \psi(y)\rangle \in E_u$, which means $w^\Phi[x, y] = w^\varphi[\psi(x), \psi(y)] \in \widehat{\sigma}(\alpha)$;
- or $\alpha \in \dot{A}$ and $\psi(x) = \psi(y)$, which entails $w^\Phi[x, y] = \varepsilon \in \widehat{\sigma}(\alpha)$.

Using Lemma 3 again, we get that $w \in \widehat{\sigma}(v)$. Because $B \subseteq A$, we also have that $\tilde{\sigma}(y) = \widehat{\sigma}(v)$. Hence $\tilde{\sigma}(x) \subseteq \tilde{\sigma}(y)$.

For the converse, we now assume that $\mathcal{WG}(x) \not\blacktriangleleft \mathcal{WG}(y)$. Using Lemma 2, we know that $w_x \in \tilde{\sigma}_x(x)$ and that $w_x \notin \tilde{\sigma}_x(y)$. This proves that $\tilde{\sigma}_x(x) \not\subseteq \tilde{\sigma}_x(y)$. ◀

We define the set of weak terms $[\![e]\!]$ of an expression e by structural induction:

$$[\![0]\!] := \emptyset \qquad [\![1]\!] := \{1_\emptyset\} \qquad [\![\alpha]\!] := \{\alpha_\emptyset\} \qquad [\![e + f]\!] := [\![e]\!] \cup [\![f]\!]$$

$$[\![e \cdot f]\!] := \{u \bullet v \mid u \in [\![e]\!] \wedge v \in [\![f]\!]\} \qquad [\![e \cap f]\!] := \{u \parallel v \mid u \in [\![e]\!] \wedge v \in [\![f]\!]\}$$

$$[\![e^\star]\!] := \{u_1 \bullet \cdots \bullet u_n \mid n \geqslant 0 \wedge \forall 0 \leqslant i \leqslant n, u_i \in [\![e]\!]\}$$

The downward closure ${}^{\blacktriangleleft}S$ of a set of weak terms S is the set of weak terms x such that there exists a weak term $y \in S$ satisfying $\mathcal{WG}(x) \blacktriangleleft \mathcal{WG}(y)$. The function ${}^{\blacktriangleleft}$ is a closure operator. The set of downward closed sets of weak terms is the free representation of reversible Kleene lattices:

▶ **Theorem 5.** $\forall e, f \in \mathcal{E}\langle X\rangle$: ${}^{\blacktriangleleft}[\![e]\!] \subseteq {}^{\blacktriangleleft}[\![f]\!] \Leftrightarrow \mathrm{Lang} \models e \leq f$.

Proof. We use the fact that for every interpretation σ,

$$\widehat{\sigma}(e) = \bigcup_{u \in [\![e]\!]} \tilde{\sigma}(u) = \bigcup_{u \in {}^{\blacktriangleleft}[\![e]\!]} \tilde{\sigma}(u).$$

This can be proved using [1, Lemma 2.1], and Lemmas 1 and 2 and Theorem 4.

Suppose ${}^{\blacktriangleleft}[\![e]\!] \subseteq {}^{\blacktriangleleft}[\![f]\!]$, and let σ be an interpretation.

$$\widehat{\sigma}(e) = \bigcup_{u \in {}^{\blacktriangleleft}[\![e]\!]} \tilde{\sigma}(u) \subseteq \bigcup_{u \in {}^{\blacktriangleleft}[\![f]\!]} \tilde{\sigma}(u) = \widehat{\sigma}(f).$$

For the converse, suppose ${}^{\blacktriangleleft}[\![e]\!] \not\subseteq {}^{\blacktriangleleft}[\![f]\!]$. Because ${}^{\blacktriangleleft}$ is a closure operator, this means $[\![e]\!] \not\subseteq {}^{\blacktriangleleft}[\![f]\!]$. Let $u \in [\![e]\!] \setminus {}^{\blacktriangleleft}[\![f]\!]$. By Lemma 2, we have $w_u \in \tilde{\sigma}_u(u) \subseteq \widehat{\sigma}_u(e)$, but because $u \notin {}^{\blacktriangleleft}[\![f]\!]$, for every $v \in [\![f]\!]$, we have $\mathcal{WG}(u) \not\blacktriangleleft \mathcal{WG}(v)$ thus $w_u \notin \tilde{\sigma}_u(v)$. Hence w_u is not in the set $\bigcup_{v \in [\![f]\!]} \tilde{\sigma}_u(v) = \widehat{\sigma}_u(f)$. ◀

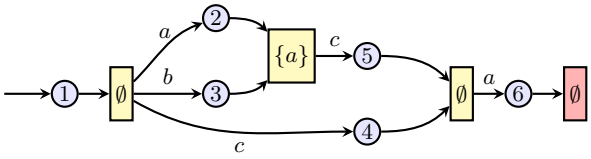


Figure 4 Weak Petri automaton.

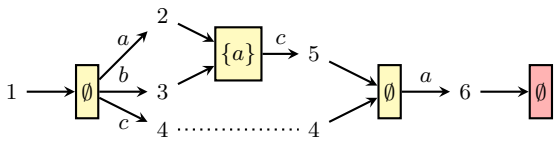
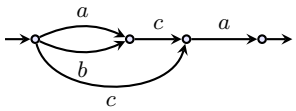


Figure 5 A run R in the automaton of Figure 4.



$A_R = \{a\}\,.$

Figure 6 Trace of R.

3 Decidability and complexity

To decide the equational theory of identity-free Kleene lattices, we used Petri automata[6]. This was a new style of automaton, which was designed to recognise sets of series parallel graphs. We modify this model slightly to recognise weak graphs, provide a construction to build automata out of expressions, and an algorithm to decide language containment (up-to closure by ◀) for these automata. This algorithm itself is inspired by the simulation algorithm for simple Petri automata. We conclude this section by showing that the problem is complete of the class ExpSpace.

3.1 Weak Petri automata

A weak Petri automaton is a Petri automaton [6, 4] whose transitions are labelled with sets of letters[2]. Formally, an *automaton* $\mathcal{A}$ over the finite alphabet X is a triple $\langle P, T, \iota\rangle$ where P is a finite set of *places*, $\iota \in P$ is the *initial place*, and $T \subseteq \mathcal{P}(P) \times \mathcal{P}(X) \times \mathcal{P}\left(\dot{X} \times P\right)$ is a set of *transitions*. Each transition $t \in T$ is composed of three parts: its *input* ${}^{\triangleright}t \subseteq P$, its *set of tests* $\widehat{t} \subseteq X$, and its *output* $t^{\triangleright} \subseteq \dot{X} \times P$. It will also be useful to write $\pi_2\left(t^{\triangleright}\right)$ for the *set of output places* of t, *i.e.* $\left\{p \in P \;\middle|\; \exists \alpha \in \dot{X} : \langle \alpha, p\rangle \in t^{\triangleright}\right\}$. The transition t is called *final* if $t^{\triangleright} = \emptyset$, and *initial* if ${}^{\triangleright}t = \{\iota\}$.

We will add a few constraints on this definition along the way, but we need more definitions to state them. An example of such an automaton is depicted in Figure 4. The graphical representation used here draws round vertices for places and rectangular vertices for transitions, with the incoming and outgoing arcs to and from the transition corresponding respectively to the inputs and outputs of said transition. The set of tests of a transition is written inside the rectangle. The initial place is denoted by an unmarked incoming arc.

Runs and reachable states

We define the operational semantics of weak Petri automata. Let us fix for the remainder of this section an automaton $\mathcal{A} = \langle P, T, \iota\rangle$. A *state* of this automaton is a set of places. In a given state $S \in \mathcal{P}(P)$, a transition t is *enabled* if ${}^{\triangleright}t \subseteq S$. In this case, we may

[2] In the following, we use the definitions from [4]. They differ slightly from those from [6], despite being overall equivalent.

fire t, leading to a new state $S' = (S \setminus {}^\triangleright t) \cup \pi_2(t^\triangleright)$. This will be denoted in the following by $S \xrightarrow{t}_{\mathcal{A}} S'$. We extend this notation to sequences of transitions in the natural way: if $S_0 \xrightarrow{t_1}_{\mathcal{A}} S_1$ and $S_1 \xrightarrow{t_2;\ldots;t_n}_{\mathcal{A}} S_n$ then we write $S_0 \xrightarrow{t_1;t_2;\ldots;t_n}_{\mathcal{A}} S_n$. In this case we say that $\langle S_0, t_1; t_2; \ldots; t_n, S_n\rangle$ is a *valid run*, or simply run, from S_0 to S_n. If $S_0 = \{\iota\}$ then the run is *initial* and if S_n is empty then it is *final*. A run which is both initial and final is called *accepting*. An accepting run of the automaton from Figure 4 is depicted in Figure 5. A state S is reachable in $\mathcal{A}$ if there is an initial run leading to S.

We may now state the first two constraints we impose on automata: if S is reachable in $\mathcal{A}$ and $S \xrightarrow{t}_{\mathcal{A}} S'$, then $(S \setminus {}^\triangleright t) \cap \pi_2(t^\triangleright) = \emptyset$, and for each transition $t \in T$, and every triple $\langle p, \alpha, \beta\rangle \in P \times \dot{X} \times \dot{X}$, we have: $\{\langle \alpha, p\rangle, \langle \beta, p\rangle\} \subseteq t^\triangleright \Rightarrow \alpha = \beta$. These constraints correspond to the classic Petri net property of safety, also called one-boundedness.

▶ Remark. These constraints are decidable: the set of transitions is finite, and because reachable states are subsets of a fixed finite set, there are only finitely many. Thus checking whether an automaton satisfies these two requirements only entails a finite number of tests.

We introduce some attributes of a run R: its *input* I_R, its *output* O_R, its *excess* E_R, its *tests* A_R, and its *internal labels* Λ_R. Let $R = S_0 \xrightarrow{t_1}_{\mathcal{A}} S_1 \ldots \xrightarrow{t_n}_{\mathcal{A}} S_n$ be a valid run in some automaton $\mathcal{A}$. I_R is the set of tokens (places) in S_0 which are consumed during the run; E_R is the rest of the tokens from S_0, those which are not moved; Λ_R is the set of labels appearing in some $t_i^\triangleright$ such that the associated token is consumed later on; A_R is the union of the sets of tests of R's transitions; and O_R is the set of outputs which are not consumed in the remainder of the run. Formally:

$$I_R := \{p \in S_0 \mid \exists i : p \in {}^\triangleright t_i\} \qquad O_R := \left\{\langle \alpha, p\rangle \;\middle|\; \exists i : \langle \alpha, p\rangle \in t_i^\triangleright \wedge \left(\forall j > i, p \notin {}^\triangleright t_j\right)\right\}$$

$$A_R := \bigcup_i \widehat{t_i} \qquad E_R := S_0 \setminus I_R \qquad \Lambda_R := \left\{\alpha \;\middle|\; \exists p, \exists i < j : \langle \alpha, p\rangle \in t_i^\triangleright \wedge p \in {}^\triangleright t_j\right\}.$$

In the example run of Figure 5, we have $I_R = \{1\}$, $E_R = \emptyset$, $O_R = \emptyset$, $A_R = \{a\}$, and $\Lambda_R = \{a, b, c\}$.

Traces

The trace language of an automaton can be obtained by extracting from every accepting run a weak graph, called its *trace*. Consider an accepting run $\langle \{\iota\}, t_0; \ldots; t_n, \emptyset\rangle$. The graph of its trace is constructed by creating a vertex k for each transition t_k of the run. We add an edge $\langle k, a, l\rangle$ whenever there is some place q such that $\langle a, q\rangle \in t_k^\triangleright$, and t_l is the first transition after t_k in the run with q among its inputs. The set of tests of the trace is A_R. The trace of the run in Figure 5 is presented in Figure 6. The definition we give below is a generalisation for arbitrary valid runs, which coincides with the informal presentation we just gave on accepting runs.

Let $R = \langle S, t_0; \ldots; t_n, S'\rangle$ be a run in $\mathcal{A}$. For every k and $p \in \pi_2(t_k^\triangleright) \setminus S'$, we define

$$\nu(k, p) = \min\{l \mid l > k \text{ and } p \in {}^\triangleright t_l\}.$$

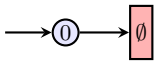
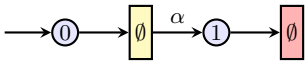
The *trace* of R, denoted by $\mathcal{G}(R)$, is the pair $\langle G_R, A_R\rangle$, where G_R has vertices $V_R = \{0, \ldots, n\} \cup S'$ and edges defined by:

$$E_R = \{\langle k, a, l\rangle \mid \langle a, p\rangle \in t_k^\triangleright \text{ and } (l = p \wedge p \in S') \vee (l = \nu(k, p))\}.$$

The *language* $\mathcal{L}(\mathcal{A})$ *of an automaton* $\mathcal{A}$ is the set of traces of accepting runs of $\mathcal{A}$. In the following, we will only consider automata such that if $\langle G, A\rangle \in \mathcal{L}(\mathcal{A})$, then either G is either isomorphic to $\nVdash$ or is a term graph: that is, if G_A is a weak graph.

3.2 From expressions to automata

In this section, we show how to build inductively from an expression e an automaton $\mathcal{A}_e$ whose language is $\mathcal{L}(\mathcal{A}_e) = \mathcal{WG}([\![e]\!])$, following [4]. For $0, 1$ and atoms, we give a graphical description of the automata:

$\mathcal{A}_0 :=$  $\mathcal{A}_1 :=$  $\mathcal{A}_\alpha :=$ 

For the inductive cases, let $\mathcal{A}_e = \langle P_e, T_e, \iota_e \rangle$ and $\mathcal{A}_f = \langle P_f, T_f, \iota_f \rangle$, and suppose $P_e \cap P_f = \emptyset$.

Intuitively, the automaton for $e + f$ is the union of $\mathcal{A}_e$ and $\mathcal{A}_f$, where we copy the initial transitions of $\mathcal{A}_f$ so that they start from ι_e instead of ι_f. Formally:

$$\mathcal{A}_{e+f} = \langle P_e \cup P_f, T_e \cup T_f \cup T, \iota_e \rangle, \text{ where } T = \left\{ \left\langle \{\iota_e\}, \widehat{t}, t^{\triangleright} \right\rangle \;\middle|\; \left\langle \{\iota_f\}, \widehat{t}, t^{\triangleright} \right\rangle \in T_f \right\}.$$

For the product, we want an automaton $\mathcal{A}_{e \cdot f}$ such that $\mathcal{L}(\mathcal{A}_{e \cdot f}) = \mathcal{L}(\mathcal{A}_e) \bullet \mathcal{L}(\mathcal{A}_f)$. This property is satisfied by the automaton $\langle P_e \cup P_f, T_e^+ \cup T_f \cup T, \iota_e \rangle$ where T_e^+ is the set of non-final transitions in T_e, and $T = \{ \langle {}^{\triangleright}t, A \cup B, t^{\triangleright} \rangle \mid \langle {}^{\triangleright}t, A, \emptyset \rangle \in T_e \wedge \langle \{\iota_f\}, B, t^{\triangleright} \rangle \in T_f \}$.

Instead of defining directly an automaton for $e^\star$, we give an automaton for the non-zero iteration e^+, and then define $\mathcal{A}_{e^\star}$ to be $\mathcal{A}_{1 \cup e^+}$. Using the last two constructs, the automaton $\mathcal{A}_{e^+}$ is easy to define: $\mathcal{A}_{e^+} = \langle P_e, T_e \cup \{ \langle {}^{\triangleright}t, A \cup B, t^{\triangleright} \rangle \mid \langle {}^{\triangleright}t, A, \emptyset \rangle \in T_e \wedge \langle \{\iota_e\}, B, t^{\triangleright} \rangle \in T_e \}, \iota_e \rangle$.

Finally, we then define $\mathcal{A}_{e \cap f}$ to be the automaton $\langle P_e \cup P_f \cup \{\iota\}, T_1 \cup T_2 \cup T_3 \cup T_4, \iota \rangle$, where ι is a fresh place, and:

- T_1 is the set of non-initial, non-final transitions of T_e and T_f;
- T_2 is the set of triples $\left\langle \{\iota\}, \widehat{t_1} \cup \widehat{t_2}, t_1^{\triangleright} \cup t_2^{\triangleright} \right\rangle$ such that t_1 (resp. t_2) is initial but not final in T_e (resp. T_f);
- T_3 is the set of triples $\left\langle {}^{\triangleright}t_1 \cup {}^{\triangleright}t_2, \widehat{t_1} \cup \widehat{t_2}, \emptyset \right\rangle$ such that t_1 (resp. t_2) is final but not initial in T_e (resp. T_f);
- T_4 is the set of triples $\langle \{\iota\}, A, \emptyset \rangle$ such that $1_A \in [\![e \cap f]\!]$.

This definition is effective, as the set of $A \subseteq X$ such that $1_A \in [\![e]\!]$ can be computed in space $O\left(|e| \times 2^{|X|}\right)$. Using the proofs for Petri automata as a guideline, it is a simple exercise to check that the correctness of the construction, that is $\mathcal{L}(\mathcal{A}_e) = \mathcal{WG}([\![e]\!])$.

3.3 Comparing automata

The algorithm to compare weak Petri automata relies on the notion of simulation. Similarly to many finite transition systems, the language of an automaton $\mathcal{A}$ is included in that of the automaton $\mathcal{B}$ if $\mathcal{B}$ can simulate $\mathcal{A}$.

▶ **Definition 6** (Simulation). Let $\mathcal{A}_1 = \langle P_1, T_1, \iota_1 \rangle$ and $\mathcal{A}_2 = \langle P_2, T_2, \iota_2 \rangle$ be two automata, we say that $\mathcal{A}_2$ can simulate $\mathcal{A}_1$ if there exists a function $\preccurlyeq$ associating to every subset of X a set of triples from $\mathcal{P}(P_1) \times \mathcal{P}(X) \times \mathcal{P}(P_2 \rightharpoonup P_1)$, such that: *(We denote the fact that the triple $\langle S, B, E \rangle$ is contained in the image by $\preccurlyeq$ of the set A by $S \preccurlyeq_A^B E$.)*

(correspondence) if $S \preccurlyeq_A^B E$ and $\eta \in E$ then $range(\eta) \subseteq S$;

(initialisation) $\{\iota_1\} \preccurlyeq_A^\emptyset \{[\iota_2 \mapsto \iota_1]\}$;

(totality) if $\emptyset \preccurlyeq_A^A E$ then $\exists \eta \in E : dom(\eta) = \emptyset$;

(progress) if $S \preccurlyeq_A^B E$ and $S \xrightarrow{t}_{\mathcal{A}_1} S'$, then $S' \preccurlyeq_A^{B \cup \widehat{t}} E'$, where E' is the set of all η' such that there is a map η in E, and a run R in $\mathcal{A}_2$ from $dom(\eta)$ to $dom(\eta')$ s.t.:

$$I_R = \{p \mid \eta(p) \in {}^{\triangleright}t\} \qquad \Lambda_R \cup A_R \subseteq \dot{A} \qquad \forall \langle \alpha, p \rangle \in O_R,\ \langle \alpha, \eta'(p) \rangle \in t^{\triangleright}$$

$$\forall p \in E_R,\ \eta(p) = \eta'(p).$$

▶ **Lemma 7.** *$\mathcal{L}(\mathcal{A}_1) \subseteq {}^{\blacktriangleleft}\mathcal{L}(\mathcal{A}_2)$ if and only if $\mathcal{A}_2$ can simulate $\mathcal{A}_1$.*

Proof. We start by showing the right to left direction: suppose that $\mathcal{A}_2$ can simulate $\mathcal{A}_1$, as witnessed by the function $\preccurlyeq$, and consider an accepting run $R = \langle S_0, t_1; \dots; t_n; S_n\rangle$ in $\mathcal{A}_1$:

$$S_0 = \{\iota_1\} \qquad \forall 1 \leqslant i \leqslant n,\ S_{i-1} \xrightarrow{t_i}_{\mathcal{A}_1} S_i \qquad S_n = \emptyset.$$

We write $B_i = \bigcup_{j<i} \widehat{t_j}$. Using the relations $\preccurlyeq^{B_i}_{A_R}$, we can find a sequence of E_i (for $0 \leqslant i \leqslant n$) such that $S_i \preccurlyeq^{B_i}_{A_R} E_i$, $E_0 = \{[\iota_2 \mapsto \iota_1]\}$, and there is some $\eta_n \in E_n$ which has an empty domain. Backtracking from this η_n using the progress condition allows us to find a sequence of maps $(\eta_i)_{0\leqslant i\leqslant n}$, with domains $(T_i)_{0\leqslant i\leqslant n}$ such that there are valid runs R_i in $\mathcal{A}_2$ from T_{i-1} to T_i, and satisfying:

$$T_0 = \{\iota_2\} \qquad T_n = \emptyset \qquad I_{R_i} = \{p \mid \eta_{i-1}(p) \in {}^{\triangleright}t_i\} \qquad \Lambda_{R_i} \cup A_{R_i} \subseteq \dot{A}_R$$

$$\forall \langle \alpha, p\rangle \in O_{R_i},\ \langle \alpha, \eta_i(p)\rangle \in t_i^{\triangleright} \qquad \forall p \in E_{R_i},\ \eta_{i-1}(p) = \eta_i(p).$$

We now build the run R' by concatenating the R_is. We obtain an accepting run in $\mathcal{A}_2$, whose set of tests is $\bigcup_i A_{R_i} \subseteq A_R$. To any transition t'_j in R', the function φ associates the index i of the run R_i from which this transition was extracted. The function φ witnesses $\mathcal{G}(R) \blacktriangleleft \mathcal{G}(R')$.

For the converse direction, we prove an intermediary result. Let $R = \langle S_0, t_1; \dots; t_n, S_n\rangle$ be an accepting run in $\mathcal{A}_1$ and R' be an accepting run from $\mathcal{A}_2$ such that $\mathcal{G}(R) \blacktriangleleft \mathcal{G}(R')$, with φ as the witnessing function. Notice that if the transition t'_i is a cause of t'_{i+1} in R' (*i.e.* they cannot be exchanged without changing the trace), either $\varphi(i) = \varphi(i+1)$, or $t_{\varphi(i)}$ is a cause of $t_{\varphi(i+1)}$ in R, thus $\varphi(i) < \varphi(i+1)$. This means that we may permute transitions in R' without changing the trace, to obtain a run R'' such that $i < j \Rightarrow \varphi(i) \leqslant \varphi(j)$.

Now, the sets of transitions sharing the same value $\varphi(i)$ are contiguous, meaning that R'' can be split as the sequence of sub-runs $R_1; \dots; R_n$, such that φ maps every transitions in R_i to i. (It may be the case that some of these runs are empty.) As $\mathcal{G}(R) \blacktriangleleft \mathcal{G}(R'')$, we know that $A_{R''} \subseteq A_R$, which means that $\forall i, A_{R_i} \subseteq A_R$. Inside the run R_i, we know that the internal edges of the graph of R_i are labelled with letters from A_R, as both their extremities are mapped to i. This means $\Lambda_{R_i} \subseteq \dot{A}_R$.

We know define the η_i. First, we set $\eta_0(\iota_2) = \iota_1$, and $\forall p \in E_{R_i}, \eta_{i-1}(p) = \eta_i(p)$. If on the other hand $\langle \alpha, p\rangle \in O_{R_i}$, let k be the index in R'' corresponding to the output state of R_i, and $j = \nu_{R''}(p, k)$. As j cannot be in R_i, we know $\varphi(j) > \varphi(k)$. Thus, in the graph of R there is an edge $\langle \varphi(k), \alpha, \varphi(j)\rangle$. By definition of the graph of a run, there must be a pair $\langle \alpha, q\rangle \in t^{\triangleright}_{\varphi(k)}$ such that $\nu_R(q, \varphi(k)) = \varphi(j)$. Then this q is a suitable choice for $\eta_i(p)$.

It is then a simple matter of unfolding the definitions to check that:

$$I_{R_i} = \{p \mid \eta_{i-1}(p) \in {}^{\triangleright}t_i\} \qquad \forall \langle \alpha, p\rangle \in O_{R_i},\ \langle \alpha, \eta_i(p)\rangle \in t_i^{\triangleright} \qquad \forall p \in E_{R_i},\ \eta_{i-1}(p) = \eta_i(p).$$

This means that whenever we have R and R' accepting runs from respectively $\mathcal{A}_1$ and $\mathcal{A}_2$ s.t. $\mathcal{G}(R) \blacktriangleleft \mathcal{G}(R')$, we can find a sequence of η_i satisfying all four conditions of a simulation. Thus, if $\mathcal{L}(\mathcal{A}_1) \subseteq {}^{\blacktriangleleft}\mathcal{L}(\mathcal{A}_2)$, for every reachable state S of $\mathcal{A}_1$, we set $\preccurlyeq^B_A$ to relate S to the set of all maps η such that there is an index i, an accepting run R in $\mathcal{A}_1$, and an accepting run R' of $\mathcal{A}_2$ satisfying (1) $A_R = A$, (2) $\bigcup_{j<i} \widehat{t_j} = B$, (3) $S = S_i$, (4) $\mathcal{G}(R) \blacktriangleleft \mathcal{G}(R')$ and (5) the construction we just provided produces $\eta_i = \eta$. ◀

3.4 Complexity

▶ **Corollary 8.** *The theory of reversible Kleene lattices is* ExpSpace*-complete.*

Proof. The equational theory of identity-free Kleene lattices being already ExpSpace-complete [8, Proposition 10.2], we know the problem at hand to be ExpSpace-hard.

Let $e, f \in \mathcal{E}\langle X \rangle$, we ask whether Lang $\models e \leq f$. By Theorem 5, this reduces to testing if ${}^{\blacktriangleleft}[\![e]\!] \subseteq {}^{\blacktriangleleft}[\![f]\!]$, which is equivalent to $[\![e]\!] \subseteq {}^{\blacktriangleleft}[\![f]\!]$ by the properties of the closure operator. Using the construction in Section 3.2, this is amounts to checking if $\mathcal{L}(\mathcal{A}_e) \subseteq {}^{\blacktriangleleft}\mathcal{L}(\mathcal{A}_f)$. This later question can be decided by looking for a simulation function, thanks to Lemma 7.

We now inspect the space complexity of this method. Let n, m, x be respectively the size of e, the size of f and the size of the alphabet. By analysing each step in Section 3.2, we get that the number of places of $\mathcal{A}_e$ is less than $2 \times n$ (similarly for $\mathcal{A}_f$). The number of transitions is harder to work out from the construction, but because $T \subseteq \mathcal{P}(P) \times \mathcal{P}(X) \times \mathcal{P}\left(\dot{X} \times P\right)$, we know it is bounded by $2^{2n+x+2x\times 2n}$. Using Savitch's theorem [22], we only need to show that there is a non-deterministic semi-algorithm to refute the existence of a simulation, which uses only exponential space in n, m and x. Here is such a procedure:

1. choose $A \subseteq X$;
2. start with $S = \{\iota_1\}$, $B = \emptyset$ and $E = \{[\iota_2 \mapsto \iota_1]\}$;
3. if $\langle S, B\rangle = \langle \emptyset, A\rangle$ and E does not contain a map η whose domain is empty return `False`;
4. choose $t \in T_1$ such that ${}^{\triangleright}t \subseteq \pi_1(S)$, fire t from S, and update B as $\widehat{t} \cup B$;
5. update E according to the progress condition in Definition 6;
6. go to step 3.

All of these computations can be performed using exponential space. For instance, S, being a pair of a set of places in $\mathcal{A}_e$ and a set of letters, can be stored in space $2n\log(2n) \times x\log(x)$, and E only needs space $(2n+1)^{2m} \times 2m\log(2n+1)$. ◀

4 Conclusion

We showed that the free representation of reversible Kleene lattices consists of downward closed sets of weak terms, or equivalently of downward closed sets of weak graphs. By considering a suitable variation of Petri automata, and producing an algorithm to decide language containment of these automata, we showed that testing the validity of equations in reversible Kleene lattices is an ExpSpace-complete problem.

The results we obtained here could be naturally extended in a number of ways.

- We would like to add to our model some features of programming languages which have been studied independently, among which tests [16], and nominal structures [13, 18, 17, 7].
- Although Kleene algebra is known not to be finitely axiomatisable [20], several authors have proposed semi-axiomatisations [15, 9, 21]. A complete axiomatisation of Kleene algebra with converse, relative to an axiomatisation of KA, is also known [10]. As far as we know, no axiomatisation of reversible Kleene lattices exists. We believe the free representation we defined in Section 2 could help establishing such an axiomatisation.
- Since we provide here an algorithm, it would be interesting to implement it. Such a procedure could fit in very well in a proof assistant such as Coq.

Although the weak Petri automata introduced in this paper were just a means to an end, we are wondering whether this might be an interesting model of computation in itself. We are confident that we could reuse to technology of boxes introduced in [8, 4] to get a Kleene theorem for these automata. Their semantics could also be reformulated with transitions labelled with weights chosen from a finite lattice (instead of sets of letters from the alphabet).

References

1. Hajnal Andréka, Szabolcs Mikulás, and István Németi. The equational theory of Kleene lattices. *TCS*, 412(52):7099–7108, 2011. doi:10.1016/j.tcs.2011.09.024.
2. Hajnal Andréka and Dmitry A. Bredikhin. The equational theory of union-free algebras of relations. *Alg. Univ.*, 33(4):516–532, 1995. doi:10.1007/BF01225472.
3. Stephen L. Bloom, Zoltán Ésik, and Gheorghe Stefanescu. Notes on equational theories of relations. *Alg. Univ.*, 33(1):98–126, 1995. doi:10.1007/BF01190768.
4. Paul Brunet. *Algebras of Relations: From algorithms to formal proofs.* PhD thesis, Université de Lyon, 2016. URL: https://tel.archives-ouvertes.fr/tel-01455083v1.
5. Paul Brunet. Reversible Kleene lattices. extended abstract, 2017. URL: https://hal.archives-ouvertes.fr/hal-01474911.
6. Paul Brunet and Damien Pous. Petri Automata for Kleene Allegories. In *Proc. LICS*, pages 68–79, July 2015. doi:10.1109/LICS.2015.17.
7. Paul Brunet and Damien Pous. A formal exploration of Nominal Kleene Algebra. In *Proc. MFCS*, 2016. doi:10.4230/LIPIcs.MFCS.2016.22.
8. Paul Brunet and Damien Pous. Petri automata. *Logical Methods in Computer Science*, 2017. submitted. arXiv:1702.01804.
9. John H. Conway. *Regular algebra and finite machines.* Chapman and Hall Mathematics Series, 1971.
10. Zoltán Ésik and Laszlo Bernátsky. Equational properties of Kleene algebras of relations with conversion. *TCS*, 137(2):237–251, 1995. doi:10.1016/0304-3975(94)00041-G.
11. Peter J. Freyd and Andre Scedrov. *Categories, Allegories.* NH, 1990.
12. Martin Fürer. The complexity of the inequivalence problem for regular expressions with intersection. In *Proc. ICALP*, pages 234–245, 1980. doi:10.1007/3-540-10003-2_74.
13. Murdoch J. Gabbay and Vincenzo Ciancia. Freshness and name-restriction in sets of traces with names. In *Proc. FoSSaCS*, pages 365–380. Springer, 2011. doi:10.1007/978-3-642-19805-2_25.
14. Stephen C. Kleene. *Representation of Events in Nerve Nets and Finite Automata.* Memorandum. Rand Corporation, 1951.
15. Dexter Kozen. A completeness theorem for Kleene Algebras and the algebra of regular events. In *Proc. LICS*, pages 214–225. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151646.
16. Dexter Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, 1997. doi:10.1145/256167.256195.
17. Dexter Kozen, Konstantinos Mamouras, Daniela Petrisan, and Alexandra Silva. Nominal Kleene coalgebra. In *Proc. ICALP*, pages 286–298. Springer, 2015. doi:10.1007/978-3-662-47666-6_23.
18. Dexter Kozen, Konstantinos Mamouras, and Alexandra Silva. Completeness and incompleteness in nominal Kleene algebra. In *Proc. RAMiCS*, pages 51–66, 2015. doi:10.1007/978-3-319-24704-5_4.
19. Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proc. SWAT*, pages 125–129, 1972. doi:10.1109/SWAT.1972.29.
20. Volodimir N. Redko. On defining relations for the algebra of regular events. *Ukrainskii Matematicheskii Zhurnal*, pages 120–126, 1964.
21. Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966. doi:10.1145/321312.321326.
22. Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

23 Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. In *Proc. STOC*, STOC '79, pages 1–12. ACM, 1979. doi:10.1145/800135.804393.

Lossy Kernels for Hitting Subgraphs*

Eduard Eiben[1], Danny Hermelin[2], and M. S. Ramanujan[3]

1 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
eiben@ac.tuwien.ac.at
2 **Industrial Engineering and Management, Ben Gurion University, Be'er Scheva, Israel**
hermelin@bgu.ac.il
3 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
ramanujan@ac.tuwien.ac.at

Abstract

In this paper, we study the Connected $\mathcal{H}$-hitting Set and Dominating Set problems from the perspective of *approximate* kernelization, a framework recently introduced by Lokshtanov et al. [STOC 2017]. For the Connected $\mathcal{H}$-hitting set problem, we obtain an α-approximate kernel for every $\alpha > 1$ and complement it with a lower bound for the natural weighted version. We then perform a refined analysis of the tradeoff between the approximation factor and kernel size for the Dominating Set problem on d-degenerate graphs, and provide an interpolation of approximate kernels between the known d^2-approximate kernel of constant size and 1-approximate kernel of size $k^{\mathcal{O}(d^2)}$.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, G.2.1 Combinatorics

Keywords and phrases parameterized algorithms, lossy kernelization, graph theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.67

1 Introduction

Polynomial time preprocessing is one of the widely used methods to tackle NP-hardness in practice and the area of *kernelization* has been extremely successful in laying down a mathematical framework for the design and rigorous analysis of preprocessing algorithms for *decision problems*. We refer the reader to the survey articles by Kratsch [17] or Lokshtanov et al. [18] for recent developments, or the textbooks [6, 11] for an introduction to the field. The central notion in kernelization is that of a *kernel*, which is a preprocessing algorithm that runs in polynomial time and transforms a 'large' instance of a decision problem into a significantly smaller, but equivalent instance.

Unfortunately, the existing notion of kernels, having been built around decision problems, does not combine well with approximation algorithms and heuristics. In particular, in order for kernels to be useful, one is required to solve the preprocessed instance exactly. However, this may not always be possible and the existing theory of kernelization says nothing about being able to infer useful information from a good *approximate* solution for the preprocessed instance. Lokshtanov et al. [19] attempted to address this limitation by introducing the notion of *α-approximate kernels*. Informally speaking, an α-approximate kernel is a polynomial time algorithm that given an instance (I, k) outputs an instance (I', k') such that $|I'| + k' \leq g(k)$

* This work was partially supported by the Austrian Science Fund (FWF, projects P26696 and W1255-N23).

for some computable function g and any c-approximate solution to the instance (I', k') can be turned in polynomial time into a $(c \cdot \alpha)$-approximate solution to the original instance (I, k). The function g is ideally polynomially bounded, in which case we call this algorithm, an α-approximate polynomial kernel. We refer the reader to the section on Preliminaries for a formal definition of the terms involved.

In their work, Lokshtanov et al. considered several problems which are known to not admit a polynomial kernel and showed that they do have an α-approximate polynomial kernel for *every* fixed $\alpha > 1$. They also proposed a machinery for proving lower bounds and managed to rule out even α-approximate kernels for some basic problems such as Longest Path and Set Cover under standard complexity theoretic hypotheses.

One of the fundamental classes of problems known to exclude polynomial kernels is the class of 'subgraph hitting' problems *with a connectivity constraint*. It is well-known that placing connectivity constraints on certain subgraph hitting problems can have a dramatic effect on their amenability to preprocessing. A case in point is the classic Vertex Cover problem. This problem is known to admit a kernel with $\mathcal{O}(k)$ vertices [6]. However, the Connected Vertex Cover problem is among the earliest problems to be shown to exclude a polynomial kernel [10] and Lokshtanov et al. [19] showed that this problem admits an α-approximate polynomial kernel for every $\alpha > 1$. Their result motivates the need to obtain a finer understanding of the role played by connectivity constraints in relation to preprocessing for other subgraph hitting problems. Therefore, a systematic study of the approximate kernelization of subgraph hitting problems with connectivity constraints is a natural strategy towards achieving this goal.

Our Contributions. In this paper, we study the Connected $\mathcal{H}$-Hitting Set (Conn-$\mathcal{H}$-HS) problem where the input is a graph G and an integer k and the objective is to check whether there is a set S of at most k vertices such that $G[S]$ is connected and $G - S$ has no vertex-induced subgraph isomorphic to a graph in the fixed finite family of graphs, $\mathcal{H}$. It is easy to see that this problem generalizes Connected Vertex Cover (set $\mathcal{H} = \{K_2\}$) and hence it is unlikely to have a polynomial kernel. As a result, we consider the approximate kernelization complexity of the *optimization* version of this problem and provide two results; one positive and one negative. Our positive result generalizes the approximate kernel given by Lokshtanov et al. for the Connected Vertex Cover problem and shows that Conn-$\mathcal{H}$-HS also admits an α-approximate polynomial kernel for *every* constant $\alpha > 1$ and fixed $\mathcal{H}$.

▶ **Theorem 1.** *For every fixed $\epsilon > 0$, there is a $(1 + \epsilon)$-approximate polynomial kernel for* Connected $\mathcal{H}$-hitting set.

Our negative result shows that this ability to obtain approximate kernels vanishes in the presence of weights, even when the domain of the weight function is highly restricted. To be precise, we study the Weighted Conn-$\mathcal{H}$-HS problem where the input also includes a weight function on the vertices and the objective now is to minimize the total *weight* of a connected set of vertices hitting all vertex-induced subgraphs of G isomorphic to a graph in $\mathcal{H}$. We show that unless $\mathsf{NP} \subseteq \mathsf{coNP/Poly}$, this problem has no α-approximate polynomial kernel for *any* constant α even when the domain of the weight function is restricted to $\{0, 1\}$. The formal statement of this theorem requires certain terms which are as yet undefined and we refer the reader to Section 3.2 for the statement.

In the second part of the paper, we initiate the fine-grained analysis of the accuracy-size tradeoff encountered when designing approximate kernels for the Dominating Set problem on the class of d-degenerate graphs. The Dominating Set problem is one of the most

fundamental problems in algorithmic graph theory. In the decision version of this problem, the input is a graph G, an integer k and the objective is to decide whether G has a set S of at most k vertices which contains at least one neighbor of every vertex in $V(G) \setminus S$. It is well-known that Dominating Set is W[2]-hard [6], implying that it cannot have *any* kernel under standard complexity theoretic hypotheses. This fact motivated the study of preprocessing for this basic problem on restricted graph classes, leading to a long and rich literature [1, 2, 3, 7, 8, 12, 15, 20].

One of the more general graph classes on which Dominating Set is known to admit a polynomial kernel, is the class of d-degenerate graphs, which contains several well-studied graph classes such as planar graphs, graphs of bounded treewidth, graphs of bounded arboricity, and graphs excluding a fixed (topological) minor. Alon and Gutner [2] initiated the study of the parameterized complexity of Dominating Set on d-degenerate graphs and Philip et al. [20] obtained a kernel of size $k^{\mathcal{O}(d^2)}$, which was the first polynomial kernel for this problem on d-degenerate graphs. In fact, it follows from their proofs that this kernel is in fact a 1-approximate kernel. At the other extreme, it follows from [16] that there is a d^2-approximate kernel of *constant size*. These two results motivate the natural question: *What is the precise tradeoff between accuracy and kernel size for* Dominating Set *on d-degenerate graphs?* We give a sequence of approximate kernels for this problem which lie 'between' the two extremes and provide an interesting interpolation of kernels.

▶ **Theorem 2.** Dominating Set *on d-degenerate graphs has a $\lceil \frac{d}{\rho} \rceil$-approximate kernel of size $k^{\mathcal{O}(d\rho)}$, for any fixed integer $\rho \in \{1, \ldots, d\}$.*

All approximate kernels obtained thus far (including that in the first part of our paper) are focussed on problems which are known to not admit polynomial kernels. Our work on Dominating Set on d-degenerate graphs thus initiates and motivates the fine-grained study of approximate kernelization even for problems which have polynomial kernels. Therefore, we believe that our result opens up a new line of investigation in the topic of approximate kernelization.

2 Preliminaries

The notion of kernels is based on *parameterized problems* from the area of Parameterized Complexity [6, 11]. Inputs of a parameterized problem are of the form (I, k) where I is a bitstring encoding an instance and k is an integer called the *parameter*, and every input is either a yes instance or a no instance. A *kernel* is a polynomial time algorithm that given an instance (I, k) of a parameterized problem outputs an instance (I', k') of the same problem such that $|I'| + k' \leq g(k)$ for some computable function g and (I, k) is a yes instance if and only if (I', k') is a yes instance. We now recall the main definitions from [19] regarding *parameterized optimization problems and approximate kernels.*

▶ **Definition 3** ([19])**.** A parameterized optimization (minimization or maximization) problem Π is a computable function $\Pi : \Sigma^* \times \mathbb{N} \times \Sigma^* \to \mathbb{R} \cup \{\pm\infty\}$.

The *instances* of a parameterized optimization problem Π are pairs $(I, k) \in \Sigma^* \times \mathbb{N}$, and a *solution* to (I, k) is simply a string $s \in \Sigma^*$, such that $|s| \leq |I| + k$. The *value* of the solution s is $\Pi(I, k, s)$. Since the problems we deal with in this paper are all minimization problems, we state some of the definitions only in terms of minimization problems when the definition for maximization problems is analogous.

▶ **Definition 4** ([19]). For a parameterized minimization problem Π, the *optimum value* of an instance $(I,k) \in \Sigma^* \times \mathbb{N}$ is $OPT_\Pi(I,k) = \min_{\substack{s\in\Sigma^* \\ |s|\leq |I|+k}} \Pi(I,k,s)$.

▶ **Definition 5** ([19]). Let $\alpha \geq 1$ be a real number and Π be a parameterized minimization problem. An *α-approximate polynomial time preprocessing algorithm* $\mathcal{A}$ for Π is a pair of polynomial time algorithms. The first one is called the *reduction algorithm*, and computes a map $\mathcal{R}_\mathcal{A} : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$. Given as input an instance (I,k) of Π the reduction algorithm outputs another instance $(I',k') = \mathcal{R}_\mathcal{A}(I,k)$.

The second algorithm is called the *solution lifting algorithm*. This algorithm takes as input an instance $(I,k) \in \Sigma^* \times \mathbb{N}$ of Π, the output instance (I',k') of the reduction algorithm, and a solution s' to the instance (I',k'). The solution lifting algorithm works in time polynomial in $|I|$,k,$|I'|$,k' and s', and outputs a solution s to (I,k) such that the following holds.

$$\frac{\Pi(I,k,s)}{OPT(I,k)} \leq \alpha \cdot \frac{\Pi(I',k',s')}{OPT(I',k')}.$$

The *size* of a polynomial time preprocessing algorithm $\mathcal{A}$ is a function $\text{size}_\mathcal{A} : \mathbb{N} \to \mathbb{N}$ defined as follows.

$$\text{size}_\mathcal{A}(k) = \sup\{|I'| + k' : (I',k') = \mathcal{R}_\mathcal{A}(I,k), I \in \Sigma^*\}.$$

▶ **Definition 6** ([19]). An *α-approximate kernelization* (or *α-approximate kernel*) for a parameterized optimization problem Π, and real $\alpha \geq 1$, is an α-approximate polynomial time preprocessing algorithm $\mathcal{A}$ for Π such that $\text{size}_\mathcal{A}$ is upper bounded by a computable function $g : \mathbb{N} \to \mathbb{N}$. We say that $\mathcal{A}$ is an α-approximate polynomial kernelization if g is a polynomial function.

▶ **Definition 7** ([19]). Let $\alpha \geq 1$ be a real number, and Π be a parameterized minimization problem. An α-approximate polynomial time preprocessing algorithm for Π is said to be *strict* if, for every instance (I,k), reduced instance $(I',k') = \mathcal{R}_\mathcal{A}(I,k)$ and solution s' to (I',k'), the solution s to (I,k) output by the solution lifting algorithm when given s' as input satisfies the following.

$$\frac{\Pi(I,k,s)}{OPT(I,k)} \leq \max\left\{\frac{\Pi(I',k',s')}{OPT(I',k')}, \alpha\right\}$$

The notion of *strictness* in the above direction allows one to 'chain' multiple α-approximate preprocessing algorithms to obtain a single α-approximate preprocessing algorithm.

▶ **Definition 8.** A *reduction rule* is simply the reduction algorithm of a polynomial time preprocessing algorithm. The reduction rule *applies* if the output instance of the reduction algorithm is not the same as the input instance.

▶ **Definition 9** ([19]). A reduction rule is *α-safe for* Π if it is the reduction algorithm of a strict α-approximate polynomial time preprocessing algorithm for Π.

The notion of 1-safe reduction rules is crucial because numerous reduction rules used in the domain of (standard) kernelization can be either easily, or with very little effort, proved to be 1-safe. Therefore, when designing α-approximate kernels, it is a useful strategy to examine existing 1-safe reduction rules and either utilize them directly or design a 'relaxed' version which one can then prove to be α-safe for some $\alpha > 1$.

▶ **Definition 10** ([19]). Let $\alpha \geq 1$ be a real number. Let Π and Π' be two parameterized minimization problems. An *α-approximate polynomial parameter transformation* (α-appt for short) $\mathcal{A}$ from Π to Π' is a pair of polynomial time algorithms, called reduction algorithm $\mathcal{R}_{\mathcal{A}}$ and solution lifting algorithm. Given as input an instance (I,k) of Π the reduction algorithm outputs an instance (I',k') of Π' such that $k' = k^{\mathcal{O}(1)}$. The solution lifting algorithm takes as input an instance (I,k) of Π, the output instance $(I',k') = \mathcal{R}_{\mathcal{A}}(I,k)$ of Π', and a solution s' to the instance I' and outputs a solution s to (I,k) such that

$$\frac{\Pi(I,k,s)}{OPT_{\Pi}(I,k)} \leq \alpha \cdot \frac{\Pi'(I',k',s')}{OPT_{\Pi'}(I',k')}.$$

▶ **Definition 11** ([19]). Let $\alpha \geq 1$ be a real number. Let Π and Π' be two parameterized minimization problems. An *α-approximate compression* from Π to Π' is an α-appt $\mathcal{A}$ from Π to Π' such that $\text{size}_{\mathcal{A}}(k) = \sup\{|I'| + k' : (I',k') = \mathcal{R}_{\mathcal{A}}(I,k), I \in \Sigma^*\}$, is upper bounded by a computable function $g : \mathbb{N} \to \mathbb{N}$, where $\mathcal{R}_{\mathcal{A}}$ is the reduction algorithm in $\mathcal{A}$. We say that $\mathcal{A}$ is an α-approximate polynomial compression if g is a polynomial function. When we simply say that Π has an α-approximate compression, we mean that there is an α-approximate compression from Π to some language Π'.

▶ **Observation 12.** *Let $\alpha, \beta \geq 1$ be fixed real numbers and let Π and Π' be parameterized minimization problems. If there is an α-appt from Π to Π' and Π' has a β-approximate polynomial compression, then Π has an $(\alpha \cdot \beta)$-approximate polynomial compression.*

Steiner trees, set systems and degenerate graphs. Given a graph G, a set $R \subseteq V(G)$ whose vertices are called *terminals*, and a weight function $w : E(G) \to \mathbb{N}$, a *Steiner tree* is a subtree T of G such that $R \subseteq V(T)$, and the *cost* of a tree T is defined as $w(T) = \sum_{e \in E(T)} w(e)$. A *$k$-component* is a tree with at most k leaves which all coincide with a subset of terminals. A *k-restricted Steiner tree* $\mathcal{T}$ is a collection of k-components, such that the union of these components is a Steiner tree T. The cost of $\mathcal{T}$ is the sum of the costs of all the k-components in $\mathcal{T}$.

Let $\mathcal{F}$ be a set system over the universe $\mathcal{U}$. An element $e \in \mathcal{U}$ is said to *hit* a set $A \in \mathcal{F}$, if $e \in A$. Moreover, we say that a set $S \subseteq \mathcal{U}$ hits a set $A \in \mathcal{F}$ if $S \cap A \neq \emptyset$. A set $S \subseteq \mathcal{U}$ is a *hitting set* of $\mathcal{F}$ if it hits every set in $\mathcal{F}$. We say that $\mathcal{S} \subseteq \mathcal{F}$ is a *set cover* of $(\mathcal{F},\mathcal{U})$, if $\bigcup_{S \in \mathcal{S}} S = \mathcal{U}$. Note that if there exists a set cover of $(\mathcal{F},\mathcal{U})$, then there is one of size at most $|\mathcal{U}|$, as every vertex is in at least one set. Moreover, one can find a set cover of size at most $|\mathcal{U}|$ by greedily adding, in every step, a new set that covers an as yet uncovered vertex.

Let $\mathcal{H}$ be a fixed finite set of finite graphs. We denote by $d_{\mathcal{H}}$ the size of a largest graph in $\mathcal{H}$. For a graph G, we denote by $\mathcal{F}_{\mathcal{H}}(G)$ the following set system defined over the universe $V(G)$: $\mathcal{F}_{\mathcal{H}}(G) = \{S \subseteq V(G) \mid G[S] \text{ is isomorphic to some } H \in \mathcal{H}\}$. That is, $\mathcal{F}_{\mathcal{H}}(G)$ comprises precisely those subsets of $V(G)$ which induce a subgraph of G isomorphic to a graph in $\mathcal{H}$. Observe that for a fixed family $\mathcal{H}$ and a graph G, the set $\mathcal{F}_{\mathcal{H}}(G)$ can be computed in time $\mathcal{O}(|V(G)|^{d_{\mathcal{H}}})$. A set $S \subseteq V(G)$ is called a $\mathcal{H}$-hitting set of G if it is a hitting set for the family $\mathcal{F}_{\mathcal{H}}(G)$. A graph G is said to be d-degenerate if every subgraph of G has a vertex of degree at most d. It is well-known that a d-degenerate graph G has less than $d|V(G)|$ edges.

3 Approximate kernels for Connected $\mathcal{H}$-hitting set

In this section, we present our positive and negative results on the Connected $\mathcal{H}$-hitting set problem and its weighted variant. In what follows, we fix a family $\mathcal{H}$ and assume without loss of generality that for any distinct pair of graphs in $\mathcal{H}$, neither is a subgraph of the other.

3.1 The α-approximate kernel for Connected $\mathcal{H}$-hitting set

We begin by defining the parameterized optimization version of Conn-$\mathcal{H}$-HS. This is a minimization problem, where the optimization function is $\mathrm{CHS} : \Sigma^* \times \mathbb{N} \times \Sigma^* \to \mathbb{R} \cup \{\pm\infty\}$ and defined as follows.

$$\mathrm{CHS}(G,k,S) = \begin{cases} \infty & \text{if } S \text{ is not a connected } \mathcal{H}\text{-hitting set in } G, \\ \min\{|S|, k+1\} & \text{otherwise.} \end{cases}$$

For the rest of Section 3.1, we define $\mathrm{OPT}(G,k) = \min_{S \subseteq V(G)} \mathrm{CHS}(G,k,S)$.

We split our approximate kernel for Conn-$\mathcal{H}$-HS into two steps. Let d denote the size of the largest graph in $\mathcal{H}$. First we compute a set D of size at most $k^{\mathcal{O}(d)}$ such that if a set S of size at most k is an $\mathcal{H}$-hitting set in $G[D]$, then S is an $\mathcal{H}$-hitting set in G. In the second step, we closely follow the idea of the approximate kernel for Steiner Tree (see [5, 19]) to bound the number of vertices outside D that we need to preserve to guarantee a 'good' connected set that hits all subgraphs in $G[D]$ isomorphic to a graph in $\mathcal{H}$.

Our starting point is the known kernel of size $k^{\mathcal{O}(d)}$ for the (not necessarily connected) $\mathcal{H}$-Hitting Set problem [6]. This kernel uses the *sunflower reduction rule* which is based on the classic sunflower lemma [14]. The sunflower reduction rule will also be a critical part of our approximate kernel and we begin by recalling the formal definition of sunflowers. Let $\mathcal{F}$ be a set system over the universe $\mathcal{U}$ and let $s \in \mathbb{N}$. A set $A_1, \ldots, A_s \in \mathcal{F}$ is called an *s-sunflower* if for every i, j such that $1 \leq i < j \leq s$, $A_i \cap A_j = \cap_{r=1}^{s} A_r$.

▶ **Proposition 13** ([14]). *Let $\mathcal{F}$ be a set system and d the size of a largest set in $\mathcal{F}$. If $|\mathcal{F}| > d!(k+1)^d$ $\mathcal{F}$, then $\mathcal{F}$ contains a $(k+2)$-sunflower. Moreover, it can be found in time polynomial in $|\mathcal{F}|$, k, and d.*

We now state and prove the following lemma, which is crucial for the correctness of the sunflower reduction rule. Although the following lemma is well-known, we state it in a way that is most convenient for our application.

▶ **Lemma 14.** *Let $\mathcal{U}$ be a universe of elements, $\mathcal{F}_1 \supset \mathcal{F}_2 \supset \cdots \supset \mathcal{F}_r$ be a family of set systems over $\mathcal{U}$ and $A_1, \ldots, A_{r-1} \subseteq \mathcal{U}$ such that for every $i \in \{1, \ldots, r-1\}$ the following holds: (a) $\mathcal{F}_{i+1} = \mathcal{F}_i \setminus A_i$ and (b) A_i is contained in a $(k+2)$-sunflower in $\mathcal{F}_i$. Then, if a set $S \subseteq \mathcal{U}$ of size at most k hits all sets in $\mathcal{F}_r$, then S also hits all sets in $\mathcal{F}_1$.*

We are now ready to formally describe the construction of the set D.

▶ **Lemma 15.** *Fix $\mathcal{H}$ and let $d = d_{\mathcal{H}}$. There exists a polynomial time algorithm that takes as input a graph G and integer k and outputs a set of vertices $D \subseteq V(G)$ of size at most $d \cdot d!(k+1)^d$ such that if a set S of size at most k is an $\mathcal{H}$-hitting set in $G[D]$, then S is an $\mathcal{H}$-hitting set in G.*

Due to this lemma, once we compute the set D, the *hitting* part of the Conn-$\mathcal{H}$-HS problem is taken care of and it is the *connectivity* which results in the hardness of standard kernelization. In other words, for every connected $\mathcal{H}$-hitting set S of a graph G of size at most k it follows from Lemma 15 that $D \cap S$ is also a $\mathcal{H}$-hitting set of G and the only role of vertices in $S - D$ is to connect the set of vertices in $D \cap S$. Such a situation could be handled relatively easily in the case of Connected Vertex Cover (see [19]) since the graph induced on $V(G) \setminus D$ is by definition an *independent set*. However, since we are dealing with an arbitrary family $\mathcal{H}$, we cannot rely on any structural consequences of a graph excluding $\mathcal{H}$; only the fact that the size of the largest graph in $\mathcal{H}$ is a fixed integer. A natural

approach to providing connectivity between vertices in a graph is to construct a Steiner tree over a particular set of terminals. Unfortunately, since we do not know the set $S \cap D$ apriori, one would have to try and compute an appropriate Steiner tree for every possible subset of D of size at most $k-1$, as the set of terminals. Since this would be too expensive for us, we will try to preserve all necessary *approximate* Steiner trees. We begin by recalling the following result of Borchers and Du [4].

▶ **Proposition 16** ([4]). *For every $t \geq 1$, graph G, terminal set R, cost function $w : E(G) \to \mathbb{N}$, and Steiner tree T, there is a t-restricted Steiner tree $\mathcal{T}$ of cost at most $(1 + \frac{1}{\lfloor \log_2 t \rfloor}) \cdot w(T)$.*

It follows from the proof of Borchers and Du [4] that if for every subset of R of size at most t, one were to preserve an *optimal* Steiner tree for this subset, then it is possible to construct a t-restricted Steiner tree of R of cost at most that of the tree $\mathcal{T}$ in Proposition 16 (see also [5, 19]). This fact will be used crucially in our algorithm.

▶ **Lemma 17.** *For every fixed $\epsilon > 0$, there exists a polynomial time algorithm that takes as an input a connected graph G and k and either correctly determines that G does not contain a connected $\mathcal{H}$-hitting set of size at most k or outputs an induced subgraph G' of G of size $\mathcal{O}(k^{d \cdot 2^{\frac{1}{\epsilon}}+1})$ such that:(1) if S is a connected $\mathcal{H}$-hitting set in G', then S is a connected $\mathcal{H}$-hitting set in G and (2) $\mathrm{OPT}(G', k) \leq (1+\epsilon) \cdot \mathrm{OPT}(G, k)$.*

Proof. The algorithm first executes the algorithm of Lemma 15 to obtain a set of vertices D of size at most $d \cdot d!(k+1)^d$ such that if a set S of size at most k is an $\mathcal{H}$-hitting set in $G[D]$, then S is an $\mathcal{H}$-hitting set in G. We fix a constant t such that $\frac{1}{\lfloor \log_2 t \rfloor} \leq \epsilon$. Now for every subset R of D of size at most t we fix a cost function assigning 1 to every edge and compute an *optimal* Steiner tree T_R for the set of terminals R using, for example, the Dreyfus-Wagner Algorithm [13]. It follows from [13] that this step takes time $\mathcal{O}(3^t|E(G)||V(G)|)$, which is polynomially bounded since ϵ is a fixed constant. If $|V(T_R)| \leq k$, then we *mark* the vertices of T_R. After we have computed T_R for every subset R (of size at most t) of D, we remove all unmarked vertices from G and denote the resulting graph by G'. We now claim that G' is the desired graph. It is easy to see that $|V(G')| \leq \sum_{i=1}^{t} \binom{|D|}{i} \cdot k = \mathcal{O}(k^{d \cdot t+1})$. Moreover, every connected $\mathcal{H}$-hitting set in G' is a $\mathcal{H}$-hitting set in $G[D]$ and hence it is also a connected $\mathcal{H}$-hitting set in G.

Note that if G' contains two different connected components A, B, then a shortest path with one endpoint in $A \cap D$ and the other in $B \cap D$ must have at least $k+1$ vertices. Otherwise, we would have marked such a path and A and B would not be distinct connected components of G'. Therefore, if both $A \cap D$ and $B \cap D$ contain a subgraph isomorphic to a graph in $\mathcal{H}$, then every connected $\mathcal{H}$-hitting set of G contains at least $k+1$ vertices and we may correctly return that G does not contain a connected $\mathcal{H}$-hitting set of size at most k.

Otherwise, for at most one component C of G', the graph $G[C \cap D]$ contains an induced subgraph isomorphic to a graph in $\mathcal{H}$. Since every $\mathcal{H}$-hitting set in $G[D]$ is a $\mathcal{H}$-hitting set in G, it follows that every *connected* $\mathcal{H}$-hitting set of $G[C]$ is also a connected $\mathcal{H}$-hitting set of G. Therefore, we assume in the following that G' is connected.

It remains for us to prove that $\mathrm{OPT}(G', k) \leq (1+\epsilon) \cdot \mathrm{OPT}(G, k)$. Observe that by the definition of the function OPT, it must be the case that $OPT(G, k), OPT(G', k) \leq k+1$. This is simply because $\mathrm{CHS}(G, k, V(G))$ and $\mathrm{CHS}(G', k, V(G'))$ are both bounded by $k+1$. Now, if it is the case that $\mathrm{OPT}(G, k) = k+1$, then $\mathrm{OPT}(G', k) \leq k+1 \leq (1+\epsilon) \cdot \mathrm{OPT}(G, k)$.

Therefore, we may assume that $\mathrm{OPT}(G, k) \leq k$. Let Q be an optimal connected $\mathcal{H}$-hitting set in G of size at most k. That is, $\mathrm{CHS}(G, k, Q) = OPT(G, k) = |Q|$. We denote $Q_D = Q \cap D$ and $Q_R = Q \setminus D$. Clearly, Q_D is a $\mathcal{H}$-hitting set in $G[D]$ and by our construction

of D, it follows that Q_D is a $\mathcal{H}$-hitting set in G. Hence, if we consider the Steiner tree instance obtained by assigning every edge in G weight 1 and choosing Q_D as the set of terminals, any spanning tree T of $G[Q]$ must in fact be an optimal Steiner tree in G for the aforementioned weight function and terminal set Q_D. We invoke Proposition 16 to infer that there is a t-restricted Steiner tree $\mathcal{T}$ of cost at most $(1+\frac{1}{\lfloor \log_2 t \rfloor})\cdot(|Q|-1)$. It remains to argue that we can reconstruct such a t-restricted Steiner tree $\mathcal{T}$ for Q_D using only the vertices in G'.

Consider a t-component C in $\mathcal{T}$ and let R be the set of terminals in C. Since $R \subseteq Q_D$, it implies that G' contains an optimal Steiner tree T_R for R. Moreover, C is a Steiner tree with R as the set terminals, hence $|T_R| \leq |C|$ and we can replace C by T_R in $\mathcal{T}$. Exhaustively repeating this argument we conclude that there is a t-restricted Steiner tree $\mathcal{T}'$ with set of terminals Q_D of cost no more than $(1+\frac{1}{\lfloor \log_2 t \rfloor})\cdot(|T|-1)$, such that all k-components in $\mathcal{T}'$ use only marked vertices. Furthermore, (see paragraph following Proposition 16), the union of all t-components in $\mathcal{T}'$, denoted by $\bigcup \mathcal{T}'$, is indeed a Steiner tree. In particular, $\bigcup \mathcal{T}'$ is connected and contains all vertices in Q_D. Therefore, $\bigcup \mathcal{T}'$ is a connected $\mathcal{H}$-hitting set in G of size at most $(1+\frac{1}{\lfloor \log_2 t \rfloor})\cdot(|Q|-1)+1 \leq (1+\epsilon)|Q|$. Since $|Q|$ is by definition the same as $\mathrm{OPT}(G,k)$, the lemma follows. ◀

▶ **Theorem 1.** *For every fixed $\epsilon > 0$, there is a $(1+\epsilon)$-approximate polynomial kernel for* Connected $\mathcal{H}$-hitting set.

Proof. We begin by describing the reduction algorithm. We first invoke the algorithm of Lemma 17. If this algorithm concludes that G does not contain a connected $\mathcal{H}$-hitting set of size at most k, then we return the instance $(H,0)$, where $H \in \mathcal{H}$. Otherwise, if this algorithm returns a graph G', then the reduction algorithm returns the instance (G',k). From Lemma 17 it follows that the size of the reduced instance is $\mathcal{O}(k^{d\cdot 2^{-\frac{1}{\epsilon}}})$.

We now describe the solution lifting algorithm as follows. Let S' be the given solution for (G',k). If S' is not a connected $\mathcal{H}$-hitting set in G', then the algorithm outputs $\emptyset$. If S' is a connected $\mathcal{H}$-hitting set in G', then the algorithm outputs S', if $|S'| \leq k$ and $V(G)$ otherwise. We denote by S the output of the solution lifting algorithm.

We now prove that this reduction algorithm and the solution lifting algorithm together constitute a $(1+\epsilon)$-approximate kernel. Note that if S' is not a connected $\mathcal{H}$-hitting set of G', then $\emptyset$ is also not a connected $\mathcal{H}$-hitting set of G and $\mathrm{CHS}(G',k',S') = \mathrm{CHS}(G,k,\emptyset) = \infty$. On the other hand, if $\mathrm{OPT}(G,k) = k+1$, then it follows from Lemma 17 and the definition of the reduction algorithm that $\mathrm{OPT}(G',k') = k'+1$. Therefore,

$$\frac{\mathrm{CHS}(G,k,V(G))}{\mathrm{OPT}(G,k)} = 1 \leq (1+\epsilon)\cdot\frac{\mathrm{CHS}(G',k',S)}{\mathrm{OPT}(G',k')} = (1+\epsilon)$$

Hence, we can assume that $\mathrm{OPT}(G,k) \leq k$ and the reduction algorithm returned the instance (G',k) such that G' is as in Lemma 17. Then either $|S'| \leq k$ and $S = S'$ or $|S'| \geq k+1$ and $S = V(G)$. However, in both cases it holds that $\mathrm{CHS}(G,k,S) = \mathrm{CHS}(G',k,S')$. Moreover, from Lemma 17 it follows that $\mathrm{OPT}(G',k) \leq (1+\epsilon)\cdot\mathrm{OPT}(G,k)$, implying the theorem. ◀

3.2 The lower bound for Weighted Connected $\mathcal{H}$-hitting set

In this section, we prove that in the presence of weights, the Connected $\mathcal{H}$-hitting set problem no longer admits an α-approximate kernel for *any* constant α. The parameterized optimization version of Weighted Connected $\mathcal{H}$-hitting set is formally defined via the function W-CHS : $\Sigma^* \times \mathbb{N} \times \Sigma^* \to \mathbb{R} \cup \{\pm\infty\}$ as follows: $\text{W-CHS}((G,w),k,S) = \infty$

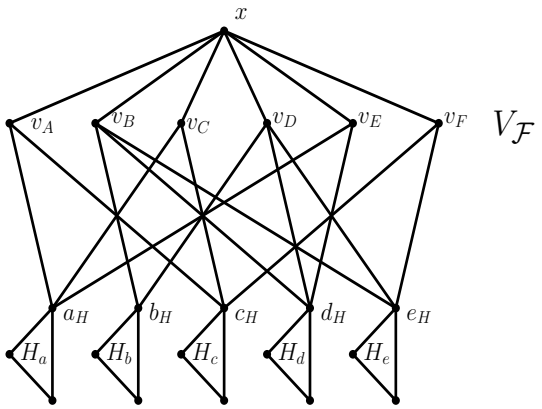


Figure 1 The graph output by our reduction algorithm starting from the SC/n instance $(\{A, B, C, D, E, F\}, \{a, b, c, d, e\})$ where the sets are defined as $A = \{a, c\}$, $B = \{b, d, e\}$, $C = \{a, c\}$, $D = \{b, d, e\}$, $E = \{a, d\}$, and $F = \{c, e\}$. Here, $\mathcal{H}$ contains only a triangle.

if S is not a connected $\mathcal{H}-$ hitting set of size at most k and W-CHS$((G, w), k, S) = w(s)$ otherwise.

We prove our lower bound by giving a polynomial time reduction from a parameterized optimization version of the classic Set Cover problem such that an α-approximate polynomial kernel for Weighted Connected $\mathcal{H}$-hitting set would imply one for Set Cover, which would contradict the lower bound in [19]. Note that since we are proving a lower bound, it is sufficient to demonstrate *one* family $\mathcal{H}$ for which Weighted Conn-$\mathcal{H}$-HS does not admit approximate kernels. However, in the interest of extracting the strongest possible consequence of our reduction, we introduce the following definition.

▶ **Definition 18.** Let $\mathcal{H}$ be a fixed finite family of finite graphs. We say that $\mathcal{H}$ is *rigid* if there is a connected graph H in $\mathcal{H}$ and a vertex $v \in V(H)$ such that no graph $H' \in \mathcal{H}$ is a subgraph of H and no graph $H' \in \mathcal{H}$ is the disjoint union of connected components each of which is isomorphic to $H - v$.

▶ **Theorem 19.** *Let $\mathcal{H}$ be a fixed rigid family of graphs. Then, there is no α-approximate polynomial compression for* Weighted Conn-$\mathcal{H}$-HS *for any constant α unless* NP $\subseteq$ coNP/Poly *even if the weight function is restricted to* $\{0, 1\}$.

Proof. We prove the theorem by giving a 1-approximate polynomial parameter transformation from SC/n to the Weighted Connected $\mathcal{H}$-hitting set problem. Recall that a polynomial parameter transformation consists of two algorithms, a reduction algorithm and a solution lifting algorithm. We describe a reduction algorithm that takes as input an instance $(\mathcal{F},\mathcal{U})$ of SC/n and outputs an instance (G, k, w) of Weighted Connected $\mathcal{H}$-hitting set such that $k = 2|\mathcal{U}| + 1$, $|G| \leq 1 + |\mathcal{F}| + d_{\mathcal{H}}|\mathcal{U}|$.

Reduction Algorithm. We construct G from $(\mathcal{F},\mathcal{U})$ as follows. The vertex set $V(G)$ is partitioned into sets $\{x\} \uplus V_{\mathcal{U}} \uplus V_{\mathcal{F}}$. Fix a graph $H \in \mathcal{H}$ which certifies the rigidity of $\mathcal{H}$. That is, there is a vertex $h^* \in V(H)$ such that no graph $H' \in \mathcal{H}$ is the disjoint union of connected components each of which is isomorphic to $H - h^*$. The set $V_{\mathcal{U}}$ induces in G, a disjoint copy H_u of H for every element $u \in \mathcal{U}$. We fix a special vertex $u_H \in H_u$ for every $u \in \mathcal{U}$. This vertex is the vertex of H_u corresponding to h^*. This is to ensure that after

deleting u_H from each H_u, we do not still have a graph from $\mathcal{H}$ contained in $G[V_\mathcal{U}]$. The set $V_\mathcal{F}$ contains a vertex v_S for every set $S \in \mathcal{F}$. Finally, x is a vertex disjoint from $V_\mathcal{U} \cup V_\mathcal{F}$. The edge set of G is defined as follows $E(G) = \{xv_S | v_S \in V_\mathcal{F}\} \cup \{v_S u_H | u \in S\} \cup_{u \in \mathcal{U}} E(H_u)$. In other words, $E(G)$ contains beside the edges for every copy of H, an edge between x and every vertex in $V_\mathcal{F}$ and then an edge between a vertex $v_S \in V_\mathcal{F}$ corresponding to the set S and the previously fixed special vertex u_H in the copy of H corresponding to an element u, if and only if $u \in S$ (see Figure 1). Finally, the weight function $w : V(G) \to \{0, 1\}$ is defined as follows. We let $w(v) = 0$ if $v \in \{x\} \cup V_\mathcal{U}$ and $w(v) = 1$ otherwise. The *weight* of a set $Q \subseteq V(G)$ is defined as $\Sigma_{q \in Q} w(q)$. This completes the description of the reduction algorithm.

Solution Lifting Algorithm. The solution lifting algorithm is straightforward. Given a solution string T for the instance (G, k, w), if T is not a connected $\mathcal{H}$-hitting set of size at most k, then we return a spurious solution string for the instance $(\mathcal{F},\mathcal{U})$. Otherwise, we return the sets in $\mathcal{F}$ which correspond to $V_\mathcal{F} \cap T$.

We are now ready to prove that this is a 1-approximate polynomial parameter transformation. Observe that in order to do so, it is sufficient to prove the following claim.

▶ **Claim 20.** *For every $p \in \mathbb{N}$ there is a set cover of $(\mathcal{F},\mathcal{U})$ of size p if and only if there is a connected $\mathcal{H}$-hitting set of G with at most k vertices and weight exactly p.*

Proof. Suppose that $\mathcal{S}$ is a set cover of $(\mathcal{F},\mathcal{U})$. We can assume without loss of generality that $|\mathcal{S}| \leq |\mathcal{U}|$. We claim that $T = \{x\} \cup \{v_S | S \in \mathcal{S}\} \cup \{u_H | u \in \mathcal{U}\}$ is a weighted connected $\mathcal{H}$-hitting set of G of weight $|\mathcal{S}|$. As all vertices in $V_\mathcal{F}$ have weight 1 and all other vertices have weight 0, the weight of T is $|\mathcal{S}|$. Moreover, all vertices in $V_\mathcal{F}$ are adjacent to x and since $\mathcal{S}$ is a set cover, every vertex u_H is adjacent to a vertex v_S for a set $S \in \mathcal{S}$ that contains u. Finally every connected component of $G - T$ is either a vertex or a graph isomorphic to $H - h^*$. Since $\mathcal{H}$ is rigid, we conclude that $G - T$ does not contain a graph in $\mathcal{H}$. Since $\mathcal{S}$ has size at most $|\mathcal{U}|$, the size of T is bounded by $2|\mathcal{U}| + 1$ which is precisely k.

In the converse direction suppose that T is a connected $\mathcal{H}$-hitting set of G of weight p. Since the only vertices with non-zero weights lie in $V_\mathcal{F}$ and they all have weight 1, we infer that $|V_\mathcal{F} \cap V(T)| = p$. We claim that $\mathcal{S} = \{S | v_S \in V(T) \cap V_\mathcal{F}\}$ is a set cover of $(\mathcal{F},\mathcal{U})$. Observe that since T is a $\mathcal{H}$-hitting set, it must be the case that for every $u \in \mathcal{U}$, T contains a vertex from H_u. Since T also contains at least one vertex of $V_\mathcal{F}$ (under the simple assumption that $|\mathcal{U}| > 1$), and only u_H is adjacent to a vertex outside H_u, it follows that $u_H \in V(T)$. Moreover, u_H is adjacent only to vertices in H_u or in $V_\mathcal{F}$. Therefore, u_H is adjacent to a vertex $v_S \in V_\mathcal{F} \cap V(T)$ for a set $S \in \mathcal{F}$. This implies that the element u is covered by the set $S \in \mathcal{S}$, completing the proof of the claim and the proof of the lemma. ◀

◀

4 Interpolating kernels for Dominating Set on d-degenerate graphs

This section is devoted to Theorem 2, i.e., the approximate kernels interpolating between two known kernels with respect to their accuracy-size tradeoff.

▶ **Proposition 21.** [20] Dominating Set *has a kernel of size $\mathcal{O}((d+2)^{2(d+2)} k^{2(d+1)^2})$ on d-degenerate graphs.*

▶ **Definition 22.** The parameterized optimization version of Dominating Set is defined via the function $\mathrm{DS} : \Sigma^* \times \mathbb{N} \times \Sigma^* \to \mathbb{R} \cup \{\pm\infty\}$ as follows:

$$\mathrm{DS}(G,k,S) = \begin{cases} \infty & \text{if } S \text{ is not a dominating set of } G, \\ \min\{|S|, k+1\} & \text{otherwise.} \end{cases}$$

For the rest of this section, we define $\mathrm{OPT}(G,k) = \min_{D \subseteq V(G)} \mathrm{DS}(G,k,D)$.

The kernelization algorithm of Philip et al. [20] can be seen to be a strict 1-approximate polynomial kernel and forms the starting point of our sequence of approximate kernels. We give here a slightly different description of this kernel (in particular of its analysis) so as to better serve our purposes. First of all, we will be working with a "colored" version of the problem where the vertices of the input graph are partitioned into two sets – the set of red vertices R and the set of blue vertices B – and the goal is to find a subset of at most k vertices of any color that dominates all red vertices. That is, a set $S \subseteq R \cup B$ with $|S| \leq k$ such that for every $v \in R$ we have $N[v] \cap S \neq \emptyset$. Clearly every instance of Dominating Set can be reduced to the colored variant by coloring all the vertices red. For presentation purposes, we will refer to the colored version as Dominating Set and instances of this problem are of the form (G,B,R,k) where B and R denote the set of blue and red vertices respectively. The functions $\mathrm{DS}(G,k,S)$ and $\mathrm{OPT}(G,k)$ are now represented as $\mathrm{DS}(G,B,R,k,S)$ and $\mathrm{OPT}(G,B,R,k)$ with the natural extended definitions. Furthermore, since edges between vertices in B are irrelevant with respect to the domination of R, we may assume without loss of generality that B is an independent set. Philip et al. [20] devised the following reduction rule and their proof of correctness of the rule also shows that it is in fact 1-safe.

Let (G,B,R,k) be the given instance of Dominating Set. For $i \in \{0,\ldots,d\}$: If there exists a set of $d+1-i$ vertices $X \subseteq R \cup B$ which have at least $k^i(d+1)$ common red neighbors $Y \subseteq R$, then remove the edges between X and Y, color all vertices in Y blue, and add $k+1$ new red vertices that are each connected to all vertices in X and no other vertex in G. The parameter remains k.

Henceforth, we assume that Reduction Rule 4 does not apply on the given instance of Dominating Set. Philip et al. [20] showed that if Reduction Rule 4 does not apply on the instance (G,B,R,k), then every vertex in G has at most $k^d(d+1)$ red neighbors, leading to the following observation.

▶ **Lemma 23.** *If Reduction Rule 4 does not apply on the instance (G,B,R,k), then either $|R| \leq k^{d+1}(d+1)$ or $\mathrm{OPT}(G,B,R,k) = k+1$.*

Due to Lemma 23, we may assume that $|R| \leq k^{d+1}(d+1)$. The following standard *twin reduction* rule can be easily seen to be 1-safe.

If $b_1, b_2 \in B$ are two non-adjacent vertices such that $N(b_1) = N(b_2)$, we remove b_1 from G.

In the following, for every $i \in \{0,\ldots,d\}$, we let B_i denote the set of blue vertices which have exactly i red neighbors and let $B_{>d}$ denote the set of blue vertices which have at least $d+1$ red neighbors. We now prove the following bound on the size of each of these sets.

▶ **Lemma 24.** *Let (G,B,R,k) be an instance of Dominating Set on which Reduction Rule 4 and Reduction Rule 4 do not apply. Then, $|B_{>d}| \leq d|R|$, and $|B_i| \leq |R|^i$ for each $i = 0,\ldots,d$.*

Observe that since we have only applied 1-safe reduction rules, the instance obtained after the exhaustive application of Reduction Rule 4 and Reduction Rule 4 is a strict 1-approximate

kernel and due to Lemma 24, the result is a strict 1-approximate kernel of size $k^{\mathcal{O}(d^2)}$. This is the kernel of Philip et al. [20] and henceforth we refer to instances of Dominating Set on which this preprocessing has been executed, as *reduced instances* and assume without loss of generality that the input has size bounded by $k^{\mathcal{O}(d^2)}$. We will now introduce a 'lossy reduction rule' to reduce the size of our kernel further at the cost of transforming it into a $\lceil \frac{d}{\rho} \rceil$-approximate kernel.

▶ **Lemma 25.** *Let $d, \rho \in \mathbb{N}$ be fixed integers such that $\rho < d$. There is an algorithm that, given a reduced instance (G, B, R, k) of* Dominating Set *runs in polynomial time and returns an instance (G', B', R', k) such that (a) $|V(G')| \leq k^{\mathcal{O}(\rho d)}$, (b) if S dominates R' in G', then S dominates R in G and (c) $\mathrm{OPT}(G', B', R', k) \leq \lceil \frac{d}{\rho} \rceil \cdot \mathrm{OPT}(G, B, R, k)$.*

Proof. Let B^* denote an auxiliary set of blue vertices which is initially empty. For each subset of ρ red vertices $R_0 \subseteq R$ we find a blue vertex $b \in B$ (if one exists) with $R_0 \subseteq N(b)$, and add it to our auxiliary set B^*. At the end of this procedure, we define the graph G' to be the subgraph of G induced by $R \cup B_0 \cup \cdots \cup B_\rho \cup B_{>d} \cup B^*$, $B' = B \cap V(G')$, and $R' = R$.

Recall that $|R| = \mathcal{O}(k^{d+1})$, and so there are $\binom{|R|}{\rho} = \mathcal{O}(k^{\rho(d+1)})$ subsets R_0. Thus, $|B^*| = \mathcal{O}(k^{\rho(d+1)})$. Moreover, $|B_0 \cup \cdots \cup B_\rho| = \mathcal{O}(k^{d\rho})$ according to Lemma 24. Therefore, $|V(G')|$ is bounded by $k^{\mathcal{O}(\rho d)}$ as required and the time required to compute G' is bounded polynomially in $|V(G)|$. We now proceed to the remaining two statements. Since $R' = R$ and G' is a subgraph of G, it follows that any set S which dominates all vertices of R' in G', also dominates all vertices of R in G. Hence, it only remains to prove the second statement.

Let S be an optimal solution for G. That is, $\mathrm{OPT}(G, B, R, k) = |S|$. We now construct a solution S' for G' as follows. We begin by setting $S' = S \cap V(G')$. Note that S' includes all vertices of $R \cap S$ since $R \subseteq V(G) \cap V(G')$. Consider now a blue vertex $b \in S \setminus V(G')$, and let $R(b)$ denote the set of red neighbors of b. Then $\rho + 1 \leq |R(b)| \leq d$ by the construction of G'. Moreover, for any subset of ρ vertices in $R(b)$, there is a vertex of B' in $V(G')$ which dominates these ρ vertices. Thus, we can replace b with at most $\lceil \frac{d}{\rho} \rceil$ vertices of B' in $V(G')$ and still dominate $R(b)$. Therefore, applying this switch for each $b \in S \setminus V(G')$, we obtain a solution S' for G' with $|S'| \leq \lceil \frac{d}{\rho} \rceil |S|$. This implies that $\mathrm{OPT}(G', B', R', k) \leq \lceil \frac{d}{\rho} \rceil \cdot \mathrm{OPT}(G, B, R, k)$, completing the proof of the lemma. ◀

From Lemma 25 it immediately follows that Dominating Set on d-degenerate graphs has $\lceil \frac{d}{\rho} \rceil$-approximate compression to the colored version of the problem. Theorem 2 then follows by gadgeteering similar to that used by Philip et al. [20]. Note that any graph that excludes K_h as a minor also excludes K_h as a topological minor (see [9] for a formal definition of minors and topological minors). Furthermore, it is known that any graph that does not contain K_h as a minor (topological minor) is d-degenerate where $d = \mathcal{O}(h^2)$ ($d = \mathcal{O}(h\sqrt{\log h})$ respectively) [2], giving us the following corollary.

▶ **Corollary 26.** *Let $\rho, h \in \mathbb{N}$. Then,* Dominating Set *on graphs excluding K_h as a minor (topological minor) has a $\mathcal{O}(\frac{h^2}{\rho})$-approximate kernel ($\mathcal{O}(\frac{h\sqrt{\log h}}{\rho})$-approximate kernel) of size $k^{\mathcal{O}(\rho h^2)}$ ($k^{\mathcal{O}(\rho h\sqrt{\log h})}$ respectively).*

5 Conclusions

Our work on the Connected $\mathcal{H}$-Hitting Set problem adds another interesting data point to the study of preprocessing for problems with connectivity constraints. We have also initiated the study of accuracy-size tradeoffs for problems which already have polynomial

kernels, via the design of a sequence of kernels capturing the gradient of the kernel-size with respect to the accuracy or approximation factor. Our results point to a few interesting questions for future research.

- Are there other connectivity-constrained problems which do not admit polynomial kernels but admit α-approximate kernels?
- Is it possible to obtain *meta-theorems* characterizing or providing at least a sufficiency condition for connectivity-constrained problems to admit α-approximate kernels?
- Is it possible to refine the interpolation (Theorem 2) by presenting a sequence of kernels between the d^2-approximate kernel of constant size and our d-approximate kernel of size $k^{\mathcal{O}(d)}$?

References

1. Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004.
2. Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009. doi:10.1007/s00453-008-9204-0.
3. Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016.
4. Al Borchers and Ding-Zhu Du. The k-steiner ratio in graphs. In Frank Thomson Leighton and Allan Borodin, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 641–649. ACM, 1995.
5. Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, 2013.
6. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
7. Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.
8. Erik D. Demaine and MohammadTaghi Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 51(3):292–302, 2008.
9. Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
10. Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and ids. *ACM Transactions on Algorithms*, 11(2):13:1–13:20, 2014.
11. Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.
12. Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 31:1–31:14, 2016.
13. S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971.
14. P. Erdös and R. Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, s1-35(1):85–90, 1960.
15. Fedor V. Fomin and Dimitrios M. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM J. Comput.*, 36:281–309, 2006.

16 Mark Jones, Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Ondrej Suchý. Parameterized complexity of directed steiner tree on sparse graphs. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 671–682, 2013.

17 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014.

18 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization–preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, pages 129–161. Springer, 2012.

19 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237. ACM, 2017.

20 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Trans. Algorithms*, 9(1):11:1–11:23, 2012.

Undecidable Problems for Probabilistic Network Programming

David M. Kahn

Cornell University, Department of Computer Science, Ithaca NY, USA
dmk254@cornell.edu

Abstract

The software-defined networking language NetKAT is able to verify many useful properties of networks automatically via a PSPACE decision procedure for program equality. However, for its probabilistic extension ProbNetKAT, no such decision procedure is known. We show that several potentially useful properties of ProbNetKAT are in fact undecidable, including emptiness of support intersection and certain kinds of distribution bounds and program comparisons. We do so by embedding the Post Correspondence Problem in ProbNetKAT via direct product expressions, and by directly embedding probabilistic finite automata.

1998 ACM Subject Classification D.2.4 Software/Program Verification, D.3.1 Formal Definitions and Theory, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.3 Formal Languages

Keywords and phrases Software-defined networking, NetKAT, ProbNetKAT, Undecidability, Probabilistic finite automata

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.68

1 Introduction

The NetKAT family of programming languages aims to simplify network programming and its verification [1, 10]. The NetKAT languages do this is by centralizing network control in a scheme known as software-defined networking, and by exploiting the algebraic structure of Kleene algebra with tests. This leaves the language NetKAT sufficiently well-behaved so as to admit a relatively fast (PSPACE-complete) decision procedure for program equivalence [5]. Several important networking problems, such as waypointing and reachability, are reducible to program equivalence in NetKAT, so this decision procedure is quite useful for automated verification of NetKAT programs.

NetKAT, however, has limitations. NetKAT is deterministic and cannot express probabilistic concepts that often arise in networking, such as randomized routing algorithms and connection failure chance [4]. The language ProbNetKAT answers this by conservatively extending NetKAT with an operator for probabilistic choice. However, the addition of this new operator further distances the language from the well-behavedness of a pure Kleene algebra with tests. This opens up many new questions concerning decidability and deductive completeness. If the benefits of the NetKAT framework are to be applied to probabilistic networking, it is important to now determine what can or cannot be decided about ProbNetKAT. As of the time of this paper, few ProbNetKAT decidability results have been put forward.

This paper introduces several properties of ProbNetKAT programs that are not decidable, alongside some contexts in which they might arise. These undecideable properties include emptiness of support intersection, support size, and certain kinds of program distribution bounds and comparisons. We prove these problems undecidable through two different

embeddings. First, the Post Correspondence Problem is embedded in ProbNetKAT via direct product expressions, which are expressions in the direct product of regular expressions. And second, probabilistic finite automata are then directly embedded in ProbNetKAT, and their associated undecidable problems translated over.

2 Definition of ProbNetKAT

To understand ProbNetKAT, it is important to understand how NetKAT treats its *packets*, the information that is passed around a network. A given packet $\pi \in Pk$ is a collection of fields of information, such that $\pi(x)$ can be considered to be the value of field x. Each of these fields may be assigned value using the $\leftarrow$ operator, or *tested* using the $=$ operator. These tests act as filters, dropping all packets that do not satisfy them. Any such test can also be complemented using the $\neg$ operator.

For bookkeeping purposes, these packets are understood to be arranged in *histories*, $\pi_1 :: \pi_2 :: ... :: \pi_n$, which are sequences of snapshots of a given packet over time, from youngest to oldest. (Such a history is often shortened notationally to $\pi_1 :: \sigma$.) The operation which takes this snapshot is called *dup*, because it *duplicates* the latest packet, the head packet π_1, and appends it to the front of the sequence. Only the head packet ever actually exists in the network in execution, but tracking the history that a given program would generate for a packet allows for answering important questions about that packet, like where it has been.

In addition to these basic operations of test and assignment, NetKAT also employs the operators and constants of Kleene algebra. Kleene algebra's addition is given with & (parallel composition), its multiplication is given with ; (sequential composition), its asterate is given with * (iteration), its annihilator element 0 is given with *drop* (drops all packets), and its identity element 1 is given with *skip* (does nothing). Together, these form a Kleene algebra on the generating set of assignments and tests. This means, for instance, that & is commutative. Interestingly, in most contexts, we interpret Kleene algebras as treating & disjunctively, but in NetKAT, & is conjunctive. And, paired with the negation operator ($\neg$), these also can form a Boolean algebra on only tests ($=$). For a Boolean algebra, *drop* act as 0, *skip* acts as 1, & acts as disjunction, and ; acts as conjunction. This means, for instance, that both ; and & are commutative on the tests.

This Kleene algebra structure paired with Boolean algebra comprises the Kleene algebra with tests, or KAT, that NetKAT is named after. This algebra is sufficient to characterize a lot of programmatic structure, allowing for manipulation of such structure into different semantically equivalent forms. For instance, letting the term b below be such a Boolean term, and letting p and q be general expressions, we can represent the programming idioms of the while loop and if-then-else clause as below. The symbols used will be defined formally shortly.

$$\text{if } b \text{ then } p \text{ else } q = b;p\&\neg b;q \qquad \text{while } b \text{ do } p = (b;p)^*;\neg b$$

The relevant syntax of NetKAT can therefore be given just by KAT expressions of the above operators on an alphabet of tests, assignments, and *dup*s, where tests are the Boolean elements.

Before we provide the formal definition of the above pieces of NetKAT, it will be useful to mention the Haskell language's monad operators, *return* (η) and *bind* ($>>=$). Letting $T(A)$ be a type predicated on the type A (like a *list* of elements from A), the monad operators are of the types $>>=: T(A) \to (A \to T(B)) \to T(B)$ and $\eta : A \to T(A)$ for some types A and B.

These operators allow the functional language of Haskell to translate functions on A into $T(A)$ in a useful way, called *lifting*. Bind and return satisfy the following axioms, corresponding to something similar to left identity, right identity, and associativity, respectively.

$$\eta(x) >\!\!>\!= f = f(x) \quad t >\!\!>\!= \eta = t \quad t >\!\!>\!= (\lambda x.f(x) >\!\!>\!= g) = (t >\!\!>\!= f) >\!\!>\!= g$$

Formally, a NetKAT program p or q is usually interpreted as a function that maps a history in $\pi :: \sigma \in H$ to a set of histories $a \in 2^H$, with the semantics defined as follows [12]. These definitions will be presented alongside some monadic counterparts to better clarify their structure and emphasize the ability to lift the operators to act on sets of histories. The relevant monad for these operations is the powerset monad.

$$\eta(\pi :: \sigma) = \{\pi :: \sigma\} \quad a >\!\!>\!= [\![p]\!] = \bigcup_{h \in a} [\![p]\!](h)$$

$$\begin{array}{lcll}
[\![skip]\!](\pi :: \sigma) & = & \{\pi :: \sigma\} & = \quad \eta(\pi :: \sigma) \\
[\![drop]\!](\pi :: \sigma) & = & \emptyset & \\
[\![x \leftarrow n]\!](\pi :: \sigma) & = & \{\pi[n/x] :: \sigma\} & = \quad \eta(\pi[n/x] :: \sigma) \\
[\![x = n]\!](\pi :: \sigma) & = & \begin{cases} \{\pi :: \sigma\} & \pi(x) = n \\ \emptyset & otherwise \end{cases} & = \quad \begin{cases} \eta(\pi :: \sigma) & \pi(x) = n \\ \emptyset & otherwise \end{cases} \\
[\![dup]\!](\pi :: \sigma) & = & \{\pi :: \pi :: \sigma\} & = \quad \eta(\pi :: \pi :: \sigma) \\
[\![\neg b]\!](\pi :: \sigma) & = & \{\pi :: \sigma\} - [\![b]\!](\pi :: \sigma) & = \quad \begin{cases} \emptyset & [\![b]\!](\pi :: \sigma) = \eta(\pi :: \sigma) \\ \eta(\pi :: \sigma) & otherwise \end{cases} \\
[\![p \& q]\!](\pi :: \sigma) & = & [\![p]\!](\pi :: \sigma) \cup [\![q]\!](\pi :: \sigma) & = \quad [\![p]\!](\pi :: \sigma) \\
 & & & \quad >\!\!>\!= \lambda s.([\![q]\!](\pi :: \sigma) >\!\!>\!= \lambda t.\eta(s) \cup \eta(t))) \\
[\![p; q]\!](\pi :: \sigma) & = & \bigcup_{h \in [\![p]\!](\pi :: \sigma)} [\![q]\!](h) & = \quad [\![p]\!](\pi :: \sigma) >\!\!>\!= [\![q]\!] \\
[\![p^*]\!](\pi :: \sigma) & = & [\![skip \& p^*; p]\!](\pi :: \sigma) & = \quad [\![skip \& p; p^*]\!](\pi :: \sigma) \\
 & = & \bigcup_{n \in \mathbb{N}} [\![p^n]\!](\pi :: \sigma) &
\end{array}$$

To get ProbNetKAT, we start by lifting these operators with bind as $\lambda a.a >\!\!>\!= [\![p]\!]$ to get a new set of operators that allow us to interpret NetKAT programs as functions from *sets* of histories to sets of histories. Then we add in the probabilistic choice operator $\oplus$, which, with a given probability r, chooses to continue to the code on its left, and otherwise the code on its right. Syntactically it may be used with the same generality as & or ;, such that all take two ProbNetKAT programs and return one. The new operator's randomness introduces distributions on the output, so a ProbNetKAT program can be interpreted as a function that maps a set of histories $a \in 2^H$ to a distribution on sets of histories μ. The semantics for the this interpretation is defined below, where many of the definitions show the same structure as in NetKAT. These semantics use the Dirac delta function δ for probability measure (which yields a distribution that puts all weight on a single point) and the product measure $\times$. The relevant monad here is the Giry monad.

$$\delta_a(A) = \begin{cases} 1 & a \in A \\ 0 & else \end{cases} \quad \eta(a) = \delta_a \quad \mu >\!\!>\!= [\![p]\!] = \lambda A. \int_{a \in 2^H} [\![p]\!](a)(A) \cdot \mu(da)$$

$$
\begin{array}{lcll}
[\![skip]\!](a) & = & \delta_a & = \eta(a) \\
[\![drop]\!](a) & = & \delta_\emptyset & = \eta(\emptyset) \\
[\![x \leftarrow n]\!](a) & = & \delta_{\{\pi[n/x]::\sigma | \pi::\sigma \in a\}} & = \eta(\{\pi[n/x] :: \sigma | \pi :: \sigma \in a\}) \\
[\![x = n]\!](a) & = & \delta_{\{\pi::\sigma \in a | \pi(x)=n\}} & = \eta(\{\pi :: \sigma | \pi :: \sigma \in a \wedge \pi(x) = n\}) \\
[\![dup]\!](a) & = & \delta_{\{\pi::\pi::\sigma \ | \pi::\sigma \in a\}} & = \eta(\{\pi :: \pi :: \sigma \ | \pi :: \sigma \in a\}) \\
[\![\neg b]\!](a) & = & \delta_{\{\pi::\sigma \in a | \pi \not\vdash b\}} & = [\![b]\!](a) >\!\!>\!\!= \lambda s.\eta(a - s) \\
[\![p \& q]\!](a) & = & ([\![p]\!](a) \times [\![q]\!](a))(\{(s,t) | s \cup t \in a\}) & = [\![p]\!](\pi :: \sigma) \\
 & & & \quad >\!\!>\!\!= \lambda s.([\![q]\!](\pi :: \sigma) \\
 & & & \quad >\!\!>\!\!= \lambda t.\eta(s \cup t))) \\
[\![p; q]\!](a) & = & [\![q]\!]([\![p]\!](a)) & = [\![p]\!](\pi :: \sigma) >\!\!>\!\!= [\![q]\!] \\
[\![p \oplus_r q]\!](a) & = & r \cdot [\![p]\!](a) + (1 - r) \cdot [\![q]\!](a) & \\
[\![p^{(0)}]\!](a) & = & [\![skip]\!](a) & \\
[\![p^{(n+1)}]\!](a) & = & [\![skip \& p; p^{(n)}]\!](a) & \\
[\![p^*]\!](a) & = & \bigsqcup_{n \in \mathbb{N}} [\![p^{(n)}]\!](a) & = [\![skip \& p; p^*]\!](a)
\end{array}
$$

This maintains many, but not all, of the typical properties we expect a Kleene algebra with tests to have. For instance, & is still commutative (thanks to the Frobenius theorem) and so are the test operations, but $[\![skip \& p^*; p]\!](\pi :: \sigma)$ is no longer an identity of p^*.

It will next be useful to define an extended form of some operators to act as short-hand for writing them down multiple times. Let p_i be a program predicated on the variable i, and let I be a finite set of i values indexed 1 through n, and let r_i be a probability predicated on i.

$$\underset{i \in I}{;} p_i = p_{i_1}; p_{i_2}; ...; p_{i_n} \qquad \underset{i \in I}{\&} p_i = p_{i_1} \& p_{i_2} \& ... \& p_{i_n}$$

$$\underset{i \in I}{\oplus_{r_i}} p_i = p_{i_1} \oplus_{r_1} (p_{i_2} \oplus_{r_2/(1-r_1)} (...(p_{i_n} \oplus_{r_n/(1 - \sum_{1 \le i < n} r_i)} drop)...))$$

This definition of sequential composition across multiple terms is only well-defined where the p_i are commutative. We will only be using it in commutative contexts like with test terms, so it is sufficient for our purposes. The sequential composition of multiple choice is designed such that each term p_i is picked with the given probability r_i. This requires that the sum of the probabilities does not exceed 1.

It is also useful to replace the assignment operator $\leftarrow$ and test operator $=$ with a complete assignment operator ! and complete test operator ? defined below, where X is the set of all fields in a packet. The complete assignment operator assigns all fields of a packet to match those of another, turning the first packet into the second. The complete test operator tests all fields of a packet against those of another, dropping the first if it does not perfectly match the second.

$$[\![\pi?]\!](a) = [\![\underset{x \in X}{;} x = \pi(x)]\!](a) \quad [\![\pi!]\!](a) = [\![\underset{x \in X}{;} x \leftarrow \pi(x)]\!](a)$$

3 A Random Loop

Some of our results rely on a program structure that loops a random number of times before exiting the loop. The following ProbNetKAT code achieves that purpose. Π will be a set of packets where every packet has a boolean field x assigned to *true*. C will stand in for the body of the loop, written to only use packets of Π. Assigning $false$ to the Π packets' field x yields a new set of packets Ψ, in one-to-one correspondance. Because the following code

results in the execution of C a number of times that is a geometric random variable with parameter r, this code will be denoted $[\![C^{G_r}]\!]$.

$$[\![C^{G_r}]\!] = [\![(\text{while } x = true \text{ do } \{C \oplus_r x \leftarrow false\}); x \leftarrow true]\!]$$

Suppose this program is given a set of histories with head packets in Π, and consider dynamically each step the code takes. First it will enter a while loop conditioned on x being $true$, so that while the packets are in Π, the loop continues. Thus, being in Ψ at the end of the while loop acts as the marker to break the loop, and the loop will never execute its body on packets from Ψ. In each iteration of the while loop, the code makes a probabilistic choice. With probability r the code runs C, which, since C is designed to run only using packets from Π, will go through C's manipulations as expected, only extending the histories with packets from Π, and thus exiting its code block with all head packets in Π. Otherwise, with probability $1 - r$, the code maps the packets into Ψ, signalling the the end of the looping. After exiting the while loop, the code ends by re-mapping all head packets back into Π.

Thus, the while loop either exits with probability $1 - r$, or it runs C on the current head packets from Π and loops again. This is the structure of running Bernoulli trials until success, so the number of times C is run is geometrically distributed with respect to r. Specifically, the probability of running C n times is given by $(1 - r) * r^n$. This accomplishes the desired goal, as a geometric distribution assigns every natural number a positive probability, so C could be run for any number of iterations.

4 Main Results

4.1 A Post-Correspondance Embedding

Direct Product Expressions (DPEs) For the first undecidable problem that will be presented, it is best to lay out an intermediate undecidable problem involving a variation on regular expressions, which will be called DPEs as shorthand for "direct product expressions". DPEs, rather than describing sets of the usual words from Σ^* (the free monoid on a generating alphabet Σ), instead describe words from direct product of n copies of Σ^* in the category of monoids, denoted $(\Sigma^*)^n$. In this direct product, multiplication is defined component-wise, such that $(a_1, a_2, ..., a_n) \cdot (b_1, b_2..., b_n) = (a_1 \cdot b_1, a_2 \cdot b_2, ..., a_n \cdot b_n)$.

Consider the minimal generating set for $(\Sigma^*)^n$, elements of which are tuples made entirely of the identity element ϵ, except for one position in which sits an element of Σ. Call this set Γ. For brevity, refer to the element of Γ containing $a \in \Sigma$ in position i as $a_{(i)}$, and in general, the tuple containing only ϵ except for $w \in \Sigma^*$ at index i as $w_{(i)}$.

$$\Gamma = \{a_{(i)} | a \in \Sigma \wedge 1 \leq i \leq n\}$$

Now, Γ^*, being the free monoid generated by Γ, can be treated in the usual manner as the alphabet for a regular expression. And because $(\Sigma^*)^n$ is also a monoid generated by Γ, there exists a unique canonical epimorphism $h : \Gamma^* \to (\Sigma^*)^n$ that acts as an identity on elements of Γ. This h is the map that performs component-wise multiplication, introducing a sort of commutativity in its image such that, while $a_{(1)} \cdot a_{(2)} \neq a_{(2)} \cdot a_{(1)}$, it is the case that $h(a_{(1)} \cdot a_{(2)}) = (a, a) = h(a_{(2)} \cdot a_{(1)})$. It is in the image of this map h that we would like to consider DPEs, though the regular expressions yielding the DPEs will be easier to write out and work with here.

Addition on DPEs continues to remain a nondeterministic choice between terms, so that the set of words matching $(a, a) + (b, b)$ is just the set containing only (a, a) and (b, b). Note

that addition is therefore not componentwise and introduces a sort of choice dependence between the choices made at different indices; (a, b) matches $(a+b, a+b)$, but not $(a, a)+(b, b)$. Finally, the asterate is defined as per usual, with w^* being the supremum of across all n of w^n, satisfying $w^* = 1 + w \cdot w^*$, for w a word in the direct product and 1 the multiplicative identity. Informally, we can then see that a DPE looks like a set of regular expressions, each on their own separate track corresponding to index, which can behave dependently on one another. Expoitation of this dependency will yield the undecidability that we seek.

Certain properties of DPEs are decidable just as easily as regular expressions. Emptiness can be decided simply by checking if the expression is ϵ. Membership of some word w can be decided by considering each of the finite permutations of elements from Γ that map to w under h, and checking membership of any of those in any regular expression that maps to the DPE under h. The union of two DPEs x and y is also decidable, simply as $x + y$.

Unlike regular expressions, however, the intersection of DPEs is not decidable, as can be seen with an embedding of the Post-Correspondence Problem (PCP) below. It should be noted that similar undecidable results concerning Kleene algebras with commutativity conditions do already exist in the literature [8, 9]. However, the formulation of the proof below in terms of DPEs is more direct and intuitive for application to ProbNetKAT, as an indexed set of words looks quite similar to a set of histories marked by head packet. I conjecture that the related undecidable results for such commutative Kleene algebras can be embedded into ProbNetKAT similarly to the method shown.

▶ **Theorem 1.** *For arbitrary DPEs A and B, $A \cap B = \emptyset$ is undecidable.*

Proof. To show the undecideability of emptiness of intersection, we will start by defining notation for summing in an expression across multiple terms. Let I be a finite, indexed set of n with element m denoted i_m, and let e_{i_m} be an expression predicated on element i_m. Because addition of expressions is commutative and there are only finitely many terms, this is well-defined.

$$\sum_{i \in I} e_i = e_{i_1} + e_{i_2} + ... + e_{i_n}$$

Now take an arbitrary instance of the PCP. The PCP is the following decision problem: Given an indexed set of word pairs $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ on an alphabet (given here by Σ), does there exist a non-empty sequence of indices $i_1, i_2, ..., i_m$ such that $x_{i_1} \cdot x_{i_2} \cdot ... \cdot x_{i_m} = y_{i_1} \cdot y_{i_2} \cdot ... \cdot y_{i_m}$?

We then provide the two regular X and Y below with $n = 2$, involving the left and right sides of the PCP pairs. This uses the summation definition and $1 \leq k \leq n$ as shorthand for $k \in \{m \in \mathbb{N} | 1 \leq m \leq n\}$.

$$X = [\sum_{1 \leq k \leq n} (x_k)_{(1)} \cdot (y_k)_{(2)}] \cdot [\sum_{1 \leq k \leq n} (x_k)_{(1)} \cdot (y_k)_{(2)}]^*$$

$$Y = [\sum_{a \in \Sigma} a_{(1)} \cdot a_{(2)}] \cdot [\sum_{a \in \Sigma} a_{(1)} \cdot a_{(2)}]^*$$

X generates every possible non-empty ordering of the word pairs from the PCP with each side of the pairs at its own index, and Y generates every possible non-empty word in Σ^* copied at each index.

Mapping these sets under h now actually separates the words and indices into their designated positions, such that we are left with a pair of two words. For example, if (wo, w) and (rd, ord) were PCP pairs, then we would see $h((wo)_{(1)} \cdot w_{(2)} \cdot (rd)_{(1)} \cdot (ord)_{(2)}) = (word, word)$.

In that way, the mapping of X under h to get the DPE A gives us every pair of words generated by different sides of the PCP pairs with the same index sequence, and the mapping of Y to get the DPE B gives us every pair of matching words. Thus, if we could decide if there exists a word (s, t) in both A and B, we would know that the s and t are matching words made from the same index sequence, comprising an affirmative solution to the PCP problem. And if such a word did not exist, then there would be no solution, as every possible word from every possible sequence of indices is represented. This therefore decides the PCP. But, the PCP is undecidable, so the existence of any such word (s, t) in the intersection is also undecidable. ◀

ProbNetKAT DPEs. Suppose one has two ProbNetKAT programs p and q and wishes to decide if their support sets (where support is used in the discrete sense as the set given by $\mathsf{supp}(\mu)$ of points for which the distribution μ assigns positive mass). This could be used to determine if q successfully avoids every set of paths through waypoints that p routes through, regardless of the probability with which any permutation of waypoints is used. Perhaps some of p's route sets have been compromised, but exactly which ones are unknown. Or it could serve as a preliminary test for equality of the programs' distributions, as the decision procedure for equality might be difficult, and is in fact unknown at the time of this writing; certainly, if the supports are completely different, the distributions must also differ.

Or suppose that one wishes to check the size of the output set of a program on a given input. Perhaps this would be to ensure that packets never get too congested by using too few routes.

Unfortunately, the first of these problems is undecidable for fixed inputs with more than one distinct head packet. This can be shown in a discrete distribution case by showing that support intersection emptiness decides the emptiness of intersection of DPEs. The same example can further be used to show that the second problem, determining output size, is undecidable as well. These results do not reflect on the decidability of intersection or output size of non-random NetKAT programs, however. NetKAT programs can in fact be represented with regular sets of guarded strings [1] , so these two questions are easily decidable in NetKAT.

To show this, we will construct an embedding for DPEs in ProbNetKAT. This embedding will satisfy that $w = (w_1, w_2, ...w_n)$ is in a given DPE with alphabet Σ if and only if, given the set $\{\pi_1, \pi_2 ... \pi_n\}$ to start, the corresponding ProbNetKAT program puts out the history set $\{\pi_1 :: w_1, \pi_2 :: w_2, ...\pi_n :: w_n\}$ with positive probability, where the set of packets Pk is given by $\Sigma \cup \Pi \cup \Psi$, $\Pi = \{\pi_i | 1 \leq i \leq n\}$, and Ψ the appropriate packet set for the random loop from earlier. The embedding g is defined inductively below on regular expressions over $a_{(i)} \in \Gamma$, which yield the appropriate DPEs under h.

$$g(\epsilon) = [\![skip]\!]$$

$$g(a_{(i)}) = [\![\text{if } \pi_i? \text{ then } a!; dup; \pi_i! \text{ else } skip]\!]$$

$$g(x \cdot y) = g(y); g(x)$$

$$g(x + y) = g(x) \oplus g(y)$$

$$g(x^*) = [\![g(x)^{G_{0.5}}]\!]$$

▶ **Theorem 2.** *The function g yields a valid embedding of DPEs into ProbNetKAT, i.e., for a regular expression s on Γ, g satisfies that*

$$(w_1, ...w_n) \in h(s) \leftrightarrow g(s)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0.$$

Proof. This follows routinely by induction on the structure of regular expressions. See the appendix for the explicit proof. The trick is that each head packet acts as a label for its history, denoting the index or track of the regular expression to which that history matches in the DPE. ◀

▶ **Theorem 3.** *For arbitrary $a \in 2^H$ and ProbNetKAT programs p and q, whether or not $\mathsf{supp}([\![p]\!](a)) \cap \mathsf{supp}([\![q]\!](a)) = \emptyset$ is undecidable.*

Proof. Take the regular expressions X and Y from the section on DPEs. Because $h(X) \cap h(Y)$ is undecidable (as was proved in the above section on DPEs), we find that $\mathsf{supp}(g(X)(\{\pi_1, \pi_2\})) \cap \mathsf{supp}(g(Y)(\{\pi_1, \pi_2\})) = \emptyset$ is also undecidable. This is an example of the type of problem we wish to prove undecidable, so we are done. ◀

▶ **Theorem 4.** *For arbitrary $a \in 2^H$, $n \in \mathbb{N}$, and ProbNetKAT program p, whether or not $\exists b \in \mathsf{supp}([\![p]\!](a))$ such that $|b| = n$ is undecidable.*

Proof. Take the regular expressions X and Y from the section on DPEs. Consider the program given by $[\![g(X)\&g(Y)]\!]$.

Recognize that the program made by g always outputs a set of size 2 given $\{\pi_1, \pi_2\}$ as input. No base operation of g changes the number of elements in the set it is given, so this follows easily by induction. Then recall that the & operation essentially performs each of its two arguments' code independently, before taking the union of the results, respecting probability. The union of the results of $g(X)$ and $g(Y)$ on the input $\{\pi_1, \pi_2\}$ is thus of size 2 with positive probability iff there is some output made by both programs with positive probability, because the only way the union of two sets of the same finite size does not go up in size is if the two sets are the same. This means if we could decide if no output in the support was of size two, we could also decide the emptiness of support intersection. Emptiness of support intersection is undecidable, so we also cannot decide if the program $[\![g(X)\&g(Y)]\!]$ given $\{\pi_1, \pi_2\}$ has an output of size two in its support. ◀

▶ **Theorem 5.** *For arbitrary $a \in 2^H$, $n \in \mathbb{N}$, and ProbNetKAT program p, whether or not $\exists b \in \mathsf{supp}([\![p]\!](a))$ such that $|b| \leq n$ is undecidable.*

Proof. Consider the same set up as in theorem 4. Note again that the program made by g always outputs a set of size 2 given $\{\pi_1, \pi_2\}$ as input, and that the union of two sets is never less than the size of the either set. Thus, all sets in the support of $[\![g(X)\&g(Y)]\!]\{\pi_1, \pi_2\}$ are of size 2 or more. If there was a set b in that support of size at most 2, it would in fact be of size 2, and such a b of size 2 is of course size at most 2. Therefore, deciding if there is such b of size at most 2 is precisely the same as deciding if there is such a b of size 2 exactly. It is undecidable if there is such a set b of size 2, as proven in theorem 4, so it also undecideable if there is a set of size at most 2. ◀

4.2 A PFA embedding

PFAs Probabilistic finite automata (PFAs), are the probabilistic extension of DFAs. Rather than each transition deterministically moving from the current state to another state on a given symbol, each transition moves to one of several states with specific probability. The language, rather than being a set of strings, is a distribution on strings. This can be formally defined with the tuple $(Q, \Sigma, \Delta, q_0, F)$, where Q is a finite set of states, Σ is a finite set of symbols, Δ is a set of matrices, q_0 is the starting state, and F represents the set of final states. More specifically, Δ is a set of square stochastic matrices Δ_a indexed by states, such

that for each $a \in \Sigma$, $\Delta_a(x, y)$ is the probability that state x transitions to state y on input symbol a. It will be useful as well to let q_0 also stand in for the horizontal vector of zeroes save for a one at index q_0, and F stand in for the vertical vector of zeroes save for those indices that are in F.

Using the notation that $\Delta_{w \cdot w'} = \Delta_w \cdot \Delta_{w'}$ for $w, w' \in \Sigma^*$, we can then define the word distribution $D(M)$ for a PFA M as the function from $w \in \Sigma^*$ to $q_0 \cdot \Delta_w \cdot F$, which returns the probability that w will be accepted. This can then be used to define the language of M for a given threshold λ as follows.

$$L_\lambda(M) = \{w \in \Sigma^* | D(M)(w) \geq \lambda\}$$

There are a few undecidable problems associated with PFAs from previous work on the subject [6, 2], some of which are closely related. These include:

- The emptiness problem: $\exists w \in \Sigma^*.D(M)(w) \geq \lambda$
  (Is the language empty for a given cutpoint?)
- The strict emptiness problem: $\exists w \in \Sigma^*.D(M)(w) > \lambda$
  (Is the strict variant of the language empty for a given cutpoint?)
- The equality problem: $\exists w \in \Sigma^*.D(M)(w) = 0.5$
  (Is there any word accepted exactly half the time?)
- The isolation problem: $\exists \epsilon > 0. \forall w \in \Sigma^*.|D(M)(w) - \lambda| \geq \epsilon$
  (Are there words that are accepted with probability arbitrarily close to a given value?)
- The value 1 problem: $\exists \epsilon > 0. \forall w \in \Sigma^*.D(M)(w) \leq 1 - \epsilon$
  (Are there words that are accepted with arbitrarily high probability?)

Equivalence and certain kinds of approximations between PFAs, however, are decidable [3, 7, 13]. This means equivalence and those approximations of PFAs will not be as useful here for finding undecidable problems. Nor do they directly yield analagous decision procedures for general ProbNetKAT programs, as the provided embedding does not fully encompass the forms that ProbNetKAT programs can take.

ProbNetKAT PFAs. It is easy to create ProbNetKAT code that only does meaningful work on certain input sets, like code that begins by dropping everything with a certain head packet. This might arise in networking if some subset of the packets Pk are the only ones properly formatted as request packets for the network, so any other packet is discarded. It is also easy to create code that produces some output with some form of geometric distribution, as the random loop does. For this reason, it would be useful to be able to compare a ProbNetKAT program's distribution on specific sets to a geometric distribution, to see whether or not they are equal within a certain error, or whether one's probabilities dominate the other's.

One might also like to do the same sorts of comparisons between programs. Because these are probabilistic algorithms, approximation within a certain error is often just as good in application as actual equality. Determining whether one program's distribution on certain sets dominates another's could be useful in a context where one is trying to improve successful service probability. For such a case, suppose that one's current neworking program drops all packets (fails) with a positive probability, and that one wishes to improve it so that it fails less often, but doesn't lower the probability of any of its successful output. This would be solved by determining if a candidate replacement program dominated the original on non-empty outputs.

Unfortunately, because PFAs can be embedded in ProbNetKAT, these sorts of problems are often undecidable. These problems also do not have any clear analogues in non-random NetKAT, as they are intimately concerned with the properties of probability distributions.

The function g performs this embedding for a PFA $M = (Q, \Sigma, \Delta, q_0, F)$. Define $Pk = \Pi \cup \Psi \cup \Sigma$, where $\Pi = \{\pi_q | q \in Q\}$, Ψ as the image of Π under $x \leftarrow false$, $0 < r < 1$, and $0 < s \leq \frac{1}{|\Sigma|}$. Both Π and Ψ maintain their roles as given in section 3 for random looping. We can then give g with the following, which works as will be described in theorem 6.

$$g(M) = [\underset{q \in Q}{\&} \text{ if } \pi_q? \text{ then } \underset{a \in \Sigma}{\oplus_s} a!; dup; (\underset{t \in Q}{\oplus_{\Delta_a(q,t)}} \pi_t!) \text{ else } drop]^{G_r}; (\underset{f \in F}{\&} \pi_f?); \pi_{q_0}!$$

▶ **Theorem 6.** *The function g yields a valid embedding of PFAs into ProbNetKAT, i.e., for a PFA M, g satisfies that*

$$g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: rev(w)\}) = (rs)^{|w|} \cdot (1 - r) \cdot D(M)(w)$$

where $rev(x)$ is the function that reverses x.

Proof. This validity of this statement can be seen by considering traversal through the code dynamically. The head packet is used to record the state of the PFA, so it starts at the packet corresponding to the starting state of the PFA, q_0. The code then enters a random loop. Each iteration of this loop starts by determining which state the configuration is in by checking head packets, and picking a random letter/packet to add to the front of the history. By putting letters on the front each time, the sequence of letters chosen is recorded as the reverse of the packet history. If it randomly drops here instead of picking a letter, then every branch of parallel execution will have dropped, so the code will output the empty set. If a letter is picked and duped into the history, then, knowing which state the configuration is in and which letter it chose, the code chooses a new head packet with probability in accordance with Δ.

Thus each iteration of the loop corresponds to a transition in the PFA, and either maintains a singleton set or drops to the empty set. For each iteration of the random loop that doesn't drop to the empty set, not only is the probability multiplied by r to continue, but also by s to pick a letter, and further by the transition probability. If the sequence of letters chosen after $|w|$ iterations is the word w, then these transition probabilities multiply to $D(w)$ as in the PFA, for a total probability of $(rs)^{|w|}D(w)$ for having reached that iteration with those choices. When the random loop finally exits, it does so with a probability $1 - r$, for a total probability of $(rs)^{|w|}D(w) \cdot (1 - r)$.

After going through this loop some arbitrary number of times and finally exiting, the code checks to see if the state is now a final state. If not, it drops, but if so, it standardizes the head packet to that of the initial state. (Nothing here changes the probability.) Thus, any non-empty outputs from the input $\{\pi_{q_0}\}$ correspond to having reached a final state through PFA transitions, and the history of the output is the reverse of the word that led there. ◀

▶ **Theorem 7.** *For an aribtrary ProbNetKAT program p, a set of non-empty history sets B, an arbitrary $a \in 2^H$, arbitrary values u and v, and a function f taking a non-empty set of histories to a linear combination of the contained histories' lengths, whether or not $\exists b \in B.[\![p]\!](a)(b) \geq u \cdot v^{f(b)}$, i.e., whether the distribution on non-empty outputs of p on input a can be bounded by a geometric function of output history length, is undecidable.*

Proof. Take the emptiness problem for PFAs on alphabet Σ. Translate it into ProbNetKAT using g as follows, noting that the reverse of a word in $w \in \Sigma^*$ always exists and always is the same length as w.

$$\begin{aligned}
\exists w \in \Sigma^*.D(M)(w) \geq \lambda &\iff \exists w \in \Sigma^*.(rs)^{|w|} \cdot (1 - r) \cdot D(M)(w) \geq (rs)^{|w|} \cdot (1 - r) \cdot \lambda \\
&\iff \exists w \in \Sigma^*.g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: rev(w)\}) \geq (rs)^{|w|} \cdot (1 - r) \cdot \lambda \\
&\iff \exists w \in \Sigma^*.g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) \geq (rs)^{|w|} \cdot (1 - r) \cdot \lambda
\end{aligned}$$

Because the emptiness problem for PFAs is undecidable, so is the statement on the final line. Letting $g(M) = p$, $a = \{\pi_{q_0}\}$, $B = \{\pi_{q_0} :: w | w \in \Sigma^*\}$, $u = (1-r)\lambda$, $v = rs$, and $f(b) = \sum_{h \in b} |h|$, we find that said statement is an example of the type of problem we are trying to prove undecidable, so we are done. ◀

▶ **Theorem 8.** *For an arbitrary ProbNetKAT programs p and q, an arbitrary $a \in 2^H$, arbitrary values u and v, and a function f taking a non-empty set of histories to a linear combination of the contained histories' lengths, whether or not $\exists b \in B. |[\![p]\!](a)(b) - [\![q]\!](a)(b)| \geq u \cdot v^{f(b)}$, i.e., whether p's distribution approximates q's on non-empty outputs to within an error exponentially decaying with history length, is undecidable.*

Proof. Let q be the program $[\![drop]\!]$. This program's output distribution always assigns probability 0 to non-empty sets.

$$\begin{aligned} \exists b \in B. |[\![p]\!](a)(b) - [\![q]\!](a)(b)| \geq u \cdot v^{f(b)} &\iff \exists b \in B. |[\![p]\!](a)(b) - [\![drop]\!](a)(b)| \geq u \cdot v^{f(b)} \\ &\iff \exists b \in B. |[\![p]\!](a)(b) - 0| \geq u \cdot v^{f(b)} \\ &\iff \exists b \in B. [\![p]\!](a)(b) \geq u \cdot v^{f(b)} \end{aligned}$$

The statement in the final line was shown undecidable in thoerem 7, so our desired theorem is undecidable in the instance when $q = [\![drop]\!]$. It is therefore undecidable. ◀

▶ **Theorem 9.** *For an aribtrary ProbNetKAT program p, a set of non-empty history sets B, an arbitrary $a \in 2^H$, arbitrary values u and v, and a function f taking a non-empty set of histories to a linear combination of the contained histories' lengths, the following are undecidable.*

- $\exists b \in B. [\![p]\!](a)(b) > u \cdot v^{f(b)}$
  (Is p's distribution on non-empty outputs bounded from above by a given geometric function that changes with output history length?)
- $\exists b \in B. [\![p]\!](a)(b) = u \cdot v^{f(b)}$
  (Does p's distribution on non-empty outputs ever coincide with a given geometric function that changes with output history length?)
- $\exists \epsilon > 0. \forall b \in B. |[\![p]\!](a)(b) - u \cdot v^{f(b)}| \geq u \cdot v^{f(b)} \cdot \epsilon$
  (Does p's distribution on non-empty outputs get within an arbitrarily small scalar of a given geometric function that changes with output history length?)
- $\exists \epsilon > 0. \forall b \in B. [\![p]\!](a)(b) \leq u \cdot v^{f(b)} \cdot (1-\epsilon)$
  (Can p's distribution on non-empty outputs be bounded from above by a scalar<1 of a given geometric function that changes with output history length?)

Proof. Follow the same translation procedure as theorem 7. Starting from the strict emptiness problem, equality problem, isolation problem, and value 1 problem for PFAs, respectively. See the appendix for an explicit proof. ◀

▶ **Theorem 10.** *For arbitrary ProbNetKAT programs p and q, a set of non-empty history sets B, and arbitrary $a \in 2^H$, the following are undecidable.*

- $\exists b \in B. [\![p]\!](a)(b) \geq [\![q]\!](a)(b)$
  (Is p's distribution on non-empty outputs bounded strictly from above (strongly dominated) by q's?)
- $\exists b \in B. [\![p]\!](a)(b) > [\![q]\!](a)(b)$
  (Is p's distribution on non-empty outputs bounded from above (weakly dominated) by q's?)

- $\exists b \in B. [\![p]\!](a)(b) = [\![q]\!](a)(b)$
  (Does p's distribution on non-empty outputs ever coincide with q's?)
- $\exists \epsilon > 0. \forall b \in B. |[\![p]\!](a)(b) - [\![q]\!](a)(b)| \geq [\![q]\!](a)(b) \cdot \epsilon$
  (Does p's distribution on non-empty outputs get within an arbitrarily small scalar of q's?)
- $\exists \epsilon > 0. \forall b \in B. [\![p]\!](a)(b) \leq [\![q]\!](a)(b) \cdot (1 - \epsilon)$
  (Can p's distribution on non-empty outputs be bounded from above by a scalar<1 of q's?)

Proof. Consider the following program p

$$[\![p]\!] = [\![(\underset{a \in \Sigma}{\oplus_s} a!; dup)^{G_r}; \pi_{q_0}! \oplus_\lambda drop]\!]$$

This program's output distribution on input $\{\pi_{q_0}\}$ is given for non-empty output sets by

$$[\![p]\!](\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) = (rs)^{|w|} \cdot (1 - r) \cdot \lambda$$

This distribution is precisely the term that we showed was undecideably comparable to arbitrary programs in theorems 7 and 9. Substituting that term, not with $u \cdot v^{f(b)}$ as was done in theorems 7 and 9, but rather with $[\![q]\!](a)(b)$, yields instances of each of the above problems. As values were merely substituted for identical values, the statements are still undecidable. Thus, each of the problems named are not generally decidable. ◀

5 Conclusion

By encoding the Post Correspondence Problem and various undecidable problems of probabilistic finite automata, we have been able to show that various problems for ProbNetKAT are undecidable. These include emptiness of support intersection, size of the output set, dominance of distribution probabilities over other programs', the satisfaction of geometric distribution bounds, and more.

However, it is still open whether or not ProbNetKAT program equality is decidable. NetKAT's equality is decidable, and many useful networking problems like waypointing are reducible to program equality. There is hope that the same power could be achieved in ProbNetKAT. It is known that equality of ProbNetKAT programs is decidable if one removes random choice (since that is just NetKAT), if one removes *dup* (which can be shown with some linear algebra and Markov chains [11]), and if one removes the asterate (since distributions become finite). It still remains to be seen if ProbNetKAT program equality can be decided if all three are present. Unfortunately it may be the case that with all three features, program equality becomes undecidable. In such an event, embeddings like those shown here may be instrumental in proving undecidability.

Future work could also be done to determine if it is decidable whether one program approximates another to within a constant error bound. Being probabilistic, this is often as good as equality in application. We have shown here that it is undecidable on non-empty outputs where the error decays exponentially with history length (theorem 8).

Acknowledgements. Thanks goes to Dexter Kozen for mentorship and support throughout this project, Steffen Smolka for insightful discussions about ProbNetKAT and beyond, and Eric Perdew for helpful revision. Additional thanks goes to the anonymous reviewers for their valuable comments.

References

1 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: Semantic foundations for networks. In *Proc. 41st ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL'14)*, pages 113–126, San Diego, California, USA, January 2014. ACM.

2 Vincent D Blondel, Vincent Canterini, et al. Undecidable problems for probabilistic automata of fixed dimension. *Theory of Computing systems*, 36(3):231–245, 2003.

3 Corinna Cortes, Mehryar Mohri, and Ashish Rastogi. *On the Computation of Some Standard Distances Between Probabilistic Automata*, pages 137–149. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. doi:10.1007/11812128_14.

4 Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic NetKAT. In Peter Thiemann, editor, *25th European Symposium on Programming (ESOP 2016)*, volume 9632 of *Lecture Notes in Computer Science*, pages 282–309, Eindhoven, The Netherlands, April 2016. Springer.

5 Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for NetKAT. In *Proc. 42nd ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL'15)*, pages 343–355, Mumbai, India, January 2015. ACM.

6 Hugo Gimbert and Youssouf Oualhadj. *Probabilistic Automata on Finite Words: Decidable and Undecidable Problems*, pages 527–538. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. URL: http://dx.doi.org/10.1007/978-3-642-14162-1_44, doi:10.1007/978-3-642-14162-1_44.

7 Stefan Kiefer, Andrzej S. Murawski, Joël Ouaknine, Björn Wachter, and James Worrell. *On the Complexity of the Equivalence Problem for Probabilistic Automata*, pages 467–481. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-28729-9_31.

8 Dexter Kozen. Kleene algebra with tests and commutativity conditions. In T. Margaria and B. Steffen, editors, *Proc. Second Int. Workshop Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 14–33, Passau, Germany, March 1996. Springer-Verlag.

9 Dexter Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.

10 Dexter Kozen. NetKAT: A formal system for the verification of networks. In Jacques Garrigue, editor, *Proc. 12th Asian Symposium on Programming Languages and Systems (APLAS 2014)*, volume 8858 of *Lecture Notes in Computer Science*, Singapore, November 17–19 2014. Asian Association for Foundation of Software (AAFS), Springer.

11 Steffen Smolka, David Kahn, Praveen Kumar, and Nate Foster. Deciding probabilistic program equivalence in NetKAT. http://www.cs.cornell.edu/~smolka/papers/mcnetkat.pdf, 2017.

12 Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. Cantor meets Scott: Domain-theoretic foundations for probabilistic network programming. In *Proc. 44th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL'17)*, pages 557–571, Paris, France, January 2017. ACM.

13 Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing*, 21(2):216–227, 1992. doi:10.1137/0221017.

A Appendix

A.1 Theorem 2 in Detail

Theorem 2 The function g yields a valid embedding of DPEs into ProbNetKAT, i.e., for a regular expression s on Γ, g satisfies that

$$w \in h(s) \leftrightarrow g(s)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0.$$

Proof. Define the predicate $P(s)$ for regular expression s to denote that g validly embeds s. We can then use induction on regular expressions to show that P holds for all expressions.

The base case expressions of our induction are the empty expression ϵ and each symbol in the alphabet Σ.

To get the case of ϵ, note that the only element in the support of $[\![skip]\!]\{\pi_1, ..., \pi_n\}$ is $\{\pi_1, ..., \pi_n\}$ itself, which has no histories beyond the head packets. Any words made from the histories must therefore be empty.

$$\begin{aligned}
&P(\epsilon): \\
w \in h(\epsilon) &\leftrightarrow \forall i, w_i = \epsilon \\
&\leftrightarrow \{\pi_1, ..., \pi_n\} = \{\pi_1 :: w_1, ..., \pi_n :: w_n\} \\
&\leftrightarrow [\![skip]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0 \\
&\leftrightarrow g(\epsilon)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0
\end{aligned}$$

To get the case of the single symbol, note that the code again only leaves a single element in the support:$\{\pi_1, ..., \pi_n\}$ with a single new packet added to the history of a single head packet. The only word made from such histories is thus that single symbol with the index of the head packet it is behind.

$$\begin{aligned}
&P(a_{(i)}): \\
w \in h(a_{(i)}) &\leftrightarrow w_i = a \wedge \forall j \neq i. w_j = \epsilon \\
&\leftrightarrow \pi_i :: w_i = \pi_i :: a \wedge \forall j \neq i. \pi_j :: w_j = \pi \\
&\leftrightarrow [\![\text{ if } \pi_i? \text{ then } a!; dup; \pi_i! \text{ else } skip]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0 \\
&\leftrightarrow g(a_{(i)})(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0
\end{aligned}$$

Now, assuming that the embedding is valid for smaller expressions x and y, we consider the inductive steps for each of the operators: multiplication, addition, and asterate.

Note that adding new elements to the history stores them in the reverse order of their addition, so we end up reversing the order of x and y terms. Further recall that the computation done on a packet at a given point is determined solely by the head packet, not the history, as no operation can read from the history. This allows us to prove the case of multiplication as follows.

$$\begin{aligned}
&P(x \cdot y): \\
w \in h(x \cdot y) &\leftrightarrow w \in h(x) \cdot h(y) \\
&\leftrightarrow \exists u, v \in (\Sigma^*)^n. u \in h(x) \wedge v \in h(y) \wedge u \cdot v = w \\
&\leftrightarrow \exists u, v \in (\Sigma^*)^n. g(x)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: u_1, ..., \pi_n :: u_n\}) > 0 \\
&\quad \wedge g(y)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: v_1, ..., \pi_n :: v_n\}) > 0 \wedge u \cdot v = w \\
&\leftrightarrow \exists u, v \in (\Sigma^*)^n. [\![g(y); g(x)]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: u_1 :: v_1, ..., \pi_n :: u_n :: v_n\}) > 0 \\
&\quad \wedge \forall i, u_i \cdot v_i = w_i \\
&\leftrightarrow [\![g(y); g(x)]\!](\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0 \\
&\leftrightarrow g(x \cdot y)(\{\pi_1, ..., \pi_n\})(\{\pi_1 :: w_1, ..., \pi_n :: w_n\}) > 0
\end{aligned}$$

The case of addition follows because scaling a probability by 0.5 does not change its positivity.

$$
\begin{aligned}
&P(x+y):\\
w \in h(x+y) &\leftrightarrow w \in h(x) \cup h(y)\\
&\leftrightarrow w \in h(x) \vee w \in h(y)\\
&\leftrightarrow g(x)(\{\pi_1,...,\pi_n\})(\{\pi_1 :: w_1,...,\pi_n :: w_n\}) > 0\\
&\qquad \vee g(y)(\{\pi_1,...,\pi_n\})(\{\pi_1 :: w_1,...,\pi_n :: w_n\}) > 0\\
&\leftrightarrow [\![g(x) \oplus_{0.5} g(y)]\!](\{\pi_1,...,\pi_n\})(\{\pi_1 :: w_1,...,\pi_n :: w_n\}) > 0\\
&\leftrightarrow [\![g(x+y)]\!](\{\pi_1,...,\pi_n\})(\{\pi_1 :: w_1,...,\pi_n :: w_n\}) > 0
\end{aligned}
$$

Finally, the asterate case follows from recalling that the random loop is capable stopping after any number of iterations, and always does so with a positive probability.

$$
\begin{aligned}
&P(x^*):\\
w \in h(x^*) &\leftrightarrow w \in \bigcup_{n\in\mathbb{N}} h(x)^n\\
&\leftrightarrow \exists n \in \mathbb{N}. w \in h(x)^n\\
&\leftrightarrow \exists n \in \mathbb{N}. [\![g(x)^n]\!](\{\pi_1,...,\pi_n\})(\{\pi_1 :: w_1,...,\pi_n :: w_n\}) > 0\\
&\leftrightarrow [\![g(x)^{G_{0.5}}]\!](\{\pi_1,...,\pi_n\})(\{\pi_1 :: w_1,...,\pi_n :: w_n\}) > 0\\
&\leftrightarrow [\![g(x^*)]\!](\{\pi_1,...,\pi_n\})(\{\pi_1 :: w_1,...,\pi_n :: w_n\}) > 0
\end{aligned}
$$

The validity of the embedding holds for all base elements and through all operations of regular expressions on Γ. Thus, by induction, the validity of the embedding holds for all regular expressions.

◀

A.2 Theorem 9 in Detail

Theorem 9 For an aribtrary ProbNetKAT program p, a set of non-empty history sets B, an arbitrary $a \in 2^H$, arbitrary values u and v, and a function f taking a non-empty set of histories to a linear combination of the contained histories' lengths, the following are undecidable.

- $\exists b \in B. [\![p]\!](a)(b) > u \cdot v^{f(b)}$
  (Is p's distribution on non-empty outputs bounded from above by a given geometric function that changes with output history length?)
- $\exists b \in B. [\![p]\!](a)(b) = u \cdot v^{f(b)}$
  (Does p's distribution on non-empty outputs ever coincide with a given geometric function that changes with output history length?)
- $\exists \epsilon > 0. \forall b \in B. |[\![p]\!](a)(b) - u \cdot v^{f(b)}| \geq u \cdot v^{f(b)} \cdot \epsilon$
  (Does p's distribution on non-empty outputs get within an arbitrarily small scalar of a given geometric function that changes with output history length?)
- $\exists \epsilon > 0. \forall b \in B. [\![p]\!](a)(b) \leq u \cdot v^{f(b)} \cdot (1-\epsilon)$
  (Can p's distribution on non-empty outputs be bounded from above by a scalar<1 of a given geometric function that changes with output history length?)

Proof. Take the strict emptiness problem for PFAs on alphabet Σ. Translate it into ProbNetKAT using g as follows, noting that the reverse of a word in $w \in \Sigma^*$ always exists and always is the same length as w.

$$\begin{aligned}
\exists w \in \Sigma^*.D(M)(w) > \lambda &\iff \exists w \in \Sigma^*.(rs)^{|w|} \cdot (1-r) \cdot D(M)(w) > (rs)^{|w|} \cdot (1-r) \cdot \lambda \\
&\iff \exists w \in \Sigma^*.g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: rev(w)\}) > (rs)^{|w|} \cdot (1-r) \cdot \lambda \\
&\iff \exists w \in \Sigma^*.g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) > (rs)^{|w|} \cdot (1-r) \cdot \lambda
\end{aligned}$$

Because the strict emptiness problem for PFAs is undecidable, so is the statement on the final line. Letting $g(M) = p$, $a = \{\pi_{q_0}\}$, $B = \{\pi_{q_0} :: w | w \in \Sigma^*\}$, $u = (1-r)\lambda$, $v = rs$, and $f(b) = \sum_{h \in b} |h|$, we find that said statement is an example of the first type of problem we are trying to prove undecidable, so we have proved that the first problem in the list is undecidable.

Take the equality problem for PFAs on alphabet Σ. Translate it into ProbNetKAT using g as follows.

$$\begin{aligned}
\exists w \in \Sigma^*.D(M)(w) = 0.5 &\iff \exists w \in \Sigma^*.(rs)^{|w|} \cdot (1-r) \cdot D(M)(w) = (rs)^{|w|} \cdot (1-r) \cdot 0.5 \\
&\iff \exists w \in \Sigma^*.g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: rev(w)\}) = (rs)^{|w|} \cdot (1-r) \cdot 0.5 \\
&\iff \exists w \in \Sigma^*.g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) = (rs)^{|w|} \cdot (1-r) \cdot 0.5
\end{aligned}$$

Because the equality problem for PFAs is undecidable, so is the statement on the final line. Letting $g(M) = p$, $a = \{\pi_{q_0}\}$, $B = \{\pi_{q_0} :: w | w \in \Sigma^*\}$, $u = (1-r)/2$, $v = rs$, and $f(b) = \sum_{h \in b} |h|$, we find that said statement is an example of the second type of problem we are trying to prove undecidable, so we have proved that the second problem in the list is undecidable.

Take the isolation problem for PFAs on alphabet Σ. Pick an instance with a positive λ. Translate it into ProbNetKAT using g as follows.

$$\begin{aligned}
&\exists \epsilon > 0. \forall w \in \Sigma^*. |D(M)(w) - \lambda| \geq \epsilon \\
&\iff \exists \epsilon > 0. \forall w \in \Sigma^*. (rs)^{|w|} \cdot (1-r) \cdot |D(M)(w) - \lambda| \geq (rs)^{|w|} \cdot (1-r) \cdot \epsilon \\
&\iff \exists \epsilon > 0. \forall w \in \Sigma^*. |g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: rev(w)\}) - (rs)^{|w|} \cdot (1-r) \cdot \lambda| \geq (rs)^{|w|} \cdot (1-r) \cdot \epsilon \\
&\iff \exists \epsilon > 0. \forall w \in \Sigma^*. |g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) - (rs)^{|w|} \cdot (1-r) \cdot \lambda| \geq (rs)^{|w|} \cdot (1-r) \cdot \epsilon
\end{aligned}$$

At this point, let $u = (1-r)\lambda$ and $v = rs$. Continue by substituting these variables into the final line, and note that for every $\epsilon/\lambda > 0$ there exists $\epsilon' = \epsilon/\lambda > 0$.

$$\begin{aligned}
&\exists \epsilon > 0. \forall w \in \Sigma^*. |g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) - (rs)^{|w|} \cdot (1-r) \cdot \lambda| \geq (rs)^{|w|} \cdot (1-r) \cdot \epsilon \\
&\iff \exists \epsilon > 0. \forall w \in \Sigma^*. |g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) - u \cdot v^{|w|}| \geq u \cdot v^{|w|} \cdot \epsilon/\lambda \\
&\iff \exists \epsilon > 0. \forall w \in \Sigma^*. |g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) - u \cdot v^{|w|}| \geq u \cdot v^{|w|} \cdot \epsilon
\end{aligned}$$

Because the isolation problem for PFAs is undecidable, so is the statement on the final line. Letting $g(M) = p$, $a = \{\pi_{q_0}\}$, $B = \{\pi_{q_0} :: w | w \in \Sigma^*\}$, and $f(b) = \sum_{h \in b} |h|$, we find that said statement is an example of the third type of problem we are trying to prove undecidable, so we have proved that the third problem in the list is undecidable.

Take the value 1 problem for PFAs on alphabet Σ. Pick an instance with a positive λ. Translate it into ProbNetKAT using g as follows.

$$\begin{aligned}
&\exists \epsilon > 0. \forall w \in \Sigma^*. D(M)(w) \leq 1 - \epsilon \\
&\iff \exists \epsilon > 0. \forall w \in \Sigma^*. (rs)^{|w|} \cdot (1-r) \cdot D(M)(w) \leq (rs)^{|w|} \cdot (1-r) \cdot (1-\epsilon) \\
&\iff \exists \epsilon > 0. \forall w \in \Sigma^*. g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: rev(w)\}) \leq (rs)^{|w|} \cdot (1-r) \cdot (1-\epsilon) \\
&\iff \exists \epsilon > 0. \forall w \in \Sigma^*. g(M)(\{\pi_{q_0}\})(\{\pi_{q_0} :: w\}) \leq (rs)^{|w|} \cdot (1-r) \cdot (1-\epsilon)
\end{aligned}$$

Because the value 1 problem for PFAs is undecidable, so is the statement on the final line. Letting $g(M) = p$, $a = \{\pi_{q_0}\}$, $B = \{\pi_{q_0} :: w | w \in \Sigma^*\}$, $u = (1-r)\lambda$, $v = rs$, and $f(b) = \sum_{h \in b} |h|$, we find that said statement is an example of the final type of problem we are trying to prove undecidable, so we are done. ◀

Computational Complexity of Graph Partition underVertex-Compaction to an Irreflexive Hexagon

Narayan Vikas

School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
vikas@cs.sfu.ca

Abstract

In this paper, we solve a long-standing graph partition problem under vertex-compaction that has been of interest since about 1999. The graph partition problem that we consider in this paper is to decide whether or not it is possible to partition the vertices of a graph into six distinct non-empty sets A, B, C, D, E, and F, such that the vertices in each set are independent, i.e., there is no edge within any set, and an edge is possible but not necessary only between the pairs of sets A and B, B and C, C and D, D and E, E and F, and F and A, and there is no edge between any other pair of sets. We study the problem as the vertex-compaction problem for an irreflexive hexagon (6-cycle). Determining the computational complexity of this problem has been a long-standing problem of interest since about 1999, especially after the results of open problems obtained by the author on a related compaction problem appeared in 1999. We show in this paper that the vertex-compaction problem for an irreflexive hexagon is NP-complete. Our proof can be extended for larger even irreflexive cycles, showing that the vertex-compaction problem for an irreflexive even k-cycle is NP-complete, for all even $k \geq 6$.

1998 ACM Subject Classification Computations on Discrete Structures, Graph Algorithms

Keywords and phrases computational complexity, algorithms, graph, partition, colouring, homomorphism, retraction, compaction, vertex-compaction

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.69

1 Introduction

The vertex-compaction problem and the compaction problem are special graph colouring problems, and can also be viewed as graph partition problems. The colouring problem is a classic problem in graph theory. The graph homomorphism problem, also called the H-colouring problem, is a generalization of the colouring problem. The vertex-compaction problem is the graph homomorphism problem with additional constraints. The compaction problem is the vertex-compaction problem with additional constraints. We describe our motivation and results after introducing the following definitions and problems.

1.1 Definitions

The pair of vertices of an edge in a graph are called the *endpoints* of the edge. An edge with the same endpoints in a graph is called a *loop*. A vertex v of a graph is said to have a loop if vv is an edge of the graph. A *reflexive graph* is a graph in which every vertex has a loop. An *irreflexive graph* is a graph in which no vertex has a loop. Any graph, in general, is a *partially reflexive graph*, in which its vertices may or may not have loops. Thus reflexive and irreflexive graphs are special partially reflexive graphs. A *bipartite graph* G is a graph whose

vertex set can be partitioned into two distinct subsets G_A and G_B, such that each edge of G has one endpoint in G_A and the other endpoint in G_B; we say that (G_A, G_B) is a bipartition of G. Thus a bipartite graph is irreflexive by definition. If uv is an edge of a graph then vu is also an edge of the graph, i.e., we assume graphs to be undirected graphs. A cycle of length k is called a k-cycle, $k \geq 3$. A *hexagon* will be used as a synonym for a 6-cycle. We shall denote an irreflexive k-cycle by C_k.

Let G be a graph. We use $V(G)$ and $E(G)$ to denote the vertex set and the edge set of G respectively. Given an induced subgraph H of G, we denote by $G - H$, the subgraph obtained by deleting from G the vertices of H together with the edges incident with them; thus $G - H$ is a subgraph of G induced by $V(G) - V(H)$. The vertices in a set $I \subseteq V(G)$ are said to be *independent* if there is no edge in the subgraph of G induced by I. When a set S is an argument of a mapping f, we define $f(S) = \{f(s)|s \in S\}$. The *distance* between a pair of vertices u and v in G, denoted as $d_G(u, v)$ or $d_G(v, u)$, is the length of a shortest path from u to v in G, if u and v are connected in G; we define $d_G(u, v)$ (and $d_G(v, u)$) to be infinite, if u and v are disconnected in G. The *diameter* of G is the maximum distance between any two vertices in G. The distance between two sets X and Y of vertices in G, denoted as $d_G(X, Y)$ or $d_G(Y, X)$, is the minimum distance between any vertex of X and any vertex of Y in G, i.e., $d_G(X, Y) = min\{d_G(x, y)|x \in X, y \in Y\}$, where $min\ A$ gives the minimum element in a set A. If a set has only one vertex, we may just write the vertex instead of the set. In the following, let G and H be graphs.

A *homomorphism* $f : G \to H$, of G to H, is a mapping f of the vertices of G to the vertices of H, such that if g and g' are adjacent vertices of G then $f(g)$ and $f(g')$ are adjacent vertices of H. If there exists a homomorphism of G to H then G is said to be *homomorphic* to H. Note that if G is irreflexive then G is k-colourable if and only if G is homomorphic to the irreflexive complete graph K_k having k vertices. Thus the concept of a homomorphism generalises the concept of a k-colourability, and the *H-colouring* problem is to decide whether or not G is homomorphic to H. The *H-colouring* problem is trivial and easily seen to be polynomial time solvable if H is bipartite or H has a loop. For any fixed non-bipartite irreflexive graph H, it is shown in [Hell and Nesetril, 1990] that the *H-colouring* problem is NP-complete.

A *compaction* $c : G \to H$, of G to H, is a homomorphism of G to H, such that for every vertex x of H, there exists a vertex v of G with $c(v) = x$, and for every edge hh' of H, $h \neq h'$, there exists an edge gg' of G with $c(g) = h$ and $c(g') = h'$. Note that the first part of the definition for a compaction (the requirement for every vertex x of H) follows from the second part unless H has isolated vertices. If there exists a compaction of G to H then G is said to *compact* to H. Given a compaction $c : G \to H$, if for a vertex v of G, we have $c(v) = x$, where x is a vertex of H, then we say that the vertex v of G *covers the vertex* x of H under c; and if for an edge gg' of G, we have $c(\{g, g'\}) = \{h, h'\}$, where hh' is an edge of H, then we say that the edge gg' of G *covers the edge* hh' of H under c (note in the definition of compaction, it is not necessary that a loop of H be covered by any edge of G under c).

We notice that the notion of a *homomorphic image* described in [Harary, 1969] (also cf. [Hell & Miller, 1979]) coincides with the notion of a compaction in case of irreflexive graphs (i.e., when G and H are irreflexive in the above definition for compaction).

A *vertex-compaction* $c : G \to H$, of G to H, is a homomorphism of G to H, such that for every vertex x of H there exists a vertex v of G with $c(v) = x$. If there exists a vertex-compaction of G to H then G is said to *vertex-compact* to H. We define vertex and edge covering under a vertex-compaction c similarly as for a compaction. Note that every compaction is also a vertex-compaction.

A *retraction* $r : G \to H$, of G to H, with H as an induced subgraph of G, is a homomorphism of G to H, such that $r(h) = h$, for every vertex h of H. If there exists a retraction of G to H then G is said to *retract* to H. Note that every retraction $r : G \to H$ is necessarily also a compaction, and hence a vertex-compaction.

1.2 Vertex-Compaction, Compaction, and Retraction Problems

The problem of deciding the existence of a vertex-compaction to a fixed graph H, called the *vertex-compaction problem for* H, and denoted as *VCOMP-H*, asks whether or not an input graph G vertex-compacts to H.

Our graph partition problem is to decide whether or not it is possible to partition the vertices of a graph into six distinct non-empty sets A, B, C, D, E, and F, such that the vertices in each of these sets are independent, and an edge is possible but not necessary only between the pairs of sets A and B, B and C, C and D, D and E, E and F, and F and A, and there is no edge between any other pair of sets. We note that our graph partition problem is the problem *VCOMP-*C_6.

The problem of deciding the existence of a compaction to a fixed graph H, called the *compaction problem for* H, and denoted as *COMP-H*, asks whether or not an input graph G compacts to H. The compaction problem is a well studied problem over last several years, and includes some popular problems. Results on the compaction problem can be found in [Vikas, 1999, 2002, 2003, 2004a, 2004b, 2004c, 2005, 2011, 2013].

Note that unlike the *H-colouring* problem, the problems *VCOMP-H* and *COMP-H* are still interesting if H is bipartite or H has a loop. Some work on graph partition problems have also been studied in [Feder, Hell, Klein, and Motwani, 1999, 2003] and [Hell, 2014].

The problem of deciding the existence of a retraction to a fixed graph H, called the *retraction problem for* H, and denoted as *RET-H*, asks whether or not an input graph G, containing H as an induced subgraph, retracts to H. Retraction problems have been of continuing interest in graph theory for a long time and have been studied in various literature including [Hell, 1972], [Hell, 1974], [Nowakowski and Rival, 1979], [Pesch and Poguntke, 1985], [Bandelt, Dahlmann, and Schutte, 1987], [Hell and Rival, 1987], [Pesch, 1988], [Bandelt, Farber, and Hell, 1993], [Feder and Hell, 1998], [Feder and Vardi, 1993, 1998], [Feder, Hell, and Huang, 1999], [Vikas, 2004b, 2004c, 2005], etc.

1.3 Motivation and Results

It can be shown that for every fixed graph H, if the problem *COMP-H* is solvable in polynomial time then the problem *VCOMP-H* is also solvable in polynomial time (similarly as in [Vikas, 2004b]). Whether the converse is true is not known. The problem *COMP-*C_6 is shown to be NP-complete in [Vikas, 1999, 2004a]. It turns out that the unique smallest bipartite graph H for which *COMP-H* is NP-complete is C_6 [Vikas, 2004a]. Therefore, with respect to the preceding question on converse, we are motivated to specifically determine the computational complexity of our partition problem *VCOMP-*C_6, to see whether like *COMP-*C_6, it is also NP-complete in support of the converse. We show in this paper that *VCOMP-*C_6 is NP-complete. Determining the computational complexity of *VCOMP-*C_6 has been a long-standing problem of interest since about 1999, especially after results on the computational complexity of *COMP-*C_6 obtained by the author appeared in 1999 [Vikas, 1999]. Determining the computational complexity of *COMP-*C_6 was also a long-standing problem of interest since about 1988, solved by the author in [Vikas, 1999, 2004a]. Although

the problem *VCOMP-C_6* is only a little variation of the problem *COMP-C_6*, it turns out to be another difficult problem to determine its computational complexity.

Similarly, our motivation for the partition problem *COMP-C_6* was with respect to the retraction problem. It can be shown that for every fixed graph H, if the problem *RET-H* is polynomial time solvable then the problem *COMP-H* is also polynomial time solvable [Vikas, 2004b]. However, whether the converse is true is again not known. As discussed in [Vikas, 1999, 2003], the question on converse was also asked by Peter Winkler in 1988 in the context of reflexive graphs, and this was the general problem that motivated Winkler for asking the computational complexity of *COMP-H* when H is a reflexive square, as the unique smallest reflexive graph H for which *RET-H* is NP-complete is a reflexive square. It has been shown in [Vikas, 1999, 2003] that when H is a reflexive square, *COMP-H* is NP-complete. As discussed in [Vikas, 2004a], since the unique smallest bipartite graph H for which *RET-H* is NP-complete is C_6, we are therefore motivated, with respect to the above question on converse, to know whether the problem *COMP-C_6* is also NP-complete like the problem *RET-C_6* supporting the converse. As mentioned above, it is shown in [Vikas, 1999, 2004a] that *COMP-C_6* is NP-complete.

The problem *RET-C_6* is shown to be NP-complete in [Feder, Hell, and Huang, 1999], and also independently by G. MacGillivray in 1988. Since C_4 is a complete bipartite graph, it is easy to see that *RET-C_4*, and hence *COMP-C_4* and also *VCOMP-C_4*, are all polynomial time solvable. In fact, when H is a chordal bipartite graph (which includes C_4), the problem *RET-H* is polynomial time solvable [Bandelt, Dahlmann, and Schutte, 1987], and hence *COMP-H* and *VCOMP-H* are also polynomial time solvable. Thus it follows that the unique smallest bipartite graph H for which *RET-H*, *COMP-H*, and *VCOMP-H* are NP-complete is C_6.

It has been shown in [Hell and Nesetril, 1990] that the *H-colouring* problem is NP-complete for any fixed irreflexive non-bipartite graph H. It follows that *RET-H*, *COMP-H*, and *VCOMP-H* are also NP-complete for any non-bipartite irreflexive graph H, which includes an irreflexive odd k-cycle, for all $k \geq 3$.

As we mentioned earlier, the *H-colouring* problem is trivial and easily seen to be polynomial time solvable when H is a bipartite graph. The natural question for bipartite graphs H, which motivated Pavol Hell and Jaroslav Nesetril (personal communications) around 1988, was to ask for the computational complexity of the *H-colouring* problem with added constraints, namely the problem *COMP-H*, and in particular for the problem *COMP-C_6*.

It can also be shown that for every fixed graph H, if the problem *RET-H* is polynomial time solvable then the problem *VCOMP-H* is also polynomial time solvable (similarly as in [Vikas, 2004b]), but whether the converse is true is not known. Hence, once again, in relation to the converse and the problem *RET-C_6*, we are motivated to know whether the problem *VCOMP-C_6* is NP-complete.

The algorithms given in [Vikas, 2011, 2013] yield a polynomial time algorithm for *VCOMP-C_6* for any input graph of diameter more than four, and it is suggested in [Vikas, 2011, 2013] as a guidance that an input graph of diameter four could be a candidate for *VCOMP-C_6* to be possibly NP-complete. We are thus motivated to see whether *VCOMP-C_6* is indeed NP-complete for an input graph of diameter four, guided by the algorithmic aspects of the vertex-compaction problem studied in [Vikas, 2011, 2013]. The instance of the input graph for which we show *VCOMP-C_6* to be NP-complete in this paper is indeed of diameter four.

Our proof and technique of construction for C_6 can be extended for larger irreflexive even cycles to show that *VCOMP-C_k* is NP-complete, for all even $k \geq 6$. Our proof showing NP-completeness of *VCOMP-C_6* directly uses graphs that we construct just by adding vertices

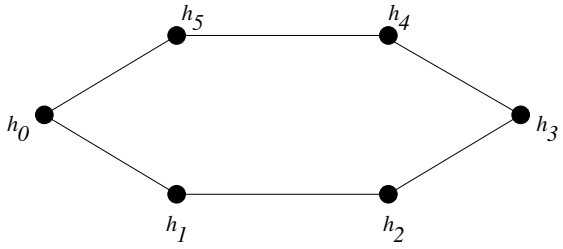


Figure 1 Irreflexive Hexagon H

and edges. Our graphs therefore lay down the foundation for construction of graphs for the case of a general irreflexive even k-cycle, by extending the paths constructed and adding edges appropriately, showing NP-completeness of *VCOMP-*C_k, for all even $k \geq 6$.

In Section 2, we present the proof showing NP-completeness of deciding the existence of a vertex-compaction to an irreflexive hexagon, i.e., the problem *VCOMP-*C_6. In Section 3, we address how our NP-completeness proof of *VCOMP-*C_6 can be extended for an irreflexive even k-cycle, showing NP-completeness of *VCOMP-*C_k, for all even $k \geq 6$.

2 Vertex-Compaction to an Irreflexive Hexagon

▶ **Theorem 2.1.** *The problem of deciding the existence of a vertex-compaction to an irreflexive hexagon is NP-complete.*

Proof. Let H be the irreflexive hexagon $h_0h_1h_2h_3h_4h_5h_0$ shown in Figure 1.

We shall prove that the problem of deciding the existence of a vertex-compaction to H, i.e., the problem *VCOMP-H*, is NP-complete. Clearly, the problem *VCOMP-H* is in NP. We give a polynomial transformation from the problem *RET-H* to *VCOMP-H*. As mentioned earlier, it is known that the problem *RET-H* is NP-complete. Since only a bipartite graph can be homomorphic to H, the problem *RET-H* remains to be NP-complete if the instance of *RET-H* is restricted to be only a bipartite graph.

Let a bipartite graph G, containing H as an induced subgraph, be an instance of *RET-H*. We construct in time polynomial in the size of G, a graph G', containing G as an induced subgraph, such that the following statements (i), (ii), and (iii) are equivalent:

(i) G retracts to H.
(ii) G' retracts to H.
(iii) G' vertex-compacts to H.

Since *RET-H*, with the instance restricted to be a bipartite graph, is NP-complete, this shows that *VCOMP-H* is also NP-complete. We prove that (i) is equivalent to (ii), and (ii) is equivalent to (iii), in two separate lemmas, Lemma 2.2 and Lemma 2.3, respectively.

One of the main challenges is to construct such a graph of diameter four. Let (G_A, G_B) be a bipartition of G, and (H_A, H_B) be a bipartition of H, with $H_A \subseteq G_A$, and $H_B \subseteq G_B$. We shall assume for convenience that $h_0 \in H_B$.

The construction of G' is as follows. For each vertex $a \in G_A - H_A$, we add a new vertex z_a adjacent to a and h_1. For every pair of vertices a and b, with $a \in G_A - H_A$, $b \in G_B - H_B$, we add a new vertex z_{ab} adjacent to z_a and b. Thus for each $a \in G_A - H_A$, we have paths $az_az_{ab}b$, $az_az_{ab'}b'$, for all $b, b' \in G_B - H_B$. See Figure 2. In the figure, we have taken three distinct vertices a, a', and a'' of $G_A - H_A$, and three distinct vertices b, b', and b'' of $G_B - H_B$. Also, in the figures in this section, we are not depicting any edge that may be present between a vertex of $G_A - H_A$ and a vertex of $G_B - H_B$.

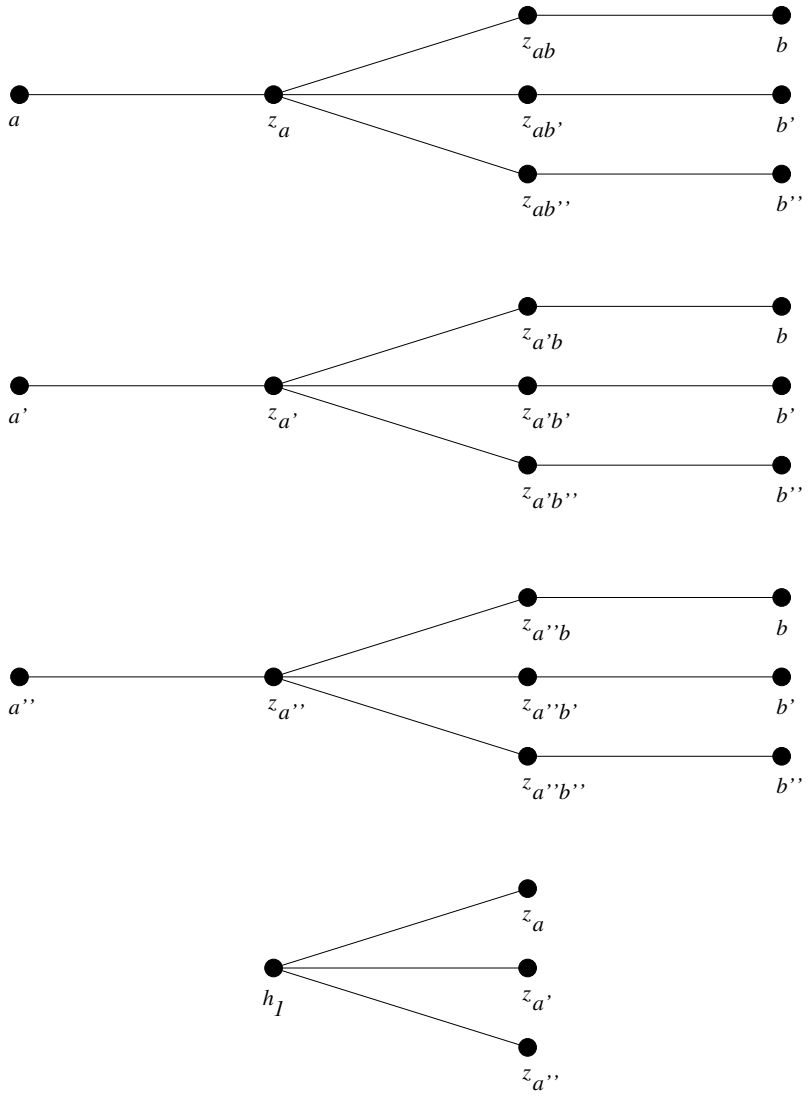


Figure 2 Construction of G', with z_a and z_{ab}, for every pair of vertices a and b, with $a \in G_A - H_A$, $b \in G_B - H_B$

In the graph G' constructed so far, the maximum distance between any pair of vertices in $V(G') - V(H)$ is already four, i.e., $d_{G'}(v, v') \leq 4$, with $v, v' \in V(G') - V(H)$. This can be observed due to the following paths and subpaths within those paths of length at most four : $az_az_{ab}bz_{a'b}$, $z_{ab}z_az_{ab'}b'z_{a'b'}$, $az_ah_1z_{a'}a'$, $bz_{ab}z_az_{ab'}b'$.

If we constructed the graph G' including the vertices of H also, i.e., if we constructed G' for every pair of vertices a and b, with $a \in G_A$, $b \in G_B$ then the diameter of G' would be four.

We continue further with the construction of G'. For each vertex $b \in G_B - H_B$, we add a new vertex x_b adjacent to z_{ab} and h_5, for all $a \in G_A - H_A$. For every pair of vertices a and b, with $a \in G_A - H_A$, $b \in G_B - H_B$, we add a new vertex x_{ba} adjacent to x_b and z_a. Thus for each $b \in G_B - H_B$, we have paths $z_{ab}x_bx_{ba}z_a$, $z_{a'b}x_bx_{ba'}z'_a$, for all $a, a' \in G_A - H_A$. See Figure 3.

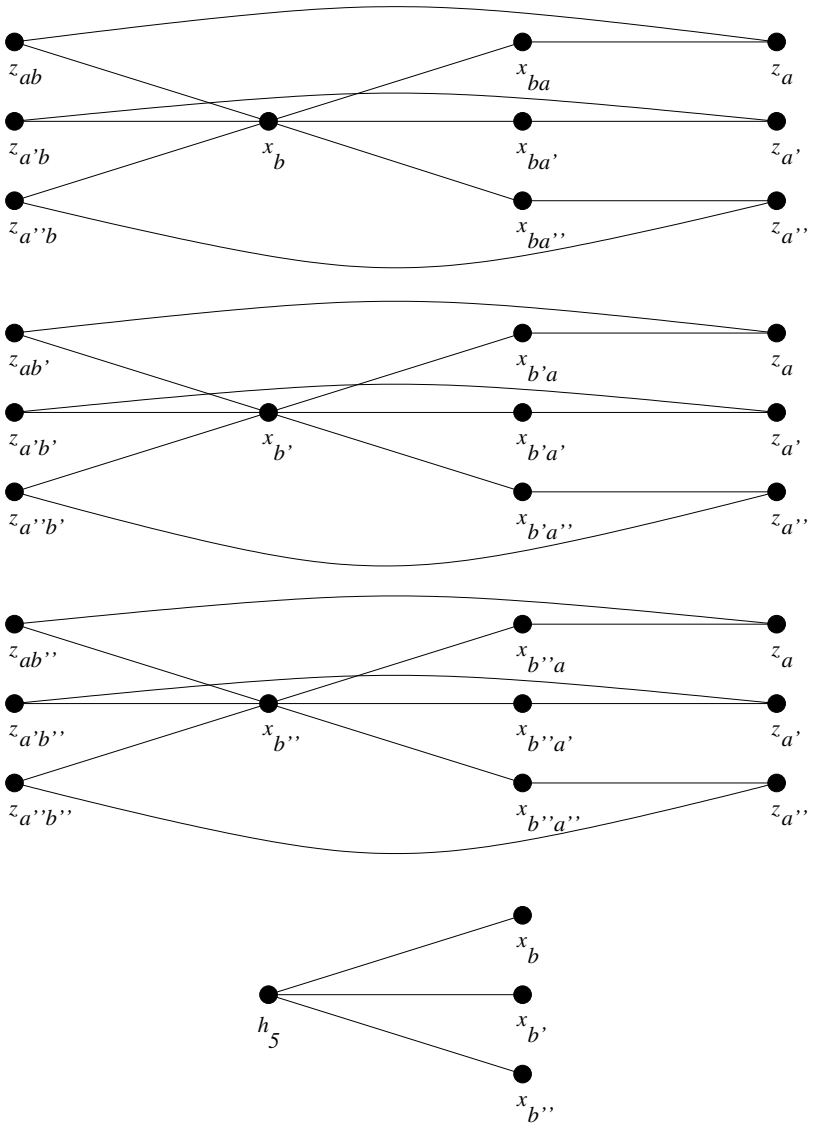


Figure 3 Construction of G', with x_b and x_{ba}, for every pair of vertices a and b, with $a \in G_A - H_A$, $b \in G_B - H_B$

The maximum distance between any pair of vertices in $V(G') - V(H)$ is still four. This can be observed due to the following paths and subpaths within those paths of length at most four : $x_{ba}x_bx_{ba'}z_{a'}x_{b'a'}$, $x_{ba}x_bx_{ba'}z_{a'}z_{a'b'}$, $x_{ba}x_bx_{ba'}z_{a'}a'$, $x_bx_{ba}z_az_{ab'}b'$.

For each vertex $a \in G_A - H_A$ (hence $d_G(h_0, a)$ is odd as we are assuming that $h_0 \in H_B \subseteq G_B$), we add to G new vertices u_1^a adjacent to h_0; u_2^a adjacent to u_1^a, a, and h_1; w_1^a adjacent to h_3, u_1^a, and a; y_1^a adjacent to h_1, u_1^a, and a; y_2^a adjacent to y_1^a, h_4, w_1^a, and u_2^a. See Figure 4. Note that there could be edges in G from a to some vertices of H but in Figure 4, we are not depicting these edges.

For each vertex $b \in G_B - H_B$ (hence $d_G(h_0, b)$ is even), we add to G new vertices u_1^b adjacent to h_0 and b; w_1^b adjacent to h_3 and u_1^b; w_2^b adjacent to w_1^b, b, and h_2; y_1^b adjacent to h_5, u_1^b, and w_2^b; y_2^b adjacent to y_1^b, h_2, w_1^b, and b. See Figure 5. There could be edges in G from b to some vertices of H but in Figure 5, we are not depicting these edges.

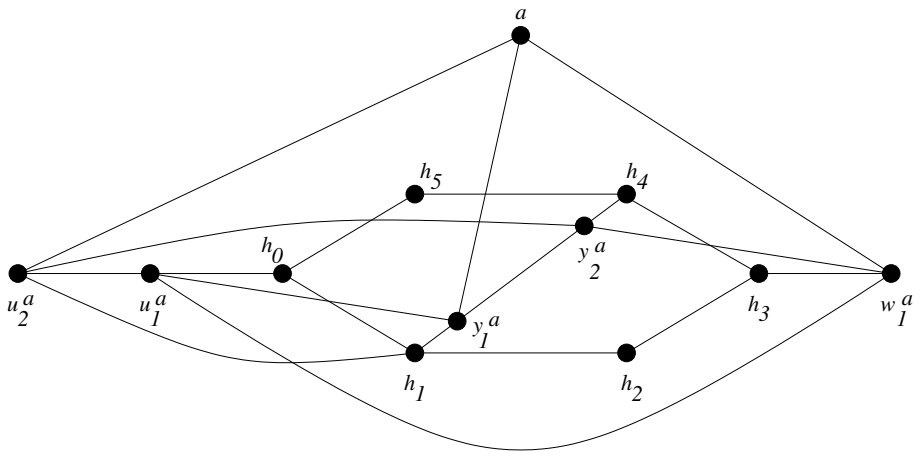


Figure 4 Construction of G' for a vertex a in $G_A - H_A$

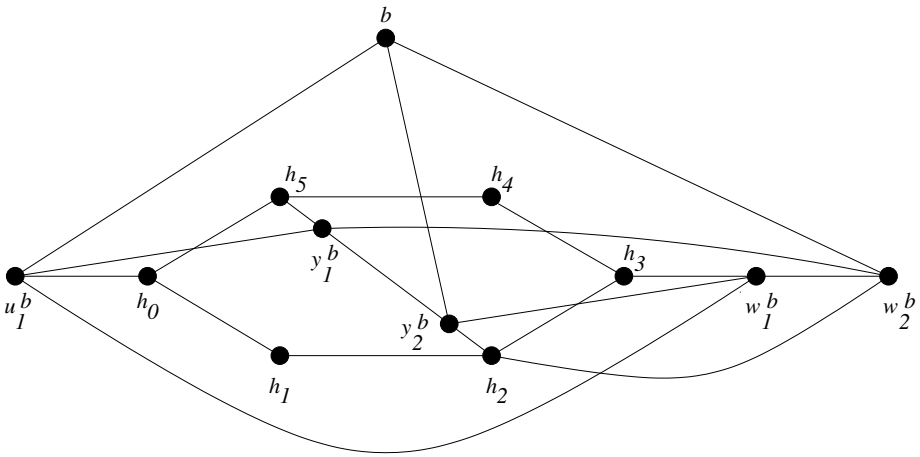


Figure 5 Construction of G' for a vertex b in $G_B - H_B$

For every vertex $a \in G_A - H_A$, we further make z_a adjacent to u_1^a and y_2^a. For every pair of vertices a and b, with $a \in G_A - H_A$, $b \in G_B - H_B$, we further also make z_{ab} adjacent to w_1^b and y_1^b.

For every vertex $b \in G_B - H_B$, we also further make x_b adjacent to u_1^b, y_2^b, and w_2^b. For every pair of vertices a and b, with $a \in G_A - H_A$, $b \in G_B - H_B$, we further make x_{ba} adjacent to w_1^a, y_1^a, and u_2^a. See Figure 6.

This completes the construction of G'. The diameter of the graph G' is four. We now prove the following two lemmas in order to prove the theorem.

▶ **Lemma 2.2.** *G retracts to H if and only if G' retracts to H.*

Proof. If G' retracts to H then it is clear that G also retracts to H, as G is a subgraph of G'. Now suppose that G retracts to H, and let $r : G \to H$ be a retraction. We define a retraction $r' : G' \to H$ as follows.

We define r' for the vertices v of G (that are also vertices of G') as
$r'(v) = r(v)$.
We define r' for the newly added vertices of G', with $a \in G_A - H_A$, as follows.

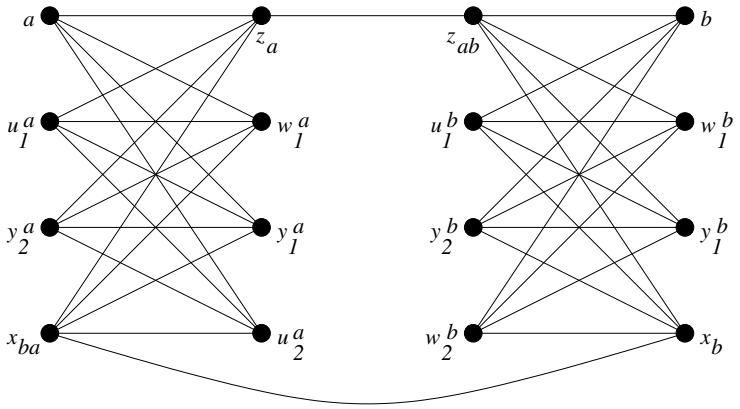


Figure 6 Construction of G' for a pair of vertices a and b, with $a \in G_A - H_A$, $b \in G_B - H_B$

If $r(a) = h_1$ or h_3, then we define
$r'(u_1^a) = h_1$, $r'(u_2^a) = h_2$,
$r'(w_1^a) = h_2$,
$r'(y_1^a) = h_2$, $r'(y_2^a) = h_3$,
$r'(z_a) = h_2$.
If $r(a) = h_5$, then we define
$r'(u_1^a) = h_5$, $r'(u_2^a) = h_0$,
$r'(w_1^a) = h_4$,
$r'(y_1^a) = h_0$, $r'(y_2^a) = h_5$,
$r'(z_a) = h_0$.
We define r' for the newly added vertices of G', with $b \in G_B - H_B$, as follows.
If $r(b) = h_0$ or h_2, then we define
$r'(u_1^b) = h_1$,
$r'(w_1^b) = h_2$, $r'(w_2^b) = h_1$,
$r'(y_1^b) = h_0$, $r'(y_2^b) = h_1$,
$r'(x_b) = h_0$.
If $r(b) = h_4$, then we define
$r'(u_1^b) = h_5$,
$r'(w_1^b) = h_4$, $r'(w_2^b) = h_3$,
$r'(y_1^b) = h_4$, $r'(y_2^b) = h_3$,
$r'(x_b) = h_4$.
We define r' for the vertices z_{ab} and x_{ba} of G', with $a \in G_A - H_A$, $b \in G_B - H_B$, as follows.
If $r(a) = h_1$ or h_3, and $r(b) = h_0$ or h_2, then we define
$r'(z_{ab}) = h_1$, $r'(x_{ba}) = h_1$.
If $r(a) = h_1$ or h_3, and $r(b) = h_4$, then we define
$r'(z_{ab}) = h_3$, $r'(x_{ba}) = h_3$.
If $r(a) = h_5$, and $r(b) = h_0$ or h_2, then we define
$r'(z_{ab}) = h_1$, $r'(x_{ba}) = h_5$.
If $r(a) = h_5$, and $r(b) = h_4$, then we define
$r'(z_{ab}) = h_5$, $r'(x_{ba}) = h_5$.

We now verify that $r' : G' \to H$ is indeed a homomorphism (and hence a retraction). We do this by considering all the edges ab of G', and proving that $r'(a)r'(b)$ is an edge of H. Before verifying, we point out that, as far as C_6 is concerned, we could use the vertices y_1^a

and y_1^b instead of the vertices z_a and x_b, respectively, but we have continued to keep z_a and x_b, as construction similar to them would be needed in the construction for larger cycles, with $a \in G_A - H_A$, $b \in G_B - H_B$.

Consider first an edge gg', with $gg' \in E(G)$. We have from our definition of r' that $r'(g) = r(g)$ and $r'(g') = r(g')$. Since $r : G \to H$ is a homomorphism, $r(g)r(g')$ must be an edge of H. Hence $r'(g)r'(g') = r(g)r(g')$ is an edge of H.

Now consider the edges $u_1^a h_0$, $u_2^a h_1$, $u_1^b h_0$, $w_1^a h_3$, $w_1^b h_3$, $w_2^b h_2$, $y_1^a h_1$, $y_2^a h_4$, $y_1^b h_5$, $y_2^b h_2$, $z_a h_1$, and $x_b h_5$, with $a \in G_A - H_A$, $b \in G_B - H_B$. From the definition of r', we have $r'(h_i) = r(h_i) = h_i$, for all $i = 0, 1, 2, 3, 4, 5$. Depending on the value of $r(a)$, we note from our definition of r' that $r'(u_1^a) = h_1$ or h_5, $r'(u_2^a) = h_2$ or h_0, $r'(w_1^a) = h_2$ or h_4, $r'(y_1^a) = h_2$ or h_0, $r'(y_2^a) = h_3$ or h_5, and $r'(z_a) = h_2$ or h_0. Hence $r'(u_1^a)r'(h_0)$, $r'(u_2^a)r'(h_1)$, $r'(w_1^a)r'(h_3)$, $r'(y_1^a)r'(h_1)$, $r'(y_2^a)r'(h_4)$, and $r'(z_a)r'(h_1)$ are always edges of H. Similarly, depending on the value of $r(b)$, from our definition of r', we have $r'(u_1^b) = h_1$ or h_5, $r'(w_1^b) = h_2$ or h_4, $r'(w_2^b) = h_1$ or h_3, $r'(y_1^b) = h_0$ or h_4, $r'(y_2^b) = h_1$ or h_3, and $r'(x_b) = h_0$ or h_4. Thus $r'(u_1^b)r'(h_0)$, $r'(w_1^b)r'(h_3)$, $r'(w_2^b)r'(h_2)$, $r'(y_1^b)r'(h_5)$, $r'(y_2^b)r'(h_2)$, and $r'(x_b)r'(h_5)$ are always edges of H.

The remaining edges of G' can also be verified. Since $r'(h) = r(h) = h$, for all $h \in V(H)$, the homomorphism $r' : G' \to H$ is a retraction. We have thus proved the lemma. ◀

▶ **Lemma 2.3.** *G' retracts to H if and only if G' vertex-compacts to H.*

Proof. If G' retracts to H then by definition G' vertex-compacts to H. Now suppose that G' vertex-compacts to H. We shall prove that G' also retracts to H. Let $c : G' \to H$ be a vertex-compaction. We let $U = \{u_1^v | v \in V(G - H)\} \cup \{h_1, h_0, h_5\}$ and $W = \{w_1^v | v \in V(G - H)\} \cup \{h_2, h_3, h_4\}$.

Since h_0 is adjacent to every other vertex in U, and G' is bipartite, the subgraph of G' induced by the vertices in U is of diameter two. Hence, the vertices of $c(U)$ induce a path of length one or two in H, as H is irreflexive. Thus $c(U)$ has either two or three vertices. Similarly, $c(W)$ has either two or three vertices. We shall prove that $c(U)$ and $c(W)$ both have three vertices.

Suppose that $c(U)$ has only two vertices. Then we know that the vertices in $c(U)$ are adjacent in H. Without loss of generality, let $c(U) = \{h_0, h_1\}$ and $c(h_0) = h_0$ (due to symmetry of vertices in H). Hence $c(U - \{h_0\}) = \{h_1\}$. We note that $d_{G'}(U - \{h_0\}, g) < 3$, for all $g \in V(G')$. Hence $d_{G'}(U - \{h_0\}, g) < d_H(c(U - \{h_0\}) = h_1, h_4) = 3$, for all $g \in V(G')$. This implies that $c(g) \neq h_4$, for all $g \in V(G')$, which is impossible, as $c : G' \to H$ is a vertex-compaction. Hence it must be that $c(U)$ has three vertices. We also note that $d_{G'}(W - \{h_3\}, g) < 3$, for all $g \in V(G')$, and hence, similarly, it must be that $c(W)$ also has three vertices.

Thus $c(U)$ and $c(W)$ both induce paths having three vertices in H. Without loss of generality, let $c(U) = \{h_1, h_0, h_5\}$ (due to symmetry). This implies that $c(h_0) = h_0$. We first prove that $c(h_3) = h_3$. We note that the diameter of G' is 4, and hence our vertex-compaction $c : G' \to H$ must also be a compaction, as otherwise the diameter of G' will be greater than 4. Let some edge gg' of G' cover the edge h_3h_4 or h_3h_2 of H under c, with $c(g) = h_3$ and $c(g') = h_4$ or h_2 (indeed there exists such an edge in G', as the vertex-compaction c is also a compaction). We note that h_3 is at distance 2 from $c(U)$ in H, as $d_H(c(U), h_3) = d_H(h_1, h_3) = 2$. Further, both h_4 and h_2 are at distance 1 from $c(U)$ in H, as $d_H(c(U), h_4) = d_H(h_5, h_4) = 1$, and $d_H(c(U), h_2) = d_H(h_1, h_2) = 1$. Thus it must be that $d_{G'}(U, g) \geq 2$ and $d_{G'}(U, g') \geq 1$. Since there is no vertex at distance more than 2 from U in G', we have $d_{G'}(U, g) = 2$ and $d_{G'}(U, g') = 1$ or 2. Further, since G' is bipartite, it must be that $d_{G'}(U, g) = 2$ and $d_{G'}(U, g') = 1$.

We note that the only vertices that could possibly be at distance 1 from U in G', as possible candidates for g', are : h_2, h_4, w_1^a, w_1^b, y_1^a, y_1^b, u_2^a, b, z_a, and x_b, with $a \in G_A - H_A$, $b \in G_B - H_B$. The only vertices that could possibly be at distance 2 from U in G', as possible candidates for g, are : h_3, y_2^a, y_2^b, w_2^b, a, z_{ab}, and x_{ba}, with $a \in G_A - H_A$, $b \in G_B - H_B$. Thus g and g' are among these vertices.

Since $c(h_0) = h_0$ and H is bipartite, $c(h_3) \neq h_4$ or h_2. Suppose that $c(h_3) \neq h_3$. Then no edge of G' with h_3 as an endpoint covers the edge h_3h_4 or h_3h_2 of H under c. Hence gg' must be an edge among $y_2^a w_1^a$, $y_2^b w_1^b$, $y_2^a y_1^a$, $y_2^b y_1^b$, $y_2^a z_a$, $y_2^a u_2^a$, $y_2^a h_4$, $y_2^b h_2$, $y_2^b b$, $y_2^b x_b$, $a y_1^a$, $a u_2^a$, $a w_1^a$, $a z_a$, $w_2^b w_1^b$, $w_2^b b$, $w_2^b x_b$, $w_2^b y_1^b$, $z_{ab} w_1^b$, $z_{ab} y_1^b$, $z_{ab} b$, $z_{ab} z_a$, $z_{ab} x_b$, $x_{ba} z_a$, $x_{ba} w_1^a$, $x_{ba} y_1^a$, $x_{ba} u_2^a$, $x_{ba} x_b$, and possibly ab, with $a \in G_A - H_A$, $b \in G_B - H_B$, where the first vertex in each of these edges stand for g and the second for g', and in order to meet the rquirements of the edge gg', the first vertex in each of these edges is assumed to achieve distance 2 from U in G' and hence may map to h_3 under c, and the second vertex in each of these edges is assumed to achieve distance 1 from U in G' and hence may map to h_4 or h_2 under c. We shall be always mentioning these edges in this order. Further, if ah_2 or ah_4 is an edge of G, for some vertex $a \in G_A - H_A$, then we need to include such an edge also for gg'. These edges for gg' are all the possible edges of G' that may cover the edge h_3h_4 or h_3h_2 of H under c assuming that $c(h_3) \neq h_3$. Since $c(h_3) \in H_A$ (as $c(h_0) = h_0 \in H_B$) and $c(h_3) \neq h_3$, we have $c(h_3) = h_1$ or h_5. The outline for proving that $c(h_3) \neq h_3$ is impossible is as follows. We suppose that $c(h_3) = h_1$, and consider each of the possible edges for gg' mentioned above, and show that they do not cover the edge h_3h_4 under c (i.e., $c(\{g, g'\}) \neq \{h_3, h_4\}$). Symmetrically, if $c(h_3) = h_5$ then it can be shown that none of the possible edges for gg' mentioned above can cover the edge h_3h_2 under c (i.e., $c(\{g, g'\}) \neq \{h_3, h_2\}$). Thus let $c(h_3) = h_1$.

Consider first the edges $y_2^a w_1^a$ and $y_2^b w_1^b$, with $a \in G_A - H_A$, $b \in G_B - H_B$. We consider them together as an edge $y_2^v w_1^v$, with $v \in V(G - H)$. Suppose that $y_2^v w_1^v$ covers the edge h_3h_4 under c. Then $c(y_2^v) = h_3$ and $c(w_1^v) = h_4$. By assumption, we have $c(h_3) = h_1$. Since w_1^v is adjacent to h_3, this implies that $c(w_1^v) = h_0$ or h_2. Thus $c(w_1^v) \neq h_4$, and we have a contradiction.

Next consider the edges aw_1^a, $w_2^b w_1^b$, and $z_{ab} w_1^b$, with $a \in G_A - H_A$, $b \in G_B - H_B$. Similar to the above, since $c(w_1^v)$ must be adjacent to $c(h_3) = h_1$, it is impossible that $c(w_1^v) = h_4$, with $v \in V(G - H)$, and hence the above edges cannot cover the edge h_3h_4 under c.

Now consider the edges $y_2^a y_1^a$ and $y_2^b y_1^b$, with $a \in G_A - H_A$, $b \in G_B - H_B$. We consider them together as an edge $y_2^v y_1^v$, with $v \in V(G - H)$. Suppose that $y_2^v y_1^v$ covers the edge h_3h_4 under c. Then $c(y_2^v) = h_3$ and $c(y_1^v) = h_4$. Since $c(u_1^v)$ must be adjacent to both $c(h_0) = h_0$ and $c(y_1^v) = h_4$, this implies that $c(u_1^v) = h_5$. Since $c(w_1^v)$ must be adjacent to both $c(u_1^v) = h_5$ and $c(y_2^v) = h_3$, it must be that $c(w_1^v) = h_4$. This implies that $y_2^v w_1^v$ covers the edge h_3h_4 under c, which we have already proved does not hold.

The remaining edges for gg' can be verified also. Symmetrically, if $c(h_3) = h_5$ then no possible edge for gg' can cover the edge h_3h_2 under c. We thus establish that $c(h_3) = h_3$, and hence $c(W) = \{h_2, h_3, h_4\}$.

We now prove that $c(h_1) \neq c(h_5)$. Suppose to the contrary that $c(h_1) = c(h_5)$. Since $c(h_0) = h_0$, we have $c(h_1), c(h_5) \in \{h_1, h_5\}$. Without loss of generality, let $c(h_1) = c(h_5) = h_1$ (due to symmetry). Since $c(U) = \{h_1, h_0, h_5\}$, it must be that $c(u_1^v) = h_5$ for some vertex v of $G - H$. Since $c(w_1^v)$, $c(h_2)$, and $c(h_4)$ must all be adjacent to $c(h_3) = h_3$, we have $c(w_1^v)$, $c(h_2)$, $c(h_4) \in \{h_2, h_4\}$. Since $c(w_1^v)$ must be adjacent to $c(u_1^v) = h_5$, it must be that $c(w_1^v) \neq h_2$, and hence $c(w_1^v) = h_4$. Since $c(h_2)$ must be adjacent to $c(h_1) = h_1$, it must be that $c(h_2) \neq h_4$, and hence $c(h_2) = h_2$. Since $c(h_4)$ must be adjacent to $c(h_5) = h_1$, it must be that $c(h_4) \neq h_4$, and hence $c(h_4) = h_2$. Now $c(y_2^a)$ must be adjacent to $c(h_4) = h_2$ and

$c(w_1^a) = h_4$, implying that $c(y_2^a) = h_3$, with $a \in G_A - H_A$. Also, $c(y_2^b)$ must be adjacent to $c(h_2) = h_2$ and $c(w_1^b) = h_4$, implying that $c(y_2^b) = h_3$ also, with $b \in G_B - H_B$. Thus we have, in general, $c(y_2^v) = h_3$. We also have that $c(y_1^a)$ must be adjacent to $c(h_1) = h_1$ and $c(u_1^a) = h_5$, implying that $c(y_1^a) = h_0$, with $a \in G_A - H_A$. Also, we have that $c(y_1^b)$ must be adjacent to $c(h_5) = h_1$ and $c(u_1^b) = h_5$, implying that $c(y_1^b) = h_0$ also, with $b \in G_B - H_B$. Thus we have, in general, $c(y_1^v) = h_0$. This is impossible as $c(y_1^v)$ must be adjacent to $c(y_2^v) = h_3$.

Thus $c(h_1) \neq c(h_5)$, i.e., $c(\{h_1, h_5\}) = \{h_1, h_5\}$. Without loss of generality, suppose that $c(h_1) = h_1$ and $c(h_5) = h_5$ (due to symmetry). Since $c(h_3) = h_3$, we have $c(h_2)$, $c(h_4) \in \{h_2, h_4\}$. Since $c(h_2)$ must be adjacent to $c(h_1) = h_1$, it must be that $c(h_2) \neq h_4$, and hence $c(h_2) = h_2$. Since $c(h_4)$ must be adjacent to $c(h_5) = h_5$, it must be that $c(h_4) \neq h_2$, and hence $c(h_4) = h_4$. We already have $c(h_0) = h_0$ and $c(h_3) = h_3$. Thus we have $c(h_i) = h_i$, for all $i = 0, 1, 2, 3, 4, 5$. Hence $c : G' \to H$ is a retraction, proving the lemma. ◀

We have thus proved Theorem 2.1. ◀

3 Vertex-Compaction to an Irreflexive k-Cycle

Our proof of Theorem 2.1 showing NP-completeness of *VCOMP-C_6* directly uses graphs that we construct simply by adding vertices and edges. Our technique of construction of graphs therefore lays down the foundation for construction of graphs for the case of a general irreflexive even k-cycle, by extending the paths constructed and adding edges appropriately, showing NP-completeness of *VCOMP-C_k*, for all even $k \geq 6$.

In [Vikas, 1999, 2004a], it is shown that the problem *COMP-C_k* is NP-complete, for all even $k \geq 6$. The problem *RET-C_k* is shown to be NP-complete, for all even $k \geq 6$, in [Feder, Hell, and Huang, 1999], and independently by G. Macgillivray in 1988. To prove NP-completeness of *VCOMP-C_k*, we give a transformation from *RET-C_k* to *VCOMP-C_k*, for all even $k \geq 6$. In our construction to prove NP-completeness of *VCOMP-C_k*, with even $k \geq 6$, we now have for example paths Z_{ab}, X_{ba}, U_a, U_b, W_a, W_b, Y_a, and Y_b of appropriate lengths instead of the vertices z_{ab}, x_{ba}, u_1^a, u_2^a, u_1^b, w_1^a, w_1^b, and w_2^b that we used in the construction in the proof of Theorem 2.1 for proving NP-completeness of *VCOMP-C_6*, and we add edges chosen appropriately.

Considering $k = 4$, it is easy to see that the problems *VCOMP-C_4*, *COMP-C_4*, and *RET-C_4* are polynomial time solvable, as C_4 is a complete bipartite graph. We now consider odd $k \geq 3$. Note that a graph G is homomorphic to a graph H if and only if the disjoint union $G \cup H$ vertex-compacts, compacts, and retracts to H. Thus we have a polynomial transformation from the *H-colouring* problem to the problems *VCOMP-H*, *COMP-H*, and *RET-H*. The *H-colouring* problem is shown to be NP-complete for any fixed non-bipartite irreflexive graph H in [Hell & Nesetril 1990]. Hence, it follows that the problems *VCOMP-H*, *COMP-H*, and *RET-H* are also NP-complete when H is any non-bipartite irreflexive graph. Thus, in particular, the problems *VCOMP-C_k*, *COMP-C_k*, and *RET-C_k* are NP-complete, for all odd $k \geq 3$.

References

1 H. J. Bandelt, A. Dahlmann, and H. Schutte, Absolute Retracts of Bipartite Graphs, Discrete Applied Mathematics, 16, 191-215, 1987.

2 H. J. Bandelt, M. Farber, and P. Hell, Absolute Reflexive Retracts and Absolute Bipartite Retracts, Discrete Applied Mathematics, 44, 9-20, 1993.

3 T. Feder and P. Hell, List Homomorphisms to Reflexive Graphs, Journal of Combinatorial Theory, Series B, 72, 236-250, 1998.

4 T. Feder, P. Hell, and J. Huang, List Homomorphisms and Circular Arc Graphs, Combinatorica, 19, 487-505, 1999.

5 T. Feder, P. Hell, S. Klein, and R. Motwani, Complexity of Graph Partition Problems, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), Atlanta, Georgia, 1999.

6 T. Feder, P. Hell, S. Klein, and R. Motwani, List Partitions, SIAM Journal on Discrete Mathematics, 16, 449-478, 2003.

7 T. Feder and M. Y. Vardi (1993), Monotone Monadic SNP and Constraint Satisfaction, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC), San Diego, California.

8 T. Feder and M. Y. 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.

9 F. Harary, Graph Theory, Addison-Wesley Publishing Company, Reading, Massachusetts, 1969.

10 P. Hell, Retractions de Graphes, Ph.D. Thesis, Universite de Montreal, 1972.

11 P. Hell, Retracts in Graphs, in Graphs and Combinatorics, Lecture Notes in Mathematics, Springer-Verlag, 406, 291-301, 1974.

12 P. Hell, Graph Partitions with Prescribed Patterns, European Journal of Combinatorics, 35, 335-353, 2014.

13 P. Hell and D. J. Miller, Graphs with Forbidden Homomorphic Images, Annals of the New York Academy of Sciences, 319, 270-280, 1979.

14 P. Hell and J. Nesetril, On the Complexity of H-colouring, Journal of Combinatorial Theory, Series B, 48, 92-110, 1990.

15 P. Hell and I. Rival, Absolute Retracts and Varieties of Reflexive Graphs, Canadian Journal of Mathematics, 39, 544-567, 1987.

16 R. Nowakowski and I. Rival, Fixed-Edge Theorem for Graphs with Loops, Journal of Graph Theory, 3, 339-350, 1979.

17 E. Pesch, Retracts of Graphs, Mathematical Systems in Economics, 110, Frankfurt am Main : Athenaum, 1988.

18 E. Pesch and W. Poguntke, A Characterization of Absolute Retracts of n-Chromatic Graphs, Discrete Mathematics, 57, 99-104, 1985.

19 N. Vikas, Computational Complexity of Compaction to Cycles, Proceedings, Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999.

20 N. Vikas (2002), Connected and Loosely Connected List Homomorphisms, 28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), Cesky Krumlov, Czech Republic, Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany, 2573, 399-412.

21 N. Vikas, Computational Complexity of Compaction to Reflexive Cycles, SIAM Journal on Computing, 32, 253-280, 2003.

22 N. Vikas, Computational Complexity of Compaction to Irreflexive Cycles, Journal of Computer and System Sciences, 68, 473-496, 2004a.

23 N. Vikas, Compaction, Retraction, and Constraint Satisfaction, SIAM Journal on Computing, 33, 761-782, 2004b.

24 N. Vikas, Computational Complexity Classification of Partition under Compaction and Retraction, Tenth Annual International Computing and Combinatorics Conference (COCOON), Jeju Island, Korea, Lecture Notes in Computer Science, 3106, Springer-Verlag, Heidelberg, Germany, 380-391, 2004c.

25 N. Vikas, A Complete and Equal Computational Complexity Classification of Compaction and Retraction to All Graphs with at most Four Vertices, Journal of Computer and System Sciences, 71, 406-439, 2005.

26 N. Vikas, Algorithms for Partition of Some Class of Graphs under Compaction, Seventeenth Annual International Computing and Combinatorics Conference (COCOON), Texas, U.S.A., Lecture Notes in Computer Science, 6842, Springer-Verlag, Heidelberg, Germany, 319-330, 2011.

27 N. Vikas, Algorithms for Partition of Some Class of Graphs under Compaction and Vertex-Compaction, Invited Paper, Algorithmica, 67, 180-206, 2013.

Recognizing Graphs Close to Bipartite Graphs*

Marthe Bonamy[1], Konrad K. Dabrowski[2], Carl Feghali[3], Matthew Johnson[4], and Daniël Paulusma[5]

1 **CNRS, LaBRI, Université de Bordeaux, Bordeaux, France**
marthe.bonamy@u-bordeaux.fr
2 **Durham University, Durham, UK**
konrad.dabrowski@durham.ac.uk
3 **IRIF & Université Paris Diderot, Paris, France**
feghali@irif.fr
4 **Durham University, Durham, UK**
matthew.johnson2@durham.ac.uk
5 **Durham University, Durham, UK**
daniel.paulusma@durham.ac.uk

Abstract

We continue research into a well-studied family of problems that ask if the vertices of a graph can be partitioned into sets A and B, where A is an independent set and B induces a graph from some specified graph class $\mathcal{G}$. We let $\mathcal{G}$ be the class of k-degenerate graphs. The problem is known to be polynomial-time solvable if $k = 0$ (bipartite graphs) and NP-complete if $k = 1$ (near-bipartite graphs) even for graphs of diameter 4, as shown by Yang and Yuan, who also proved polynomial-time solvability for graphs of diameter 2. We show that recognizing near-bipartite graphs of diameter 3 is NP-complete resolving their open problem. To answer another open problem, we consider graphs of maximum degree Δ on n vertices. We show how to find A and B in $O(n)$ time for $k = 1$ and $\Delta = 3$, and in $O(n^2)$ time for $k \geq 2$ and $\Delta \geq 4$. These results also provide an algorithmic version of a result of Catlin [1979] and enable us to complete the complexity classification of another problem: finding a path in the vertex colouring reconfiguration graph between two given k-colourings of a graph of bounded maximum degree.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases degenerate graphs, near-bipartite graphs, reconfiguration graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.70

1 Introduction

We consider the problem of partitioning the vertex set of a graph into two sets A and B where A is an independent set and B belongs to a specified graph class $\mathcal{G}$. The classes $\mathcal{G}$ we consider are those of k-degenerate graphs (a graph is k-degenerate if every induced subgraph has a vertex of degree at most k). A 0-degenerate graph has no edges, so in this case the problem is simply that of recognizing bipartite graphs. A graph is 1-degenerate if and only if it is a forest; we say that a graph is *near-bipartite* if it can be decomposed into an independent set and a forest. The problem of deciding whether or not a graph is near-bipartite is NP-complete (see [6]). We consider this problem when the input is restricted, but our main focus is the problem of *finding* the decomposition into an independent set and a

* This paper received support from EPSRC (EP/K025090/1), London Mathematical Society (41536), the Leverhulme Trust (RPG-2016-258) and Fondation Sciences Mathématiques de Paris.

k-degenerate graph for graphs of bounded maximum degree. We also describe an application to a graph reconfiguration problem.

Related Work. As noted above, recognizing near-bipartite graphs is known to be an NP-complete problem. Yang and Yuan [29] proved that the problem remains NP-complete even for graphs of maximum degree 4 and for graphs of diameter 4. They also showed that it can be solved in polynomial time for graphs of maximum degree at most 3 and for graphs of diameter at most 2. They left open the problem of determining its computational complexity for graphs of diameter 3 (see also [4]). We note that the problem of recognizing near-bipartite graphs has also been studied from the perspective of parameterized complexity (in this setting the problem is known as Independent Feedback Vertex Set). With the size of A as the parameter, FPT algorithms have been found for general graphs [26]. The problem of finding a decomposition where the size of A is minimum has been shown to be NP-hard even for planar bipartite graphs of maximum degree 4, but linear-time solvable for graphs of bounded treewidth, chordal graphs and cographs [27] (it was already known that near-bipartite cographs can be recognized in linear time [4]). Recently, we extended the result of cographs to P_5-free graphs, and we also proved that recognizing near-bipartite line graphs of maximum degree 4 is NP-complete [2]. The variant where the maximum degree of the forest induced by B is bounded by some constant has also been studied, in particular from a structural point of view (see, for instance, [12]).

The Induced Forest 2-Partition problem is closely related to the problem of recognizing near-bipartite graphs. It asks whether the vertex set of a given graph can be decomposed into two disjoint sets A and B, where both A and B induce forests. Wu, Yuan and Zhao [28] proved that Induced Forest 2-Partition is NP-complete for graphs of maximum degree 5 and polynomial-time solvable for graphs of maximum degree at most 4. Brandstädt et al. [6] proved NP-completeness of another closely related problem, namely that of deciding whether the vertex set of a given graph can be decomposed into an independent set and a tree. Note that both this problem and that of recognizing near-bipartite graphs can be seen as restricted variants of the classical 3-Colouring problem,[1] which is well known to be NP-complete [20]. In fact, for a fixed graph class $\mathcal{G}$ (that is, $\mathcal{G}$ is not part of the input), Brandstädt et al. [6] considered the following more general problem:

Stable($\mathcal{G}$)	
Instance:	A graph $G = (V, E)$.
Question:	Can V be decomposed into two disjoint sets A and B, where A is an independent set and B induces a graph in $\mathcal{G}$?

Note that Stable($\mathcal{G}$) is equivalent to 3-Colouring if we choose $\mathcal{G}$ to be the class of bipartite graphs. For a positive integer k, we denote the class of k-degenerate graphs by $\mathcal{D}_k$. Then Stable($\mathcal{D}_1$) is the problem of finding near-bipartite graphs. If $\mathcal{G}$ is the class of complete graphs, Stable($\mathcal{G}$) is the problem of recognizing split graphs, which can be solved in polynomial time. Brandstädt et al. [6] proved that Stable($\mathcal{G}$) is NP-complete when $\mathcal{G}$ is the class of trees or the class of trivially perfect graphs, and polynomial-time solvable when $\mathcal{G}$ is the class of co-bipartite graphs, the class of split graphs, or the class of threshold graphs. Moreover, Stable($\mathcal{G}$) has also been shown to be NP-complete when $\mathcal{G}$ is

[1] For a fixed integer $k \geq 1$, the k-Colouring problem asks whether a given graph is *k-colourable*, that is, whether its vertices can be coloured with at most k colours such that no two adjacent vertices are coloured alike. As trees and forests are 2-colourable, the observation follows.

the class of triangle-free graphs [7], the class of cographs [17], the class of graphs of maximum degree 1 [22], or, more generally, the class of graphs that have any additive hereditary property not equal to or divisible by the property of being edgeless [19], whereas it is also polynomial-time solvable if $\mathcal{G}$ is the class of complete bipartite graphs [13] (see [5] for a faster algorithm). The Stable($\mathcal{G}$) problem has also been studied for hereditary graph classes $\mathcal{G}$ with subfactorial or factorial speed [11, 21] (the speed of a graph class is the function that given an integer n returns the number of labelled graphs on n vertices in the class).

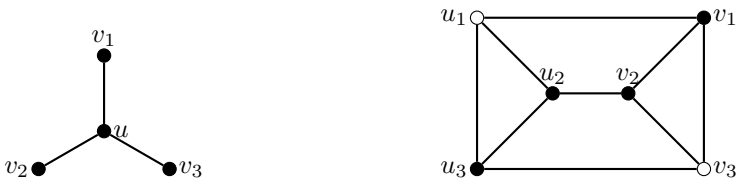
To obtain a more general problem than Stable($\mathcal{G}$), we can relax the condition on the set A being independent. This leads to the $(\mathcal{G}_1, \mathcal{G}_2)$-Recognition problem, which asks whether the vertex set of a graph can be decomposed into disjoint sets A and B, such that A induces a graph in $\mathcal{G}_1$ and B induces a graph in $\mathcal{G}_2$. For instance, if $\mathcal{G}_1$ is the class of cliques and $\mathcal{G}_2$ is the class of disjoint unions of cliques, the $(\mathcal{G}_1, \mathcal{G}_2$-Recognition problem is equivalent to recognizing unipolar graphs (see [24] for a quadratic algorithm). Generalizing Stable($\mathcal{G}$) also leads to a family of transversal problems, such as Feedback Vertex Set. However, such problems are beyond the scope of our paper.

As Stable($\mathcal{D}_k$) is NP-complete for every $k \geq 1$, its complexity has been determined for special graph classes. Recall the aforementioned hardness and tractability results of Yang and Yuan [29] on graph classes of bounded maximum degree and classes of bounded diameter when $k = 1$. Brandstädt, Brito, Klein, Nogueira and Protti [4] proved that Stable($\mathcal{D}_1$) is NP-complete on perfect graphs and, as stated earlier, that it is polynomial-time solvable for cographs. In particular, their hardness result for perfect graphs shows that the complexities of Stable($\mathcal{D}_1$) and 3-Colouring do not coincide, as k-Colouring is polynomial-time solvable for perfect graphs even when the number of colours k is part of the input [16].

Our Results. In Section 2, we consider the Stable($\mathcal{D}_1$) problem for graphs of bounded diameter. Recall that Yang and Yuan [29] left one missing case: they proved that recognizing near-bipartite graphs of diameter 4 is an NP-complete problem and that near-bipartite graphs of diameter 2 can be recognized in polynomial time. We show that Stable($\mathcal{D}_1$) is NP-complete for graphs of diameter 3, which resolves their open problem.

In Section 3, we consider near-bipartite decompositions of *subcubic* graphs (that is, graphs of maximum degree at most 3). By a result of Catlin and Lai [9], the only connected subcubic graph that is not near-bipartite is K_4 (see also Yang and Yuan [29]) and so near-bipartite subcubic graphs can be recognized in polynomial time. However, the proofs of [9, 29] do not lead to a linear-time algorithm for finding the desired partition (A, B); in particular, the result of [9] would require solving an NP-complete problem, namely Independent Set for cubic graphs [15]. We give a linear-time algorithm that finds such a partition.

We say a partition (A, B) of the vertex set of a graph is a *k-degenerate decomposition* if A is independent and B induces a $(k-2)$-degenerate graph (so Section 3 is concerned with 3-degenerate decompositions of graphs of maximum degree 3). In Section 4, we consider, more generally, Δ-degenerate decompositions of graphs of maximum degree at most Δ for any $\Delta \geq 3$. We give a quadratic-time algorithm to find such a decomposition in this general case (contrast with the linear-time algorithm for the special case of $\Delta = 3$). By a result of Matamala [23], the only connected graph with maximum degree at most Δ that does not have such a decomposition is $K_{\Delta+1}$. Matamala's result generalizes that of [9] and thus does not imply a polynomial-time algorithm for finding such a decomposition. We show that our algorithms for finding Δ-degenerate decompositions of graphs of maximum degree at most Δ also provide an algorithmic version of a result of Catlin [8]. In Section 5, we show how they can be applied to completely settle the complexity classification of a graph colouring reconfiguration problem considered in [14]. In Section 6 we give directions for future work.



(a) claw **(b)** triangular prism

Figure 1 The claw and the triangular prism. A near-bipartite decomposition of the triangular prism is indicated: the white vertices form an independent set and the black vertices induce a forest.

2 Graphs of Diameter 3

We present a hardness result (proof omitted) for the problem of deciding whether a graph of diameter at most 3 has a near-bipartite decomposition. In [25], Mertzios and Spirakis proved a remarkable result: 3-Colouring is NP-complete for graphs of diameter 3. The outline of their proof is straightforward: a reduction from 3-Sat that constructs, for any instance ϕ, a graph H_ϕ that is 3-colourable if and only if ϕ is satisfiable. It is possible to reduce 3-Sat to Stable($\mathcal{D}_1$) for graphs of diameter 3 using the same construction, thus showing that Stable($\mathcal{D}_1$) is also NP-complete on this class. That is H_ϕ is near-bipartite if and only if ϕ is satisfiable. So Theorem 1 below is an observation about the proof of Mertzios and Spirakis.

▶ **Theorem 1.** Stable($\mathcal{D}_1$) *is* NP*-complete for graphs of diameter at most* 3*.*

3 Subcubic Graphs

In this section we prove the following result.

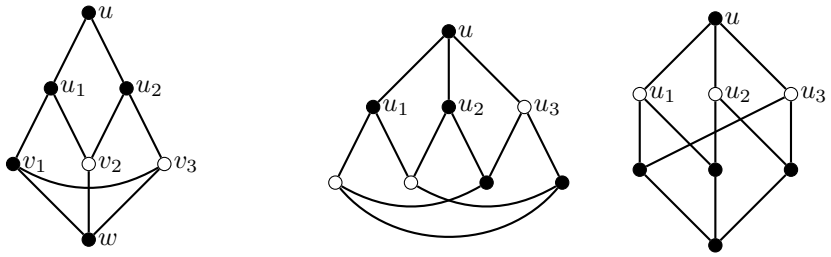
▶ **Theorem 2.** *Let G be a subcubic graph on n vertices, with no component isomorphic to K_4. Then a near-bipartite decomposition of G can be found in $O(n)$ time.*

Proof. We will repeatedly apply a set of *rules* to G. Each rule takes constant time to apply and after each application of a rule, the resulting graph contains fewer vertices. The rules are applied until the empty graph is obtained. We then reconstruct G from the empty graph by working through the rules applied in reverse order. As we rebuild G in this way, we find a near-bipartite decomposition of each obtained graph. We do this by describing how to extend, in constant time, a near-bipartite decomposition of a graph before some rule is undone to a near-bipartite decomposition of the resulting graph after that rule is undone. If we can do this then we say that the rule is *safe*. We conclude that the total running time of the algorithm is $O(n)$. It only remains to describe the rules, show that it takes constant time to do and undo each of them and prove that they are safe.

We need the following terminology. The *claw* is the graph with vertices u, v_1, v_2, v_3 and edges uv_1, uv_2, uv_3; the vertex u is the *centre* of the claw (see Figure 1). The *triangular prism* is the graph obtained from two triangles on vertices u_1, u_2, u_3 and v_1, v_2, v_3, respectively, by adding the edges u_iv_i for $i \in \{1, 2, 3\}$ (see Figure 1). Two vertices are *false twins* if they have the same neighbourhood (note that such vertices must be non-adjacent).

Let u be an arbitrary vertex of G. Our choice of u as an arbitrary vertex implies that u can be found in constant time. We then use the first of the following rules that is applicable.

Rule 1. If u has degree at most 2, then remove u.



(a) H_1 **(b)** H_2 **(c)** H_3

Figure 2 The graphs used in Rule 8. A near-bipartite decomposition of each is indicated: the white vertices form an independent set and the black vertices induce a forest.

Rule 2. If there is a vertex v of degree at most 2 that is at distance at most 3 from u, then remove v.

Rule 3. If G contains an induced diamond D whose vertices are at distance at most 3 from u, then remove the vertices of D.

Rule 4. If there is a pair of false twins u_1, u_2 each at distance at most 2 from u, then remove u_1, u_2 and their common neighbours (note that $u \in \{u_1, u_2\}$ is possible).

Rule 5. If u is in a connected component that is a triangular prism P, then remove the vertices of P.

Rule 6. If Rules 1–5 do not apply but u is in a triangle T, then the neighbours of the vertices in T that are outside T are pairwise distinct (since there is no induced diamond) and at least two them, which we denote by x', y', are non-adjacent (otherwise u belongs to a triangular prism). Remove the vertices of T and add an edge between x' and y'.

Rule 7. If u is the centre of an induced claw but has a neighbour v that belongs to a triangle, then apply one of the Rules 1–6 on v.

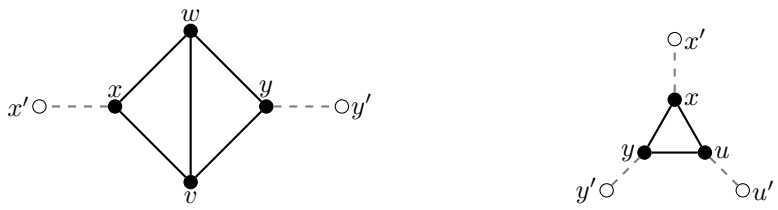
Rule 8. If the graph induced by the vertices at distance at most 3 from u contains the graph H_1, H_2 or H_3, depicted in Figure 2, with the vertex u in the position shown in the figure, then remove the vertices of this graph H_i.

Rule 9. If Rules 1–8 do not apply but u is the centre of an induced claw and its three neighbours u_1, u_2, u_3 are also centres of induced claws, then remove u, u_1, u_2, u_3 and for $i \in \{1, 2, 3\}$ add an edge joining the two neighbours of u_i distinct from u and denote it by e_i; we say that such an edge is *new* (note that such neighbours of two distinct u_i and u_j may overlap).

Let us show that at least one of the rules is always applicable. Suppose that, on the contrary, there is a vertex u of a subcubic graph for which no rule applies. Then u and its neighbours each have degree 3 (Rules 1 and 2) and so each either belongs to a triangle or is the centre of an induced claw. By Rule 6, u must be the centre of an induced claw and therefore, by Rule 7, the same is also true for each neighbour of u. This implies that Rule 9 applies, a contradiction.

Because G is subcubic, each of these rules takes constant time to verify and process. In particular, in some rules we need to detect some induced subgraph of constant size that contains u or replace u by some other vertex v. In all such cases we need to explore a set of vertices of distance at most 4 from u. As G is subcubic, this set has size at most $1 + 3 + 3^2 + 3^3 + 3^4 = 121$, so we can indeed do this in constant time.

It is clear that, as claimed, the application of a rule reduces the number of vertices and that if we repeatedly choose an arbitrary vertex u and apply a rule, we eventually obtain the empty



(a) diamond **(b)** triangle

Figure 3 The diamond and triangle (solid edges and vertices) together with their neighbourhoods in a cubic graph.

graph. We now consider undoing the applied rules in reverse order to rebuild G. As this is done, we will irrevocably *colour* vertices with colour 1 or 2 in such a way that the vertices coloured 1 will form an independent set and the vertices coloured 2 will induce a forest. Thus a rule is safe if this colouring can be extended whenever that rule is undone. When we reach G, the final colouring will correspond to the required near-bipartite decomposition.

We must prove each rule is safe. At each step of reconstructing G, we refer to the graph before a rule is undone as the *prior graph* and to the graph after that rule is undone as the *subsequent graph*. Note that the application of any of the Rules 1–9 again yields a subcubic graph. By the result of Yang and Yuan [29], every connected subcubic graph is near-bipartite, apart from K_4. So we need to ensure that an application of a rule does not create a K_4. This cannot happen when we remove vertices, but we will need to consider it for Rules 6 and 9.

▶ **Claim 3.** *Rules 1–5 are safe.*

In Rules 1–5 we only delete vertices. Rule 1 is safe since if both neighbours of u are coloured 2, then u can be coloured 1; otherwise u can be coloured 2. Similarly, we see that Rule 2 is safe. To see that Rule 3 is safe, let D be the diamond with vertex labels as illustrated in Figure 3, where u is one of v, w, x, y. If x' and y' are coloured 2, we colour x and y with 1 and v and w with 2. Otherwise we colour v with 1 and x, y and w with 2. We now show that Rule 4 is safe. Let u_1 and u_2 be false twins (at distance at most 2 from u). As G is subcubic, every vertex in $N(u_1)$ with a neighbour in $N(u_1)$ has no neighbours outside $N(u_1) \cup \{u_1, u_2\}$ and every vertex in $N(u_1)$ with no neighbour in $N(u_1)$ has at most one neighbour not equal to u_1 or u_2. Moreover, as G is subcubic, $N(u_1)$ contains no cycle. Hence we can always colour u_1, u_2 with 1 and the vertices of $N(u_1)$ with 2 regardless of the colours of vertices outside $N(u_1) \cup \{u_1, u_2\}$. Indeed, every vertex of $N(u_1)$ will have at most one neighbour that is not coloured 1, so cannot be in a cycle of vertices coloured 2 in the subsequent graph. Rule 5 is also safe since P is 3-regular and hence would be a component of our subsequent graph, so we can colour its vertices by assigning colour 1 to exactly one vertex from each of the two triangles and colour 2 to its other vertices (see Figure 1). This completes the proof of Claim 3.

▶ **Claim 4.** *Rules 6 and 7 are safe.*

First, let us demonstrate that Rule 6 is safe. If x' and y' are contained in a K_4 of the prior graph, then the subsequent graph contains a diamond whose vertices are at distance at most 3 from u. This contradicts Rule 3. Let T be the triangle with vertex labels as illustrated in Figure 3. Suppose x', y' and u' are coloured 2. Then we colour u with 1 and x and y with 2. The vertices in the subsequent graph with colour 2 still induce a forest, as we have replaced an edge in the forest by a path on four vertices. Suppose x' and y' are coloured 2 and u' is

coloured 1. Then we colour x with 1, and y and u with 2. Otherwise, since x' and y' are joined by an edge in the prior graph, we may assume that x' has colour 1 and y' has colour 2. In this case we can colour y with 1, and x and u with 2. This completes the proof that Rule 6 is safe. Since Rules 1–6 are safe, it follows that Rule 7 is also safe. This completes the proof of Claim 4.

▶ **Claim 5.** *Rule 8 is safe.*

We now show that Rule 8 is safe. Suppose u is contained in H_1. We use the vertex labels from Figure 2. As Rule 1 could not be applied, we find that u has a third neighbour u_3 distinct from u_1 and u_2. Regardless of whether u_3 is coloured 1 or 2, we colour u, u_1, u_2, v_1, w with 2 and v_2, v_3 with 1 to obtain a near-bipartite decomposition of G. We can also readily colour the vertices of H_2 or H_3 should u be contained in one of them (note that since H_2 and H_3 are 3-regular, these graphs can only appear as components in our subsequent graph). This completes the proof of Claim 5.

▶ **Claim 6.** *Rule 9 is safe.*

Suppose that the prior graph contains fewer than three new edges. Then we may assume without loss of generality that $e_1 = e_2$. Then u_1 and u_2 are false twins at distance 1 from u and we can apply Rule 4, a contradiction. So we may assume that the prior graph contains exactly three new edges.

We claim that the application of Rule 9 does not yield a K_4. For contradiction, suppose it does. Let K be the created K_4. Then at least one new edge is contained in K. If exactly one new edge e is contained in K, then $K - e$ is a diamond in the subsequent graph. Then we could have applied Rule 3, a contradiction. If all three new edges are in K, then they must induce either a path on four vertices or a triangle in the subsequent graph. In the first case the subsequent graph is H_2 and in the second case the subsequent graph is H_3. In both cases we would have applied Rule 8, a contradiction. Finally, suppose that K contains exactly two new edges, say e_1 and e_2. If e_1 and e_2 do not share a vertex, then they cover the vertices of K. Hence the end-vertices of e_1 are false twins (at distance 2 from u) in the subsequent graph, since they are both adjacent to u_1 and to each end-vertex of e_2. Then we could have applied Rule 4, a contradiction. If $e_1 = v_1v_2$ and $e_2 = v_3v_4$ share a vertex, say $v_2 = v_4$, then v_1 and v_3 are adjacent in the subsequent graph and the vertex $w \in K \setminus \{v_1, v_2, v_3\}$ is adjacent only to v_1, v_2 and v_3. Therefore Rule 8 could have been applied, a contradiction.

Thus an application of Rule 9 does not yield a K_4, and we may colour u_1, u_2, u_3 with 2 and u with 1. Indeed note that since if two end-vertices of a new edge are coloured 2, then in the subsequent graph the vertices coloured 2 will still induce a forest, in which such a new edge is replaced by a path of length 2. This completes the proof of Claim 6 and therefore completes the proof of Theorem 2. ◀

4 Graphs of Bounded Maximum Degree

Let $k \geq 3$ be an integer. Recall that a graph G has a k-degenerate decomposition if its vertex set can be decomposed into sets A and B where A is an independent set and B induces a $(k-2)$-degenerate graph. Note that 3-degenerate decompositions are near-bipartite decompositions. We give an $O(n^2)$ algorithm for finding a Δ-degenerate decomposition of a graph on n vertices of maximum degree at most Δ for every $\Delta \geq 3$ (note that for $\Delta = 3$ we can also use Theorem 2).

For $k \geq 1$, we say that an order $v_1, v_2, \ldots, v_n$ of the vertices of a graph G is *k-degenerate* if for all $i \geq 2$, the vertex v_i has at most k neighbours in $\{v_1, \ldots, v_{i-1}\}$. It is clear that a

graph is k-degenerate if and only if it has a k-degenerate order. If $\mathcal{O}$ is a k-degenerate order for G and W is a subset of the vertex set of G, then we let $\mathcal{O}|_W$ be the restriction of $\mathcal{O}$ to W, and let $G[W]$ be the subgraph of G induced by W. For a set of vertices C, we denote the *neighbourhood* of C by $N(C) = \bigcup\{N(u) \mid u \in C\} \setminus C$.

We need the following lemma, which is a refinement of Lemma 8 in [14] with the same proof.

▶ **Lemma 7.** *Let $k \geq 2$. Let G be a $(k-1)$-degenerate graph on n vertices. If a $(k-1)$-degenerate order $\mathcal{O}$ of G is given as input, a k-degenerate decomposition (A, B) of G can be found in $O(kn)$ time. In addition, we can ensure that $\mathcal{O}|_B$ is a $(k-2)$-degenerate order of $G[B]$, the set A is a maximal independent set and the first vertex in $\mathcal{O}$ belongs to A.*

A pair of non-adjacent vertices $\{u, v\}$ in a graph G is *strong* if u and v have a common neighbour in each component of the graph $G \setminus \{u, v\}$. In particular, note that if u and v have a common neighbour and $G \setminus \{u, v\}$ is connected, then $\{u, v\}$ is a strong pair. We need the following two lemmas (we omit the proof of the first one).

▶ **Lemma 8.** *Let $k \geq 3$. Let G be a connected k-regular graph on n vertices that contains a strong pair $\{u, v\}$. If $\{u, v\}$ is given as input, a k-degenerate decomposition of G can be found in $O(kn)$ time.*

▶ **Lemma 9.** *Let $k \geq 3$. Let G be a k-regular connected graph on n vertices, which contains a set C of $k+1$ vertices that induce a clique minus an edge uv. If C, u and v are given as input, then a k-degenerate decomposition of G can be found in $O(kn)$ time.*

Proof. Let x be a vertex in C distinct from u and v. Let G' be the graph obtained from G by deleting C. Each of u and v has exactly one neighbour that does not belong to C and all other vertices of C have no neighbours outside C. Let t be the neighbour of u not in C, and let w be the neighbour of v not in C. We may assume that t is distinct from w, otherwise we are done by Lemma 8. We can find a $(k-1)$-degenerate order $\mathcal{O}$ of G' in $O(kn)$ time by taking the vertices in the reverse of the order they are found in a breadth-first search from t, and then, if t and w do not belong to the same component of G', appending the vertices in the reverse of the order they are found in a breadth-first search from w. By Lemma 7, we can compute a k-degenerate decomposition (A, B) of G' in $O(kn)$ time such that $\mathcal{O}|_B$ is a $(k-2)$-degenerate order of B. If both t and w belong to B, let $A' = A \cup \{u, v\}$ and $B' = B \cup (C \setminus \{u, v\})$ and, since $(C \setminus \{u, v\})$ is a clique on $k-1$ vertices with no edge joining it to B, if follows that (A', B') is a k-degenerate decomposition of G. Assume now without loss of generality that $t \in A$ (we make no assumption about whether w is also in A). Then let $A' = A \cup \{x\}$ and $B' = B \cup (C \setminus \{x\})$. Then A' is an independent set. Recall that $\mathcal{O}|_B$ is a $(k-2)$-degenerate order of B. We show that we can append the vertices of $C \setminus \{x\}$ to obtain a $(k-2)$-degenerate order of B'. First add v, then the vertices of $C \setminus \{u, v, x\}$ and finally u. It is clear that no vertex has more than $k-2$ neighbours earlier in the order. ◀

▶ **Lemma 10.** *Let $k \geq 3$. Let G be a k-regular connected graph on n vertices containing a clique C on k vertices whose neighbourhood is of size 2. If C is given as input, a k-degenerate decomposition of G can be found in $O(kn)$ time.*

Proof. Let u and v be vertices not in C such that for each vertex in C, its unique neighbour not in C is either u or v. Neither u nor v can be adjacent to every vertex in C (as then the other would be adjacent to none, contradicting the premise that the neighbourhood has size 2). Since $k \geq 3$, one of u and v has at least two neighbours in C. Consider the

graph G' obtained from G by removing C and adding the edge uv (if it does not already exist). Note that u and v each have degree at most k in G' and at least one of them, say u, has degree less than k. Therefore, we can find a $(k-1)$-degenerate order $\mathcal{O}$ of G' in $O(kn)$ time by taking the vertices in the order they are found in a breadth-first search from u. Thus we can obtain a k-degenerate decomposition (A, B) of G' by Lemma 7, such that $\mathcal{O}|_B$ is a $(k-2)$-degenerate order of $G[B]$ and A is a maximal independent set. At least one of u and v must belong to B. Assume without loss of generality that either $v \in A$, $u \in B$ or both u and v belong to B, and, in the latter case, assume that u has at least two neighbours in C. Consider a neighbour t of u in C. We set $A' = A \cup \{t\}$ and $B' = B \cup (C \setminus \{t\})$, and claim that (A', B') is a k-degenerate decomposition of G. It is clear that A' is an independent set. Recall that $\mathcal{O}|_B$ is a $(k-2)$-degenerate order for $G[B]$. We must amend it to find a $(k-2)$-degenerate order for $G[B']$ that also includes the vertices of $C \setminus \{t\}$. We consider two cases.

First suppose $v \in A$. Then append to $\mathcal{O}|_B$ first the neighbours of u in $C \setminus \{t\}$ and then the neighbours of v. As the vertices of $C \setminus \{t\}$ are adjacent to t the only one that could have more than $k-2$ vertices before it in the order is the one that appears last, but this is also adjacent to v so we do indeed have a $(k-2)$-degenerate order.

Now suppose $v \in B$. Then u has a neighbour in G' that belongs to A (as A is a maximal independent set). Hence u has at most $k-2$ neighbours in B'. Append to $\mathcal{O}|_B$ the vertices of $C \setminus \{t\}$, ending with a neighbour of u (we know there is at least one), then move u to be the final vertex in the order. Again the only vertex of $C \setminus \{t\}$ that could have more than $k-2$ neighbours before it in the order is the one that appears last, and by choosing it to be a neighbour of u and putting u later in the order we ensure that a $(k-2)$-degenerate order is obtained. ◀

Given a graph G, five of its vertices t, u, v, w, x and a set of vertices C, we say that C induces a (u, v)-*lock with special vertices* $(t, \{w, x\})$ if $t, w, x \in C$ and $N(C) = \{u, v\}$, and both u and v are adjacent to t, each vertex in $\{w, x\}$ is adjacent to precisely one vertex in $\{u, v\}$, and $G[C]$ contains all possible edges except for wt and xt. We say that C is a *lock* if it is a (u, v)-*lock with special vertices* $(t, \{w, x\})$ for some choice of t, u, v, w, x.

▶ **Lemma 11.** *Let $k \geq 3$. Let G be a k-regular connected graph on n vertices containing a (u, v)-lock C with special vertices $(t, \{w, x\})$. If C and u, v, t, w, x are given as input, then a k-degenerate decomposition of G can be found in $O(kn)$ time.*

Proof. Since t has two neighbours outside C, it follows that $C \setminus \{t\}$ is a clique on k vertices. If w and x have the same neighbour in $\{u, v\}$, say u, then $N(C \setminus \{t\}) = \{t, u\}$ and so we are done by Lemma 10. We may therefore assume that w and x have distinct neighbours in $\{u, v\}$. Let G' be the graph obtained from G by deleting $C \setminus \{t\}$ and note that G' is connected since t is adjacent to both u and v. Note that both u and v have degree $k-1$ in G'. We can therefore find a $(k-1)$-degenerate order $\mathcal{O}$ of G' in $O(kn)$ time by taking the vertices in the reverse of the order they are found in a breadth-first search from u. Furthermore, since the only neighbours of t in G' are u and v, both of which have degree $k-1$, by moving t to the start of the order $\mathcal{O}$, we obtain a another $(k-1)$-degenerate order $\mathcal{O}'$. By Lemma 7, we can therefore find a k-degenerate decomposition (A, B) of G' such that $\mathcal{O}'|_B$ is a $(k-2)$-degenerate order on B and $t \in A$. Thus both u and v belong to B. We let $A' = A \cup \{w\}$ and $B' = B \cup (C \setminus \{w, t\})$, and claim that (A', B') is a k-degenerate decomposition of G. It is clear that A' is an independent set. We have the $(k-2)$-degenerate order $\mathcal{O}'|_B$ on B. We obtain a $(k-2)$-degenerate order on B' by appending to $\mathcal{O}'|_B$ the vertices of $C \setminus \{w, t\}$ beginning with x. Indeed, the only neighbour of x that is earlier in the

Algorithm 1: Finding a k-degenerate decomposition for connected Δ-regular graphs.

Input : A connected Δ-regular graph G
Output : A k-degenerate decomposition of G

1 *find a good pair u, v and let C be a component of $G \setminus \{u, v\}$*;
2 **if** *u, v is a strong pair* **then** apply Lemma 8;
3 **else if** *the union of C and one or both of u and v is a clique on $\Delta + 1$ vertices minus an edge* **then** apply Lemma 9;
4 **else if** *C is a clique on Δ vertices whose neighbourhood is $\{u, v\}$* **then** apply Lemma 10;
5 **else if** *C is a (u, v)-lock* **then** apply Lemma 11;
6 **else**
7 | *find a good pair $u', v' \in C$ such that either $C' = G \setminus \{u', v'\}$ is connected or $G \setminus \{u', v'\}$ has a component C' that is strictly contained in C*;
8 | set $u \leftarrow u'$, $v \leftarrow v'$, $C \leftarrow C'$;
9 | **go to** *Line 2*
10 **end**

order is its single neighbour in $\{u, v\}$ (and note that $1 \leq k - 2$ since $k \geq 3$). Furthermore, since $w, t \in A'$, every vertex in $C \setminus \{w, t, x\}$ has only $k - 2$ neighbours in B'. ◀

A pair of non-adjacent vertices u, v in a graph is a *good* pair if u and v have a common neighbour. Note that if a good pair u, v is not strong, then $G \setminus \{u, v\}$ must be disconnected. We are now ready to state and prove the following result.

▶ **Theorem 12.** *Let $\Delta \geq 3$ and G be a graph on n vertices with maximum degree at most Δ. If no component of G is isomorphic to $K_{\Delta+1}$, then a Δ-degenerate decomposition of G can be found in $O(\Delta n^2)$ time.*

Proof. We may assume that G is connected, otherwise it can be considered componentwise. If G is not Δ-regular, then it has a vertex u of degree at most $\Delta - 1$, so we can find a $(\Delta - 1)$-degenerate order $\mathcal{O}$ of G in $O(kn)$ time by taking the vertices in the reverse of the order they are found in a breadth-first search from u. In this case, we are done by Lemma 7. For Δ-regular graphs, we use the procedure shown in Algorithm 1. (Whenever we apply one of the lemmas of this section, we set $k = \Delta$. We note in passing that if we could assume that G is biconnected, then we would immediately have, by [1, Lemma 3], that there is always a good pair u, v such that $G \setminus \{u, v\}$ is connected. As we cannot make this assumption however, we are not able to make use of this result.)

Let us make a few comments on this procedure. As G is regular, connected and not complete, we can initially choose any vertex as u and find another vertex v to form a good pair in $O(\Delta^2) = O(\Delta n)$ time. If we perform a breadth-first search (which takes $O(n+m) = O(\Delta n)$ time) from a neighbour of u that retreats from u or v whenever they are encountered, we discover a component of $G \setminus \{u, v\}$. If the component contains a common neighbour of u and v but is not equal to $G \setminus \{u, v\}$, we repeat starting from a neighbour of u or v that was not discovered. Thus we discover in $O(\Delta n)$ time that either u, v is a strong pair (if we find all components of $G \setminus \{u, v\}$ and they each contain a common neighbour of u and v), or that it is not. We set C to be one of the components of $G \setminus \{u, v\}$ arbitrarily. By Lemma 8, we therefore conclude that Lines 1 and 2 take $O(\Delta n)$ time. It is easy to check in $O(\Delta n)$ time whether we apply Lemmas 9–11 on Lines 3–5 and applying these lemmas takes $O(\Delta n)$

in each case. Now suppose that we do not apply any of these lemmas, in which case we reach Line 7. We will show that we can find u', v' and, if necessary, C' in $O(\Delta n)$ time. If we find u', v' such that $C' = G \setminus \{u', v'\}$ is connected, then u', v' is a strong pair, so after executing Line 9, the algorithm will stop on Line 2. In all other cases C' will be strictly smaller than C. This means that we apply Line 9 at most $O(n)$ times, implying that we execute Lines 2–9 at most $O(n)$ times. This will give an overall running time of $O(\Delta n^2)$. It remains to show that if execution reaches Line 7 then we can find the required good pair u', v' and the component C' in $O(\Delta n)$ time.

Let us first show that C contains good pairs — that is, that it is not a clique. If C is a clique, then it contains either $\Delta - 1$ or Δ vertices (as each vertex has degree Δ in G and the only other possible neighbours of vertices in C are u and v). If C has $\Delta - 1$ vertices, then each vertex of C must be adjacent to both u and v and we would have applied Lemma 9 on Line 3, a contradiction. If C has Δ vertices, then each vertex of C is adjacent to exactly one of u and v (and neither u nor v can be adjacent to every vertex in C, as this would form a $K_{\Delta+1}$, contradicting the fact that G is connected), in which case we would have applied Lemma 10 on Line 4, a contradiction. Therefore we may assume that C is not a clique.

We need describe how to choose a good pair u', v' in C. If we can show that u and v are in the same component of $G \setminus \{u', v'\}$ (which must necessarily contain all of $G \setminus C$), then we are done as either $G \setminus \{u', v'\}$ is connected or there is another component C' of $G \setminus \{u', v'\}$ which must be contained in C (and note that in this case C' can be found in $O(\Delta n)$ time using breadth-first search).

If u and v have a common neighbour outside C or at least three common neighbours in C, then any good pair in C can be chosen as u', v' (as u and v will then be in the same component of $G \setminus \{u', v'\}$). If u and v have exactly two common neighbours t_1, t_2 that both belong to C, then any good pair other than t_1, t_2 can be chosen as u', v'. If t_1, t_2 is the only good pair in C (so all other vertices in C are adjacent), then C is a clique minus an edge and must contain Δ vertices (t_1, t_2 and the $\Delta - 2$ neighbours of t_1 that are not in $\{u, v\}$). Considering degree, any vertex in C other than t_1 or t_2 must be adjacent to exactly one of u and v. If every vertex in $C \setminus \{t_1, t_2\}$ is adjacent to, say u, then $C \cup \{u\}$ is a clique on $\Delta + 1$ vertices minus an edge and we would have applied Lemma 9 on Line 3, a contradiction. We may therefore assume that at least one vertex in $C \setminus \{t_1, t_2\}$ is adjacent to u and at least one is adjacent to v, so there is a path from u to v avoiding t_1 and t_2, and $G \setminus \{t_1, t_2\}$ is connected, so we are done.

Finally, suppose that u and v have exactly one common neighbour t that belongs to C. Then any good pair not including t can be chosen as u', v', as then u and v will be in the same component of $G \setminus \{u', v'\}$. Suppose, for contradiction, that no such pair exists. Then $C \setminus \{t\}$ is a clique. The vertex t has $\Delta - 2$ neighbours in $C \setminus \{t\}$. Since $\Delta \geq 3$, let z be one of those neighbours. Since t is the only common neighbour of u and v, we have that z can only be adjacent to at most one of u and v. Therefore, t has a neighbour non-adjacent to z, so z must have a neighbour non-adjacent to t, which we denote w. As w is also adjacent to at most one of u and v, it also has a neighbour x that is a non-neighbour of t (and cannot be t itself). So $C \setminus \{t\}$ contains at least Δ vertices: the $\Delta - 2$ neighbours of t plus w and x. As $C \setminus \{t\}$ induces a clique, it must have exactly Δ vertices, since G cannot contain a $K_{\Delta+1}$. Thus the set C forms a lock, and so we would have applied Lemma 11 on Line 5. This contradiction completes the proof. ◀

We note that Theorems 2 and 12 concern decompositions (A, B) of the vertex set of a graph where A is independent and B induces a $(\Delta - 2)$-degenerate graph. As B therefore cannot be a clique on Δ vertices, we have the following corollary.

▶ **Corollary 13.** *Let $\Delta \geq 3$ and G be a graph on n vertices with maximum degree at most Δ. If the clique number of G is at most Δ, a decomposition of the vertices of G into sets A and B, where A is an independent set and B induces a graph that has clique number at most $\Delta - 1$, can be found in $O(n)$ time if $\Delta = 3$ and $O(\Delta n^2)$ time for all Δ.*

Catlin [8] proved that such decompositions exist, but his result did not imply the existence of a polynomial-time algorithm to find the decomposition. We note that his result contains the additional claim that A is maximum; we cannot hope for an algorithmic version, as this would solve the NP-hard problem of finding a maximum independent set in a cubic graph [15].

5 Reconfigurations of Vertex Colourings

Our interest in Stable($\mathcal{D}_k$) when $k \geq 2$ case stems from an open problem in the area of graph reconfigurations. For a graph G and integer $k \geq 1$, the k-colouring reconfiguration graph $R_k(G)$ has the vertex set consisting all possible k-colourings of G and two vertices of $R_k(G)$ are adjacent if and only if the corresponding k-colourings differ on exactly one vertex. A central problem in the area of reconfiguration is the Reachability problem. In the context of k-colourings, this is the problem of finding a path (if one exists) in $R_k(G)$ between two given k-colourings α and β of a graph G. The complexity of this problem has been studied for graphs G with maximum degree Δ and any positive integer k. This problem is PSPACE-hard for $k \geq 4$, $\Delta \geq k$ [3], solvable in $O(n + m)$ time on (general) graphs with n vertices and m edges for $k \leq 3$ [18] and solvable in $O(n^2)$ time for $k \geq 4$, $0 \leq \Delta \leq k - 2$ [10]. This leaves us with the case $k \geq 4$, $\Delta = k - 1$. In [14, Theorem 6], three of the authors of the current paper showed that this case can be solved in polynomial time as long as the input graph G is not Δ-regular, that is, if not all vertices in G have maximum degree Δ. The case where G is Δ-regular was left as an open problem, but using Theorem 12 we can now resolve it by giving an $O(n^2)$-time algorithm (assuming Δ is fixed and not part of the input).

▶ **Theorem 14.** *Fix $\Delta \geq 0$. Let G be a connected graph on n vertices with maximum degree Δ. The problem of finding a path (if one exists) between two k-colourings α and β in $R_k(G)$ is*

- *$O(n)$-time solvable if $1 \leq k \leq 3$;*
- *$O(n^2)$-time solvable if $k \geq 4$ and $0 \leq \Delta \leq k - 1$;*
- PSPACE*-hard if $k \geq 4$ and $\Delta \geq k$.*

6 Future Work

As well as showing that the problem of recognizing near-bipartite graphs of diameter 3 is NP-complete, we have proven that for every integer $\Delta \geq 3$, the Stable($\mathcal{D}_{\Delta-2}$) problem is polynomial-time solvable on graphs of maximum degree Δ. Is Stable($\mathcal{D}_{\Delta-2}$) NP-complete for graphs of maximum degree $\Delta + 1$? Recall that this is known to be the case for $\Delta = 3$, as proven by Yang and Yuan [29]. Their proof can easily be adapted to prove that Stable($\mathcal{D}_{\Delta-2}$) is NP-complete for graphs of maximum degree $2\Delta - 2$ for every $\Delta \geq 4$.

References

1 Bradley Baetz and David R. Wood. Brooks' vertex-colouring theorem in linear time. *CoRR*, abs/1401.8023, 2014.

2 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Independent feedback vertex set for P_5-free graphs. Manuscript, 2017.

3 Paul S. Bonsma and Luis Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009.

4 Andreas Brandstädt, Synara Brito, Sulamita Klein, Loana Tito Nogueira, and Fábio Protti. Cycle transversals in perfect graphs and cographs. *Theoretical Computer Science*, 469:15–23, 2013.

5 Andreas Brandstädt, Peter L. Hammer, Van Bang Le, and Vadim V. Lozin. Bisplit graphs. *Discrete Mathematics*, 299(1–3):11–32, 2005.

6 Andreas Brandstädt, Van Bang Le, and Thomas Szymczak. The complexity of some problems related to graph 3-colorability. *Discrete Applied Mathematics*, 89(1–3):59–73, 1998.

7 Leizhen Cai and Derek G. Corneil. A generalization of perfect graphs – i-perfect graphs. *Journal of Graph Theory*, 23(1):87–103, 1996.

8 Paul A. Catlin. Brooks' graph-coloring theorem and the independence number. *Journal of Combinatorial Theory, Series B*, 27(1):42–48, 1979.

9 Paul A. Catlin and Hong-Jian Lai. Vertex arboricity and maximum degree. *Discrete Mathematics*, 141(1–3):37–46, 1995.

10 Luis Cereceda. *Mixing graph colourings*. PhD thesis, London School of Economics, 2007.

11 Konrad K. Dabrowski, Vadim V. Lozin, and Juraj Stacho. Stable-Π partitions of graphs. *Discrete Applied Mathematics*, 182:104–114, 2015.

12 François Dross, Mickaël Montassier, and Alexandre Pinlou. Partitioning sparse graphs into an independent set and a forest of bounded degree. *CoRR*, abs/1606.04394, 2016.

13 Tomás Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. List partitions. *SIAM Journal on Discrete Mathematics*, 16(3):449–478, 2003.

14 Carl Feghali, Matthew Johnson, and Daniël Paulusma. A reconfigurations analogue of Brooks' Theorem and its consequences. *Journal of Graph Theory*, 83(4):340–358, 2016.

15 Michael Randolph Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

16 Martin Grötschel, László Lovász, and Alexander Schrijver. Polynomial algorithms for perfect graphs. *Annals of Discrete Mathematics*, 21:325–356, 1984.

17 Chính T. Hoàng and Van Bang Le. On P_4-transversals of perfect graphs. *Discrete Mathematics*, 216(1–3):195–210, 2000.

18 Matthew Johnson, Dieter Kratsch, Stefan Kratsch, Viresh Patel, and Daniël Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321, 2016.

19 Jan Kratochvíl and Ingo Schiermeyer. On the computational complexity of $(\mathcal{O}, \mathcal{P})$-partition problems. *Discussiones Mathematicae Graph Theory*, 17(2):253–258, 1997.

20 László Lovász. Coverings and coloring of hypergraphs. *Congressus Numerantium*, VIII:3–12, 1973.

21 Vadim V. Lozin. Between 2- and 3-colorability. *Information Processing Letters*, 94(4):179–182, 2005.

22 Nadimpalli V. R. Mahadev and Uri N. Peled. *Threshold graphs and related topics*, volume 56 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, 1995.

23 Martín Matamala. Vertex partitions and maximum degenerate subgraphs. *Journal of Graph Theory*, 55(3):227–232, 2007.

24 Colin McDiarmid and Nikola Yolov. Recognition of unipolar and generalised split graphs. *Algorithms*, 8(1):46–59, 2015.

25 George B. Mertzios and Paul G. Spirakis. Algorithms and almost tight results for 3-colorability of small diameter graphs. *Algorithmica*, 74(1):385–414, 2016.

26 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theoretical Computer Science*, 461:65–75, 2012.

27 Yuma Tamura, Takehiro Ito, and Xiao Zhou. Algorithms for the independent feedback vertex set problem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E98-A(6):1179–1188, 2015.

28 Yongqi Wu, Jinjiang Yuan, and Yongcheng Zhao. Partition a graph into two induced forests. *Journal of Mathematical Study*, 29:1–6, 1996.

29 Aifeng Yang and Jinjiang Yuan. Partition the vertices of a graph into one independent set and one acyclic set. *Discrete Mathematics*, 306(12):1207–1216, 2006.

Parameterized Algorithms and Kernels for Rainbow Matching

Sushmita Gupta[1], Sanjukta Roy[2], Saket Saurabh[3], and Meirav Zehavi[4]

1 **University of Bergen, Norway**
Sushmita.Gupta@uib.no
2 **The Institute of Mathematical Sciences, HBNI, Chennai, India**
sanjukta@imsc.res.in
3 **University of Bergen, Norway, and**
The Institute of Mathematical Sciences, HBNI, Chennai, India
saket@imsc.res.in
4 **University of Bergen, Norway**
Meirav.Zehavi@uib.no

Abstract

In this paper, we study the NP-complete *colorful* variant of the classical Matching problem, namely, the Rainbow Matching problem. Given an edge-colored graph G and a positive integer k, this problem asks whether there exists a matching of size at least k such that all the edges in the matching have distinct colors. We first develop a deterministic algorithm that solves Rainbow Matching on paths in time $\mathcal{O}^\star(\left(\frac{1+\sqrt{5}}{2}\right)^k)$ and polynomial space. This algorithm is based on a curious combination of the method of bounded search trees and a "divide-and-conquer-like" approach, where the branching process is guided by the maintenance of an auxiliary bipartite graph where one side captures "divided-and-conquered" pieces of the path. Our second result is a randomized algorithm that solves Rainbow Matching on general graphs in time $\mathcal{O}^\star(2^k)$ and polynomial-space. Here, we show how a result by Björklund et al. [JCSS, 2017] can be invoked as a black box, wrapped by a probability-based analysis tailored to our problem. We also complement our two main results by designing kernels for Rainbow Matching on general and bounded-degree graphs.

Keywords and phrases Rainbow Matching, Parameterized Algorithm, Bounded Search Trees, Divide-and-Conquer, 3-Set Packing, 3-Dimensional Matching

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.71

1 Introduction

The classical notion of matching has been extensively studied for several decades in the area of Combinatorial Optimization [6, 14]. Given an undirected graph G, a set of edges is called a *matching* if the edges are pairwise non-adjacent. That is, no two edges share a common vertex. In the Maximum Matching problem, the objective is to find a matching of maximum size. The first polynomial time algorithm for Maximum Matching was given by Edmonds [6] in his classic paper *Paths, Trees and Flowers*. It is important to remark that this is the paper which underlined the importance of study of polynomial time algorithms for the first time. After a series of improvements, the current fastest algorithm for Maximum Matching was given by Micali and Vazirani and it runs in time $\mathcal{O}(m\sqrt{n})$ [15]. However, finding a matching that satisfies some additional constraints often immediately becomes NP-complete, where three notable examples are Minimum Maximal Matching [18], Induced

Matching [17] and Multiple Choice Matching [11]. In this paper, we study the NP-hard variant of Maximum Matching called Multiple Choice Matching from the viewpoint of parameterized complexity.

The Multiple Choice Matching problem, also called Rainbow Matching, is one of the NP-hard variant of Maximum Matching mentioned in the classical book by Garey and Johnson [9, Problem GT55]. In this work, we will stick to the name Rainbow Matching. This problem is formally defined as follows.

Rainbow Matching	**Parameter:** k
Input: An undirected graph G, a coloring function $\chi : E(G) \to \{1, \ldots, q\}$ and a positive integer k.	
Question: Does there exist a matching of size at least k such that all the edges in the matching have distinct colors?	

A matching where all the edges have distinct colors will be called *colorful matching*. Itai et al. [11] showed, already in 1978, that Rainbow Matching is NP-complete on (edge-colored) bipartite graphs. Close to three decades later, Le and Pfender [13] revisited the computational complexity of this problem. Specifically, they showed that the Rainbow Matching problem is NP-complete even on (edge-colored) paths, complete graphs, P_8-free trees in which every color is used at most twice, P_5-free linear forests in which every color is used at most twice, and P_4-free bipartite graphs in which every color is used at most twice. In this paper, we consider this problem from the parameterized rather than classical complexity perspective.

A *parameterization* of a problem is the association of an integer k with each input instance, which results in a *parameterized problem*. For our purposes, we need to recall three central notions that define the parameterized complexity of a parameterized problem. The first one is the notion of a *kernel*. Here, a parameterized problem is said to admit a *kernel* of size $f(k)$ for some function f that depends *only* on k if there exists a polynomial-time algorithm, called a *kernelization algorithm*, that translates any input instance into an equivalent instance of the same problem whose size is bounded by $f(k)$ and such that the value of the parameter does not increase. In case the function f is polynomial in k, the problem is said to admit a *polynomial kernel*. Hence, kernelization is a mathematical concept that aims to analyze the power of preprocessing procedures in a formal, rigorous manner. The second notion that we use is the one of *fixed-parameter tractability (FPT)*. Here, a parameterized problem Π is said to be FPT if there is an algorithm that solves it in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the size of the input and f is a function that depends only on k. Such an algorithm is called a *parameterized algorithm*. In other words, the notion of FPT signifies that it is not necessary for the combinatorial explosion in the running time of an algorithm for Π to depend on the input size, but it can be confined to the parameter k. Finally, we recall that Parameterized Complexity also provides tools to refute the existence of polynomial kernels and parameterized algorithms for certain problems (under plausible complexity-theoretic assumptions). We refer the reader to the books [3, 5] for more information on these notions in particular, and on Parameterized Complexity in general. The notation $\mathcal{O}^\star(\cdot)$ is used to hide factors polynomial in the input size.

1.1 Our Contribution

Our starting point is the FPT algorithm mentioned in the article of Le and Pfender [13]. This algorithm is based on the connection between Rainbow Matching and 3-Set Packing. In the 3-Set Packing problem, we are given a universe U, a set family $\mathcal{F}$ consisting of subsets

of U of size at most 3 and an integer k, and the objective is to check whether there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ containing at least k pairwise-disjoint sets. Observe that given an instance $I = (G, \chi, k)$ of Rainbow Matching, we can view I as an instance of 3-Set Packing by setting $U = V(G) \cup \{1, \ldots, q\}$, and letting $\mathcal{F}$ contain every set $\{u, v, \chi(e)\}$ corresponding to an edge $e = uv \in E(G)$. Now, observe that (G, χ, k) is a yes-instance of Rainbow Matching if and only if $(U, \mathcal{F}, k)$ is a yes-instance of 3-Set Packing. This immediately implies that known algorithms for 3-Set Packing can be employed to solve Rainbow Matching. In particular, using the known algorithms for 3-Set Packing, we obtain the following algorithms for Rainbow Matching: (1) a deterministic algorithm running in time $\mathcal{O}^\star(8.097^k)$ [19]; (2) a randomized algorithm running in time $\mathcal{O}^\star(1.4953^{3k}) = \mathcal{O}^\star(3.3434^k)$ [2].

Rainbow Matching on Paths. Our first contribution concerns the Rainbow Matching problem on paths. We obtain the following algorithm, which is faster than the one that we design later for general graphs.

▶ **Theorem 1.** *There exists a deterministic algorithm for* Rainbow Matching *on paths that runs in time* $\mathcal{O}^\star\left(\left(\frac{1+\sqrt{5}}{2}\right)^k\right)$ *and uses polynomial space.*

The proof of Theorem 1 is based on a combination of the classical method of bounded search trees [3, 5, 7, 8] together with a "divide-and-conquer-like" approach. The algorithm always maintains a family of vertex-disjoint paths $\mathcal{S}$, and the objective is to find a colorful matching of size k that uses exactly one edge from each path in $\mathcal{S}$ and $k - |\mathcal{S}|$ edges from P. We call this variant of Rainbow Matching the Disjoint Set Rainbow Matching problem. Observe that when $\mathcal{S} = \emptyset$, then Disjoint Set Rainbow Matching is precisely Rainbow Matching. To compactly represent potential partial solutions to our problem at every step of the recursion, that is, partial witnesses that there may indeed exist a colorful matching that uses exactly one edge from each path in $\mathcal{S}$, our algorithm works as follows. The algorithm uses an auxiliary bipartite graph where we maintain a partial solution to our problem, in terms of a matching in this bipartite graph. This has the additional benefit that the measure becomes very simple: we just measure the size left to cover, *i.e.* $k - t$, where $t = |\mathcal{S}|$ denotes the size of the partial solution. (We remark that we are able to construct a solution in the same time as it takes to solve the decision version of the Disjoint Set Rainbow Matching problem.)

Rainbow Matching on General Graphs. Our second contribution is an algorithm on general graphs that is better than the known algorithms for 3-Set Packing. In particular, we obtain the following result.

▶ **Theorem 2.** *There exists a randomized algorithm for* Rainbow Matching *with constant, one-sided error that runs in time* $\mathcal{O}^\star(2^k)$ *and uses polynomial space.*[1]

The proof of Theorem 2 is based on the general method described in [2] for solving various packing and matching problems. We tailor the analysis of Bjorklund et al. [2] to the Rainbow Matching problem. This gives us the desired saving over the algorithm for 3-Set Packing.

[1] Specifically, if the algorithm determines that an input instance is a yes-instance, then this answer is necessarily correct.

Kernelization. Finally, we turn to consider the question of kernelization. Here, we exploit the connection between our problem and 3-Set Packing to design a kernelization algorithm. We also design a smaller kernel for Rainbow Matching on paths, or more generally, for graphs of bounded degree.

▶ **Theorem 3.** Rainbow Matching *admits a kernel of size* $\mathcal{O}(k^3)$ *on general graphs. Moreover, it admits a kernel of size* $\mathcal{O}(dk^2)$ *on graphs of maximum degree* d*.*

1.2 Related Work

Le and Pfender [13] gave a factor $(\frac{2}{3} - \epsilon)$ approximation algorithm for Rainbow Matching on general graphs for every $\epsilon > 0$. They also designed a few polynomial time algorithms when the instances of Rainbow Matching are restricted to special graph classes. As stated before, Le and Pfender [13] also related this problem to 3-Set Packing, and showed that the problem is FPT. Moreover, they showed that the problem is FPT on P_5 free forests parameterized by the number of components. Colorful matchings have also been studied from graph theoretic and combinatorial perspectives. For example, they are related to Ryser's famous conjecture regarding Latin transversal [16]. In the language of colorful matchings, the conjecture says that every proper edge coloring of the complete bipartite graph $K_{2n+1,2n+1}$ with $2n + 1$ colors contains a rainbow matching with $2n + 1$ edges. Additional examples are studies of sufficient conditions for edge-colored graphs to guarantee the existence a colorful matching of a certain size. Moreover, previous studied also examined what is the size of the largest colorful matching in an edge-colored graph with additional restrictions. For more information on these topics, we refer to [13] and references therein. Finally, let us mention that colorful matchings belong to a family of problems called rainbow subgraph problems. A *rainbow subgraph* of an edge-colored graph is a subgraph whose edges have distinct colors. We refer to [12] for a survey containing results and questions regarding rainbow subgraps.

1.3 Preliminaries

Let $[n]$ denote the set $\{1, \ldots, n\}$. For a graph G, we let $V(G)$ and $E(G)$ denote its vertex set and its edge set, respectively. For two vertices $u, v \in V(G)$, we use uv to denote an edge between u and v.

Reduction Rule. To design our kernelization algorithm, we rely on the notion of a *reduction rule*. A reduction rule is a polynomial-time procedure that replaces an instance $(\mathcal{I}, k)$ of a parameterized problem Π (where k is the parameter) by a new instance $(\mathcal{I}', k')$ of Π. The rule is said to be *safe* if $(\mathcal{I}, k)$ is a yes-instance if and only if $(\mathcal{I}', k')$ is a yes-instance. As customary in the field we allow reduction rule to return the answer yes or the answer no (see [3, 5]).

2 Algorithm for Rainbow Matching on Paths

In this section we give a deterministic algorithm for Rainbow Matching on paths that is faster than the algorithm we will present for Rainbow Matching on general graphs in the next section. Towards that we will solve the following general problem, and from there solve Rainbow Matching in paths.

Disjoint Set Rainbow Matching	**Parameter:** k
Input: A path P with edge coloring $\chi : E(G) \to [q]$, a collection $\mathcal{S}$ of (vertex disjoint) paths (vertex disjoint from P) of arbitrary lengths, and a positive integer k.	
Question: Does there exist a colorful matching of size k that uses exactly one edge from each path in $\mathcal{S}$ and $k - \lvert\mathcal{S}\rvert$ edges from P?	

Note that Rainbow Matching is a special case of Disjoint Set Rainbow Matching, where P is the input graph (a path), $\mathcal{S} = \emptyset$ and k is the parameter. Thus, solving Disjoint Set Rainbow Matching will yield an algorithm for Rainbow Matching.

Overview. To solve Disjoint Set Rainbow Matching, whenever possible we apply reduction rules, or solve the instance in polynomial time. In the absence of either of these, the algorithm branches on an edge, based on whether it is part of the solution or not.

Measure. We associate the measure $\mu(P, \mathcal{S}, k) = k - |\mathcal{S}|$ to the instance $(P, \mathcal{S}, k)$. We will use this measure to bound the number of nodes in the search tree. When the instance is clear from the context, we will simply use μ.

Auxiliary bipartite graph. At every step of the search we maintain a bipartite graph $\mathcal{B}(\mathcal{S})$ on the vertex set $([q], \mathcal{S})$ and edge set containing pairs $cP' \in [q] \times \mathcal{S}$ such that color c appears on an edge in the path P' in (the collection) $\mathcal{S}$.

We first prove a lemma which allows us to solve the Disjoint Set Rainbow Matching problem when the measure is at most one.

▶ **Lemma 4.** *If $\mu(P, \mathcal{S}, k) \leq 1$, then we can test if $(P, \mathcal{S}, k)$ is a yes-instance in polynomial time.*

Proof. We divide the proof based on whether $\mu(P, \mathcal{S}, k) < 0$ or $\mu(P, \mathcal{S}, k) = 0$ or $\mu(P, \mathcal{S}, k) = 1$. If $\mu(P, \mathcal{S}, \mathcal{C}, k) < 0$, then $k < |\mathcal{S}|$ and so clearly no matching of size k can exist which chooses exactly one edge from each path in $\mathcal{S}$.

If $\mu(P, \mathcal{S}, k) = 0$, then $k = |\mathcal{S}|$. Let $Q_1, \ldots, Q_k$ denote the paths in $\mathcal{S}$. We will show that there exists a colorful matching of size k that uses exactly one edge from each path in $\mathcal{S}$ if and only if there is a matching in $\mathcal{B}(\mathcal{S})$ that saturates $\mathcal{S}$. Let $\mathcal{M}$ be a colorful matching of size k that uses exactly one edge from each path in $\mathcal{S}$. Furthermore, for each i $(1 \leq i \leq k)$ let $m_i \in \mathcal{M}$ be the edge that is part of path Q_i. Then $\{\chi(m_i)Q_i \mid i \in [k]\}$ forms a matching that saturates $\mathcal{S}$ in $\mathcal{B}(\mathcal{S})$. In the reverse direction given a matching $\mathcal{M}'$ in $\mathcal{B}(\mathcal{S})$ that saturates $\mathcal{S}$, we obtain a colorful matching $\mathcal{M}$ that uses exactly one edge from each path Q_i in $\mathcal{S}$, as follows. Since $\mathcal{M}'$ saturates $\mathcal{S}$ we have that for every path $Q_i \in \mathcal{S}$ there is an edge jQ_i for some $j \in [q]$. This implies that there is an edge, say m_i on Q_i such that $\chi(m_i) = j$, *i.e.* m_i has color j. Since, the paths in $\mathcal{S}$ are pairwise vertex disjoint, the set $\mathcal{M}^\star = \{m_i \mid i \in [k]\}$ forms a matching in the graph P. Recall that $\mathcal{M}'$ is a matching in $\mathcal{B}(\mathcal{S})$ in which one of the endpoints of the edges are from the set $[q]$, thus, it follows that $\mathcal{M}^\star$ is a colorful matching of size k that uses exactly one edge from each path in $\mathcal{S}$. Thus, implying that in this case we can check whether or not $(P, \mathcal{S}, k)$ is a yes-instance in polynomial time by checking if the bipartite graph $\mathcal{B}(\mathcal{S})$ has a matching that saturates $\mathcal{S}$ [10].

If $\mu(P, \mathcal{S}, k) = 1$, then $k = |\mathcal{S}| + 1$. In this case, we consider every edge $e = uv$ on P. Let us now consider a specific iteration concerning an edge $e = uv$ on P. Then, we construct $\mathcal{B}(\mathcal{S} \cup \{uv\})$. Similar to the case of $\mu(P, \mathcal{S} \cup \{uv\}), k) = 0$, we have now reduced the problem into checking whether or not, there is a matching in $\mathcal{B}(\mathcal{S} \cup \{uv\}))$ that saturates $\mathcal{S} \cup \{uv\}$.

This condition can be tested in polynomial time. If in at least one iteration, we found a saturating matching then we have a yes-instance, and otherwise we have no-instance. This completes the proof. ◀

Lemma 4 yields the following reduction rule.

▶ **Reduction Rule 5.** *If $\mu(P, \mathcal{S}, k) \leq 1$, then using Lemma 4 test whether or not $(P, \mathcal{S}, k)$ is a yes-instance. If Lemma 4 returns yes, then return that $(P, \mathcal{S}, k)$ is a yes-instance; else, return that $(P, \mathcal{S}, k)$ is a no-instance.*

The safeness of Reduction Rule 5 follows from Lemma 4. The next reduction rule allows us to identify a prefix of the path P such that there exists a colorful matching of size at least k that contains *exactly* one edge from the prefix.

▶ **Reduction Rule 6.** *In the instance $(P, \mathcal{S}, k)$, let $P = v_1, v_2, \ldots, v_{n-1}, v_n$. Suppose that for every index $i \in [n-1]$ the following property is true: when the subpaths $P_{i-1} = v_1, \ldots, v_{i-1}$ and $P' = v_i, v_{i+1}$ of P are added to $\mathcal{S}$, the size of a maximum matching in the new bipartite graph $\mathcal{B}(\mathcal{S} \cup \{P_{i-1}, P'\})$ (obtained after the addition of P_{i-1} and P' to $\mathcal{S}$, and suitable edges) is at most $|\mathcal{S}| + 1$. Then, return that $(P, \mathcal{S}, \mathcal{C}, k)$ is a no-instance.*

Next we show that the correctness of Reduction Rule 6.

▶ **Lemma 7.** *Reduction Rule 6 is safe.*

Proof. For the sake of contradiction, we assume that $(P, \mathcal{S}, k)$ is a yes-instance. In this case we will show that there exists an index $i \in [n-1]$ with the following property: when the subpaths $P_{i-1} = v_1, \ldots, v_{i-1}$ and $P' = v_i, v_{i+1}$ of P are added to $\mathcal{S}$, the size of a maximum matching in the new bipartite graph $\mathcal{B}(\mathcal{S} \cup \{P_{i-1}, P'\})$ is $|\mathcal{S}| + 2$. This will be contradiction, and thereby prove the lemma.

Since $(P, \mathcal{S}, k)$ is a yes-instance there exists a colorful matching of size k, denoted by $\mathcal{M}$, that uses k colors from $[q]$, exactly one edge from each path in $\mathcal{S}$, and $k - |\mathcal{S}|$ edges from P. Thus, there is a maximum matching in the bipartite graph on $\mathcal{B}(\mathcal{S})$ that saturates every vertex in the $\mathcal{S}$-side. It is obtained by taking edges that connect a vertex in the $\mathcal{S}$-side (*i.e.* a path in the collection $\mathcal{S}$) with the color that appears on the matching edge in $\mathcal{M}$ that is part of the same path. We use $\mathcal{M}'$ to denote this bipartite matching.

Let $\{e_{j_1}, e_{j_2}, \ldots, e_{j_{k-|\mathcal{S}|}}\}$ denote the matching edges in $\mathcal{M}$ as they appear left to right in P. For some $i \geq 3$, let $v_i v_{i+1}$ denote the edge e_{j_2} (the matching edge with the second smallest index in P). It follows that edge e_{j_1} appears in the subpath $P_{i-1} = v_1, \ldots, v_{i-1}$. Note that the vertex $\chi(e_{j_2}) \in [q]$ is not saturated by $\mathcal{M}'$ in $\mathcal{B}$ because e_{j_2} is part of the matching $\mathcal{M}$ while it is inside P. Similarly, the vertex $\chi(e_{j_1}) \in [q]$ is also not saturated by $\mathcal{M}'$ in $\mathcal{B}$. Hence, when paths P_{i-1} and P' are added to the bipartite graph, $\mathcal{M}'$ can be extended by exactly two more edges: edges between $\chi(e_{j_1})$ and P_{i-1} and $\chi(e_{j_2})$ and P'. Thus, the new bipartite graph has a matching of size $|\mathcal{S}| + 2$. ◀

The safeness of Reduction Rule 6, leads us to the following conclusion.

▶ **Lemma 8.** *If $(P, \mathcal{S}, k)$ is a yes-instance on which Reduction Rules 5 and 6 are not applicable, then there exists an index $i \in [n-1]$ such that there exists a colorful matching of size k that uses exactly one edge from the subpath $P_i = v_1, \ldots, v_i$ of P. Furthermore, such an index i can be found in polynomial time.*

Proof. Since Reduction Rules 5 and 6 are not applicable we have that $\mu(P, \mathcal{S}, k) \geq 2$. This implies that $k \geq |\mathcal{S}| + 2$. Let i denote the *smallest* integer in $[n-1]$ for which the following holds:

Property ($\star\star$) when the subpaths $P_{i-1} = v_1, \ldots, v_{i-1}$ and $P' = v_i, v_{i+1}$ of P are added to $\mathcal{M}$, the size of a maximum matching in the new bipartite graph $\mathcal{B}(\mathcal{S} \cup \{P_{i-1}, P'\})$ (obtained after the addition of P_{i-1} and P' to $\mathcal{S}$, and suitable edges) is $|\mathcal{S}| + 2$.

Observe that since $k \geq |\mathcal{S}| + 2 \geq 2$, such an integer i must exist.

Note that any colorful matching of size k uses at most *one* edge from P_i, else, it contradicts the fact that i is the smallest integer that satisfies Property ($\star\star$). Also note that since we have a colorful matching of size at least 2 in P_{i+1}, so $i \geq 3$. However, since there always exists a colorful matching of size k that uses one of the first two edges of the path P; hence, the matching must use one of the edges on P_i. This implies that there exists a colorful matching of size k that uses exactly one edge on P_i. Clearly, we can find the smallest integer described in the statement of the lemma in polynomial time. This concludes the proof. ◀

Lemma 8 yields a branching rule that can be described as follows. Let P_i be the subpath of P (given by Lemma 8) such that there exists a colorful matching of size k that uses exactly one edge from it. We recursively solve two subproblems one where we assume that edge v_iv_{i+1} is in the colorful matching of size k we are constructing, and the other where we assume that edge v_iv_{i+1} is not part of the solution we are constructing. Note that this rule is exhaustive because an edge (in particular, v_iv_{i+1}) can either belong to a matching, or it does not.

Algorithm. Now we can describe the branching rule in details along with the recursive call to a subproblem. Let $\mathcal{I} = (P = v_1, \ldots, v_n, \mathcal{S}, k)$ be the instance of Disjoint Set Rainbow Matching, where none of the Reduction Rules 5 or 6 are applicable. Let P_i be the subpath of P as described in Lemma 8.

Branch 1: (**The edge v_iv_{i+1} belongs to a colorful matching of size k.**)
We recursively solve the problem on the instance $(P \setminus \{v_1, \ldots, v_{i+1}\}, \mathcal{S} \cup \{P_{i-1}\} \cup \{[v_iv_{i+1}]\}, k)$. Since the size of $\mathcal{S}$ increases by 2, the measure μ decreases by 2. Observe that by Lemma 8 we know that there exists a colorful matching of size k that uses exactly one edge from the subpath P_i. However, since we have assumed that the edge v_iv_{i+1} belongs to the colorful matching of size k, thus, edge $v_{i-1}v_i$ cannot be part of the same matching, and so one of the edges in P_{i-1} must be part of the matching as well. Thus, in this case we know that two of the edges in our matching are due to the edge v_iv_{i+1} and the other is from P_{i-1}.

Branch 2: (**The edge v_iv_{i+1} does not belong to a colorful matching of size k.**)
In this case we recursively solve the problem on $(P \setminus \{v_1, \ldots, v_i\}, \mathcal{S} \cup \{P_i\}, k)$. Since the size of $\mathcal{S}$ increases by 1 and k remains the same, the measure μ decreases by 1. The correctness of this step follows from Lemma 8.

If either of the branches returns "yes", we return the same. Else, we return that the given instance is a no-instance.

The resulting branching vector for this algorithm is $(2, 1)$. Thus, solving the polynomial $x^2 \geq x + 1$ for a positive root yields $x \geq \frac{1}{2}(1 + \sqrt{5}) = 1.6181$. This upper bounds the running time of our algorithm. The correctness of the algorithm follows from Lemmas 4, 7 and 8.

Recall that as explained at the very onset of our discussion: Since Rainbow Matching is a special case of Disjoint Set Rainbow Matching, hence our algorithm can solve Rainbow Matching by using the algorithm for the latter on the instance $(G, \mathcal{S} = \emptyset, k)$. This completes the proof of Theorem 1.

3 FPT Algorithm for Rainbow Matching on General Graphs

This section is inspired by the proof of Theorem 4 in [2], which solves 3-Set Packing in time $\mathcal{O}^\star(3.3434^k)$. We show that by an analysis tailored to Rainbow Matching, we improve upon the time complexity $\mathcal{O}^\star(3.3434^k)$. More precisely, the objective of this section is to prove Theorem 2.

Towards the proof of Theorem 2, we need to consider a problem called 3-Set Prepacking, which was introduced in [2]. The input of this problem consists of an n-element universe U, an n_1-element subuniverse $U_1 \subseteq U$, a family $\mathcal{F}$ of 3-sets, a positive integer k, and three non-negative integers p_0, p_1 and p_2 whose sum is k. The task is to determine whether there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size k such that the 3-sets in $\mathcal{F}'$ are pairwise-disjoint, and for all $i \in \{0,1,2\}$, there exist exactly p_i sets S in $\mathcal{F}'$ such that $|S \cap U_1| = i$. We would need to rely on the following result.

▶ **Proposition 9.** *There exists a randomized algorithm for* 3-Set Prepacking *with constant, one-sided error that runs in time* $\mathcal{O}^\star(2^{3p_0+2p_1+p_2})$ *and uses polynomial space. Specifically, if the algorithm determines that an input instance is a yes-instance, then this answer is necessarily correct.*

Let us denote the algorithm given by Proposition 9 by PrepackAlg. We present a reduction from our problem to 3-Set Prepacking. For this purpose, we describe a procedure Reduce that given an instance $(G, \chi : V(G) \to [q], k)$ of Rainbow Matching, constructs an instance $\text{reduce}(G, \chi, k) = (U, U_1, \mathcal{F}, k, p_0, p_1, p_2)$ of 3-Set Prepacking with the same parameter k. Let us denote $n_1 = |V(G)| = n - q$, where n would denote $|U|$. First, Reduce sets $U = V(G) \cup [q]$, $\mathcal{F} = \{\{u, v, \chi(uv)\} : uv \in E(G)\}$, $p_0 = 0$, $p_1 = 0$ and $p_2 = k$. Second, Reduce sets $U_1 = V(G)$. Let us now argue that we obtain an equivalent instance.

▶ **Lemma 10.** *Let* $(G, \chi : V(G) \to [q], k)$ *be an instance of* Rainbow Matching*. Then,* $\text{reduce}(G, \chi, k) = (U, U_1, \mathcal{F}, k, p_0, p_1, p_2)$ *is a yes-instance of* 3-Set Prepacking *if and only if* $(G, \chi : V(G) \to [q], k)$ *is a yes-instance of* Rainbow Matching*.*

Proof. In the first direction suppose that $\text{reduce}(G, \chi, k) = (U, U_1, \mathcal{F}, k, p_0, p_1, p_2)$ is a yes-instance of 3-Set Prepacking. In particular, we then have that there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size k such that the 3-sets in $\mathcal{F}'$ are pairwise-disjoint. Let us denote $\mathcal{M} = \{uv \in E(G) : \exists S \in \mathcal{F}' \text{ s.t. } \{u, v\} \subseteq S\}$. Since $|\mathcal{F}'| = k$, we have that $|\mathcal{M}| = k$, and since the 3-sets in $\mathcal{F}'$ are pairwise-disjoint, we have that $\mathcal{M}$ is a colorful matching. Thus, $(G, \chi : V(G) \to [q], k)$ is a yes-instance of Rainbow Matching.

In the other direction suppose $(G, \chi : V(G) \to [q], k)$ is a yes-instance of Rainbow Matching. Then there exists a colorful matching $\mathcal{M}$ of size k. Let us denote $\mathcal{F}' = \{\{u, v, \chi(uv)\} | \ uv \in \mathcal{M}\}$. Since the size of $\mathcal{M}$ is k, we have that the size of $\mathcal{F}'$ is k and since $\mathcal{M}$ is a colorful matching we have that the sets in $\mathcal{F}'$ are pairwise disjoint. Notice that every set in $\mathcal{F}'$ exactly two elements from U_1. Therefore, for all $i \in \{0,1,2\}$, there exist exactly p_i sets S in $\mathcal{F}'$ such that $|S \cap U_1| = i$. Thus, $(U, U_1, \mathcal{F}, k, p_0, p_1, p_2)$ is a yes-instance of 3-Set Prepacking. ◀

Let us now prove Theorem 2.

Proof of Theorem 2. Given an instance $(G, \chi : V(G) \to [q], k)$ of Rainbow Matching, we construct the instance $\text{reduce}(G, \chi, k) = (U, U_1, \mathcal{F}, k, p_0, p_1, p_2)$ of 3-Set Prepacking. Then, we run the algorithm given by Proposition 9. We accept if and only if the algorithm from Proposition 9 accepted. The correctness follows from Proposition 9 and Lemma 10. Since, $p_0 = p_1 = 0$ and $p_2 = k$, by Proposition 9, the total running time is $\mathcal{O}^\star(2^k)$. ◀

4 Kernelization Algorithms

In this section we give a proof for Theorem 3. We first describe a kernel on general graphs. The kernelization algorithm on general graphs is actually a known kernel for 3-Set Packing given in [3, Theorem 12.20] (also see [1, 4]). The best known kernel for 3-Set Packing is given by Abu-Khzam [1] and it has $\mathcal{O}(k^2)$ elements and $\mathcal{O}(k^3)$ sets. However, as we explain now, we cannot use the kernel given by Abu-Khzam [1] directly for our purposes. This is in contrast to the fact that one can use the best known parameterized algorithms for 3-Set Packing to design parameterized algorithms for Rainbow Matching. Given an instance (G, χ, k) of Rainbow Matching, we can transform it to an instance $(U, \mathcal{F}, k)$ of 3-Set Packing as explained in the introduction. Now if we apply a kernelization algorithm for 3-Set Packing, then it will return an equivalent instance $(U', \mathcal{F}', k')$ of 3-Set Packing and *not of* Rainbow Matching. A priori, it is not clear how we can transform $(U', \mathcal{F}', k')$ to an instance of Rainbow Matching without increasing the size bounds on the kernel we obtain for Rainbow Matching. Thus, to design a kernelization algorithm for Rainbow Matching we give a kernelization algorithm for 3-Set Packing such that it is easy to transform an instance of the latter to an instance of Rainbow Matching. This kernel is given here mainly for *completeness*.

4.1 Kernelization for Rainbow Matching on general graphs

Now we give the kernelization algorithm alluded to in the first part of Theorem 3. Towards that we will use the sunflower lemma – a classical result of Erdős and Rado. We first define the terminology used in the statement of the lemma. A *sunflower* with k *petals* and a *core* Y is a collection of sets $S_1, \ldots, S_k$ such that $S_i \cap S_j = Y$ for all $i \neq j$; the sets $S_i \setminus Y$ are petals and we require *none of them to be empty*. Note that a family of pairwise disjoint sets is a sunflower (with an empty core).

▶ **Proposition 11.** [3, pg 38] (Sunflower lemma) *Let $\mathcal{A}$ be a family of sets (without duplicates) over a universe U, such that each set in $\mathcal{A}$ has cardinality exactly d. If $|\mathcal{A}| > d!(k-1)^d$, then $\mathcal{A}$ contains a sunflower with k petals and such a sunflower can be computed in time polynomial in $|\mathcal{A}|$, $|U|$, and k.*

Proof of first part of Theorem 3. Given an instance (G, χ, k) of Rainbow Matching, we view this as an instance $\mathcal{J}$ of 3-Set Packing as follows: $U = V(G) \cup \{1, \ldots, q\}$ and $\mathcal{F}$ consists of a set $\{u, v, \chi(uv)\}$ corresponding to every edge $e = uv \in E(G)$.

▶ **Reduction Rule 12.** *Let $(U, \mathcal{F}, k)$ be an instance of* 3-Set Packing *and suppose that $\mathcal{F}$ contains a sunflower $S = \{S_1, \ldots, S_{3(k-1)+2}\}$ of cardinality $3k-1$ with core Y. Then, return $(U', \mathcal{F}', k)$, where $U' = \bigcup_{X \in \mathcal{F}'} X$, and $\mathcal{F}' = (\mathcal{F} \setminus S_1)$ is obtained by deleting a set S_1 from $\mathcal{F}$.*

To show the correctness of Reduction Rule 12, we need to show the following lemma.

▶ **Lemma 13.** *Reduction Rule 12 is safe.*

Proof. We will prove that $(U, \mathcal{F}, k)$ is a yes-instance of 3-Set Packing if and only if $(U', \mathcal{F}', k)$ is a yes-instance of 3-Set Packing. It is clear that if $(U', \mathcal{F}', k)$ is a yes-instance of 3-Set Packing then so is $(U, \mathcal{F}, k)$; so the backward direction holds straightaway.

For the forward direction, we assume that we have a solution $\mathcal{S}$ to $(U, \mathcal{F}, k)$, *i.e.*, a set of k pairwise disjoint sets. If $\mathcal{S}$ does not contain S_1, then it is also a solution for $(U', \mathcal{F}', k)$. So let us assume that $S_1 \in \mathcal{S}$. Observe that the number of elements appearing in the sets in $\mathcal{S}$, apart from those present in S_1, is $3(k-1)$. Also, note that no set in $\mathcal{S} \setminus \{S_1\}$ intersects the

core Y. Thus, the number of sets in the sunflower S that intersects the elements present in the sets of $\mathcal{S}$ is upper bounded by $1 + 3(k-1)$ (the first one for S_1). This implies there exists a set $S^\star \in S$ that is pairwise disjoint with every set in $\mathcal{S} \setminus \{S_1\}$. Thus, $(\mathcal{S} \setminus \{S_1\}) \cup S^\star$ is a solution of size k for $(U', \mathcal{F}', k)$. This completes the proof. ◀

Now, we are ready to describe the kernelization algorithm. If the number of sets in $\mathcal{F}$ is more than $6(3k-2)^3$, then the kernelization algorithm applies the sunflower lemma to find a sunflower of size $3k-1$, and applies Reduction Rule 12 on this sunflower.

The algorithm applies this procedure exhaustively, and obtains a new family of sets $\mathcal{F}'$ of size at most $6(3k-2)^3$. This concludes the size bound on the family $(U', \mathcal{F}', k)$ of 3-Set Packing. Observe that throughout the process, we have never reduced the size of any set in the family and each set in the family still corresponds to an edge and its color. Thus, given $(U', \mathcal{F}', k)$, let W be the vertices present in U'. Then we return $(G[W], \chi', k)$, where edge coloring χ' is the restriction of χ to the edges present in $G[W]$. This concludes the description of the kernelization algorithm. ◀

4.2 A Kernel on graphs of bounded degree

In this section we design a small kernel for Rainbow Matching on graphs of bounded degree. Let (G, χ, k) be an instance of Rainbow Matching. Throughout this section we assume that the *maximum degree* of G is upper bounded by a fixed constant d.

For $i \in [q]$, let $E_i = \{e \in E(G) \mid \chi(e) = i\}$. We call the set of edges E_i as a *color class with color i*. Next we give reduction rule that bounds the size of each color class.

▶ **Reduction Rule 14.** *If there exists $i \in [q]$ such that $|E_i| \geq 2d(k-1)+1$ then delete E_i and reduce k by 1. That is, we obtain an instance $(G', \chi', k-1)$. Here, G' is obtained by deleting all the edges in E_i and edge coloring χ' is obtained by restricting χ to the edges in G'.*

▶ **Lemma 15.** *Reduction Rule 14 is safe.*

Proof. We will prove that G has a colorful matching of size k if and only if G' has a colorful matching of size $k-1$. We first prove the forward direction. If G has a colorful matching $\mathcal{M}$ of size k that contains an edge $uv \in E_i$, then $\mathcal{M} \setminus \{uv\}$ is a colorful matching of size $k-1$ in the graph G'. If $\mathcal{M}$ does not have an edge of color i, then $\mathcal{M}$ itself is a colorful matching of size at least $k-1$ for G'.

For the backward direction, let $\mathcal{M}'$ be a colorful matching of size $k-1$ of G'. Observe that $\mathcal{M}'$ has $2(k-1)$ distinct vertices and each vertex has degree at most d. If every edge in $\mathcal{M}'$ has an endpoint that is adjacent to an edge in E_i, then the vertices in $\mathcal{M}'$ can share at most $2d(k-1)$ vertices. That is, at most $2d(k-1)$ edges from E_i can share vertices with $\mathcal{M}'$. Since, $|E_i| \geq 2d(k-1)+1$, E_i has at least one edge that does not share a vertex with $\mathcal{M}'$. Let that edge be uv. Then, it follows that $\mathcal{M}' \cup \{uv\}$ is a k size colorful matching of G, and our proof is complete. ◀

We apply Reduction Rule 14 exhaustively. If the premise of the rule is not satisfied, then for each color i, we have that $|E_i| \leq 2d(k-1)$. Next we give a polynomial time procedure that either outputs a colorful matching of size at least k or bounds the number of colors.

▶ **Lemma 16.** *Let (G, χ, k) be an instance of* Rainbow Matching *for which Reduction Rule 14 is not applicable. Then, in polynomial time either we can conclude that (G, χ, k) is a yes-instance or the number of distinct colors in the instance is upper bounded by $2d(k-1)$.*

Proof. We iteratively try to build a colorful matching of size k. If we fail to do so, then it will enable us to bound the number of color classes. Let $\mathcal{M}$ be an empty set. We repeat the below procedure until the graph is empty.

1. Pick an edge uv of G arbitrarily, and add it to $\mathcal{M}$. Let the edges incident on u have colors $c_u^1, c_u^2, \ldots, c_u^\ell$ and the edges incident on v have colors $c_v^1, c_v^2, \ldots, c_v^p$.
2. Delete all the edges in $\bigcup_{i=1}^{\ell} E_{c_u^i}$ and $\bigcup_{i=1}^{p} E_{c_v^i}$. Let the resulting graph be also called G.

If we can continue the above process for k steps (*i.e.* $|\mathcal{M}| \geq k$) then $\mathcal{M}$ is a colorful matching of size at least k. In this case we output $\mathcal{M}$ as the desired colorful matching. To see its correctness, observe that in every iteration we deleted the edges incident on both the endpoints of the added to $\mathcal{M}$. So, the edges we added to $\mathcal{M}$ are indeed pairwise vertex disjoint. Also, note that we delete all the edges with colors that are used on the edges that are incident to the edges that were added to $\mathcal{M}$. Hence, the edges in $\mathcal{M}$ have distinct color.

Otherwise, our procedure ends within at most $k-1$ steps, and so $|\mathcal{M}| \leq k-1$. Now, let us count the number of color classes we delete in each iteration. In other words, we count the number of color classes that are deleted each time we add an edge to M. If all the edges incident on u have distinct colors then $\ell \leq d$ because degree of u is at most d. Similarly, we are argue that $p \leq d$. Together they imply that we delete at most $2d$ color classes in each iteration. Hence, in at most $k-1$ iterations we delete at most $2d(k-1)$ color classes. Following this we are left with an an empty graph. Thus, we have shown that in this case, we can have at most $2d(k-1)$ color classes. ◀

Lemmas 15 and 16 together prove second part of Theorem 3.

5 Conclusion, Discussion and Open Problems

In this paper, we considered Rainbow Matching from the viewpoint of parameterized complexity, and designed faster parameterized algorithms as well as kernels for this problem. Rainbow Matching is easily seen as a generalization of another well studied problem in parameterized algorithms, namely 3-Dimensional Matching, when we allow the input graph to be a multigraph. In this problem, we are given a set family $(U, \mathcal{F})$, together with a partition $U = \uplus_{i=1}^{3} U_i$ and a positive integer k. Here, every set $F \in \mathcal{F}$ has the property that for all $i \in [3]$, $|F \cap U_i| = 1$. The question is whether there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ containing k pairwise-disjoint sets. We first show that Rainbow Matching is indeed a generalization of 3-Dimensional Matching. Towards proving this, we give a polynomial time parameter-preserving reduction from 3−Dimensional Matching to Rainbow Matching. That is, we give the following ppt reduction, 3−Dimensional Matching $\leq_{\text{ppt}}$ Rainbow Matching. Here, let us only present a rough sketch of the proof. The idea of the construction is as follows. In the bipartite graph of the constructed instance of Rainbow Matching, one side represents the elements of U_1, and the other side represents the elements of U_2. Then, for every set $\{u_1, u_2, u_3\}$ in $\mathcal{F}$, where $u_i \in U_i$ for all $i \in [3]$, we add an edge between u_1 and u_2 whose color is u_3. It is easy to see that a solution for the original problem instance can be directly translated to a solution for the new problem instance, and vice versa. Moreover, the parameter k in both instances is set to be the same.

We also saw that there is a ppt reduction from Rainbow Matching to 3-Set Packing and thus we have the following chain of reductions.

3−Dimensional Matching $\leq_{\text{ppt}}$ Rainbow Matching $\leq_{\text{ppt}}$ 3−Set Packing.

It is known that 3-Dimensional Matching admits a randomized algorithm with running time $\mathcal{O}^\star(2^k)$ [2] and a deterministic algorithm with running time $\mathcal{O}^\star(2.5961^{2k}) = \mathcal{O}^\star(6.7398^k)$ [19]. We gave in the introduction a deterministic algorithm for Rainbow Matching that is the same as the one for 3-Set Packing. However, we remark that the algorithm for 3-Dimensional Matching given in [19] can actually be used to solve Rainbow Matching in $\mathcal{O}^\star(6.7398^k)$ time. Can we design a faster randomized or deterministic algorithm for Rainbow Matching or even 3-Dimensional Matching?

We gave an $\mathcal{O}(k^2)$ kernel on paths. Does there exist a linear kernel on paths? Could we get improved kernel for Rainbow Matching on simple family of graphs such as trees, graphs of constant treewidth or planar graphs. Could we show that $\mathcal{O}(k^3)$ size bound on the kernel for Rainbow Matching is optimal?

Finally, by a direct application of our randomized parameterized algorithm for Rainbow Matching running in time $\mathcal{O}^\star(2^k)$, we have that there exists a randomized algorithm for Rainbow Matching running in time $\mathcal{O}^\star(2^{n/2}) = \mathcal{O}^\star(1.4143^n)$. Here, n is the number of vertices in the input graph and the $n/2$ is an upper bound on the maximum size of a colorful matching in a graph. Using a simple dynamic programming algorithm, it is possible to design a $\mathcal{O}^\star(2^n)$ algorithm for Rainbow Matching. Designing a deterministic algorithm for Rainbow Matching running in time $(2-\epsilon)^n$ for some fixed $\epsilon > 0$ is another interesting open problem.

References

1 F. N. Abu-Khzam. An improved kernelization algorithm for r-set packing. *Information Processing Letters*, 110:621–624, 2010.

2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *Journal of Computer and System Sciences*, 87:119–139, 2017.

3 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

4 H. Dell and D. Marx. Kernelization of packing problems. In *SODA'12*, 2012.

5 R. G. Downey and M. R. Fellows. *Fundamentals of parameterized complexity*. Springer, 2013.

6 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

7 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009.

8 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

9 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

10 J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Computing*, 2:225–231, 1973.

11 Alon Itai, Michael Rodeh, and Steven L. Tanimoto. Some matching problems for bipartite graphs. *J. ACM*, 25(4):517–525, 1978.

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

13 Van Bang Le and Florian Pfender. Complexity results for rainbow matchings. *Theor. Comput. Sci.*, 524:27–33, 2014.

14 László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

15 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27, 1980.

16 Herbert J Ryser. Neuere probleme der kombinatorik. *Vorträge über Kombinatorik, Oberwolfach*, pages 69–91, 1967.

17 Larry J. Stockmeyer and Vijay V. Vazirani. Np-completeness of some generalizations of the maximum matching problem. *Inf. Process. Lett.*, 15(1):14–19, 1982.

18 Mihalis Yannakakis and Fanica Gavril. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, 1980.

19 Meirav Zehavi. Mixing color coding-related techniques. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 1037–1049, 2015.

Compositional Weak Metrics for Group Key Update

Ruggero Lanotte[1], Massimo Merro[2], and Simone Tini[3]

1 **Dipartimento di Scienze e Alta Tecnologia, Università dell'Insubria, Italy**
`ruggero.lanotte@uninsubria.it`
2 **Dipartimento di Informatica, Università degli Studi di Verona, Italy**
`massimo.merro@univr.it`
3 **Dipartimento di Scienze e Alta Tecnologia, Università dell'Insubria, Italy**

Abstract

We investigate the compositionality of both weak bisimilarity metric and weak similarity quasi-metric semantics with respect to a variety of standard operators, in the context of probabilistic process algebra. We show how compositionality with respect to nondeterministic and probabilistic choice requires to resort to rooted semantics. As a main application, we demonstrate how our results can be successfully used to conduct compositional reasonings to estimate the performances of group key update protocols in a multicast setting.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases Behavioural metric, compositional reasoning, group key update

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.72

1 Introduction

Behavioural distances [35, 17, 14] allow us to compare the behaviour of probabilistic systems. Basically, they are the quantitative analog of the classical notions of *behavioural equivalence* and *preorder*. In the *weak semantic* approach, where *non-observable* actions are abstracted away, *weak bisimilarity metric* [18] and its asymmetric counterpart, *weak similarity quasi-metric* [30], have been proposed as the quantitative analog of *weak probabilistic bisimilarity* and *weak probabilistic similarity*, respectively [5, 4].

In order to specify and verify systems in a compositional manner, it is necessary to work with behavioural semantics which are preserved by all operators of the language. In this light, different forms of compositionality have been proposed for strong bisimilarity metrics by adopting different notions of *uniform continuity* [20]. Intuitively, a uniformly continuous operator ensures that a small variation in the behaviour of a system component leads to a smooth and bounded variation in the behaviour of the whole system (absence of chaotic behaviour when system components and parameters are modified in a controlled manner). More precisely, the uniform continuity of an n-ary process algebra operator f ensures that, once fixed the maximal tolerable distance ε between processes $f(s_1,\ldots,s_n)$ and $f(s'_1,\ldots,s'_n)$, there are values δ_i such that, whenever the distance between process arguments s_i and s'_i is below δ_i, for $1 \leq i \leq n$, then the distance between $f(s_1,\ldots,s_n)$ and $f(s'_1,\ldots,s'_n)$ is guaranteed to be below ε. The notions of uniform continuity considered in [20] are: (i) *non-extensiveness*, requiring $\varepsilon = \max(\delta_1,\ldots,\delta_n)$, (ii) *non-expansiveness*, with $\varepsilon = \delta_1+\ldots+\delta_n$, and (iii) *Lipschitz continuity*, where $\varepsilon = L\cdot(\delta_1+\ldots+\delta_n)$, for some $L \in \mathbb{R}_{\geq 1}$.

In this paper, we extend and generalise the work of [20] to *rooted* and *asymmetric* semantics. In particular, for all standard operators of probabilistic process algebras, such as

probabilistic CCS [24] and probabilistic CSP [24], we derive the notions of uniform continuity which are satisfied by *rooted* bisimilarity metric and/or *rooted* similarity quasimetric. It is well-known that weak probabilistic bisimilarity, unlike similarity, is not preserved by nondeterministic choice. Thus, with no surprise, the nondeterministic choice operator is not uniformly continuous with respect to weak bisimilarity metric, while it is uniformly continuous with respect to weak similarity quasimetric. In this paper, we show that probabilistic choice is uniformly continuous with respect to neither weak bisimilarity metric nor weak similarity quasimetric. Thus, in order to recover uniform continuity, we work with rooted (bi)similarities [36], where, in first step of the (bi)simulation game any *strong* transition must be matched by the same *strong* transition.

As main case study, we consider an abstract specification of the *group key update* protocol. In this protocol, whenever a principal joins or leaves the group, in order to guarantee *backward* and *forward confidentiality*, it is necessary to generate and distribute a new group key. However, this operation has a cost in terms of:

(i) the number of attacks aiming at compromising the group key,
(ii) the degradation of communication service,
(iii) battery consumption.

We show how our compositional theory can be used to estimate the distance between the ideal protocol, where groups cannot dynamically change, and some variations of the protocol obtained by playing with the following parameters:

(i) number of principals,
(ii) probability that principals leave the group,
(iii) probability that principals join the group.

The results of our analysis allow us to assert that the protocol under consideration has good efficiency in groups with low dynamicity, regardless of the size of the group.

Outline. Section 2 provides background on probabilistic semantics. Section 3 contains the main results on uniform continuity. Section 4 applies our theory to an abstract group key update protocol. Section 5 concludes and discusses related and future work.

2 Preliminaries

Nondeterministic probabilistic labelled transition systems (pLTSs) [33] represent a very general semantic model for probabilistic processes as they combine LTSs [26] and discrete time Markov chains [34, 23], to model reactive behaviour, nondeterminism and probability. The state space is given by a *signature* Σ consisting of both a set of *operators* and a *rank function* r, where $\mathsf{r}(f)$ returns the arity of the operator f. The set $\mathsf{T}(\Sigma)$ of *terms* over Σ, or *processes*, is the least set such that $f(t_1, \ldots, t_n) \in \mathsf{T}(\Sigma)$ whenever $f \in \Sigma$, $\mathsf{r}(f) = n$ and $t_1, \ldots, t_n \in \mathsf{T}(\Sigma)$. Notice that $\mathsf{T}(\Sigma) \neq \emptyset$ if and only if Σ contains *constants*, i.e., functions with arity 0. We write $\Delta(\mathsf{T}(\Sigma))$ to denote the set of all probability distributions with *finite support* over $\mathsf{T}(\Sigma)$, which are mappings $\pi \colon \mathsf{T}(\Sigma) \to [0, 1]$, with $\sum_{t \in \mathsf{T}(\Sigma)} \pi(t) = 1$.

▶ **Definition 1** (pLTS [33]). A *nondeterministic probabilistic labelled transition system* (*pLTS*) is given by a triple $P = (\mathsf{T}(\Sigma), \mathit{Act}, \to)$ where:

(i) Σ is a signature,
(ii) Act is a countable set of *actions*, and
(iii) $\to \subseteq \mathsf{T}(\Sigma) \times \mathit{Act} \times \Delta(\mathsf{T}(\Sigma))$ is a *transition relation*.

As usual, we write $t \xrightarrow{\alpha} \pi$ for $(t, \alpha, \pi) \in \rightarrow$. Let $der(t, \alpha) = \{\pi \in \Delta(\mathsf{T}(\Sigma)) \mid t \xrightarrow{\alpha} \pi\}$ be the set of the *derivatives* of t according to action α. We say that a pLTS P is *image finite* if $der(t, \alpha)$ is finite for all $t \in \mathsf{T}(\Sigma)$ and $\alpha \in Act$.

We consider a signature that contains many of the operators from probabilistic CCS and probabilistic CSP specified via the SOS rules in Table 1–3. The operators we consider are:

1. constants $\mathsf{0}$ (idle process) and ε (skip process);
2. a family of n-ary probabilistic prefix operators $\alpha.([p_1]\_ \oplus \ldots \oplus [p_n]\_)$ with $\alpha \in Act$, $n \geq 1$, $p_1, \ldots, p_n \in (0, 1]$ and $\sum_{i=1}^{n} p_i = 1$;
3. nondeterministic choice $\_ + \_$;
4. action restriction $(\nu\, \alpha)\_$ with $\alpha \in Act \setminus \{\tau, \surd\}$;
5. sequential composition $\_\,;\_$;
6. CSP-like parallel composition $\_ \parallel_B \_$, with $B \subseteq Act \setminus \{\tau, \surd\}$,
7. CCS-like parallel composition $\_ \mid \_$, which assumes a function $^-\colon Act \setminus \{\tau, \surd\} \rightarrow Act \setminus \{\tau, \surd\}$ with $\overline{\overline{a}} = a$,
8. probabilistic choice $\_ +_p \_$;
9. finite iteration $\_^n$,
10. finite replication $!^n\_$,
11. infinite iteration $\_^\omega$,
12. binary Kleene-star iteration $\_^*\_$,
13. infinite replication (bang) operator $!\_$, and
14. probabilistic bang operator $!_p\_$.

All rules in Table 1–3 obey to the *PGSOS format* [9, 10]. We assume a set of actions $Act = A \cup \{\tau, \surd\}$, with $\surd$ denoting the *successful* termination action, and τ denoting *non-observable* action. We let $\alpha, \beta, \ldots$ range over Act and $a, b, \ldots$ over $Act \setminus \{\tau\}$. The rules of Table 1–3 assume a set of *process variables*, ranged over by x, y and a set of *distribution variables*, ranged over by μ, ν, allowing us to generalise the notions of term and distribution to *open term* and *open distribution* in the standard way. The rules are then defined by using *open transitions*, such as $x \mid y \xrightarrow{a} \mu \mid \nu$, taking open terms to open distributions. The PGSOS rules rely on some notations and operations on distributions. For $t \in \mathsf{T}(\Sigma)$, $\delta(t)$ denotes the *Dirac distribution*, defined by $(\delta(t))(t) = 1$. The convex combination $\sum_{i\in I} p_i\pi_i$ of a finite set of distributions $\{\pi_i\}_{i\in I}$, with $p_i \in (0, 1]$ and $\sum_{i\in I} p_i = 1$, is defined by $(\sum_{i\in I} p_i\pi_i)(t) = \sum_{i\in I}(p_i\pi_i(t))$. We write $\pi \oplus_p \pi'$ for $p\pi + (1-p)\pi'$. For $f \in \Sigma$ and $\pi_i \in \Delta(\mathsf{T}(\Sigma))$, $f(\pi_1, \ldots, \pi_n)$ denotes the product distribution defined by $f(\pi_1, \ldots, \pi_n)(f(t_1, \ldots, t_n)) = \prod_{i=1}^{n} \pi_i(t_i)$. Notice that all distributions defined in this inductive way have *finite support*.

▶ **Definition 2** (PGSOS-TSS [6, 9]). A *PGSOS-transition system specification* (*PGSOS-TSS*) is a triple $T = (\Sigma, Act, R)$ where:

- **(i)** Σ is a signature,
- **(ii)** Act is a countable set of actions,
- **(iii)** R is a countable set of PGSOS rules,
- **(iv)** for each $f \in \Sigma$ and $\alpha \in Act$, the set of rules with conclusion of the form $f(x_1, \ldots, x_n) \xrightarrow{\alpha} \theta$ is finite.

We recall that *closed substitutions* map process variables to processes, and distribution variables to distributions. Closed substitutions allows us to derive the *supported model* of a TSS, namely a pLTS in which the transition relation $\rightarrow$ contains all and only those transitions inductively derived by the SOS rules [7, 6, 9]. Item (4) in Definition 2 ensures that the supported model of a TSS is always image finite.

▶ **Definition 3** (Disjoint extension [1]). Let $T_1 = (\Sigma_1, A, R_1)$ and $T_2 = (\Sigma_2, A, R_2)$ be two PGSOS-TSSs. We say that T_2 is a *disjoint extension* of T_1, written $T_1 \sqsubseteq T_2$, iff $\Sigma_1 \subseteq \Sigma_2$, $R_1 \subseteq R_2$ and R_2 introduces no new rule for any operator in Σ_1.

2.1 Weak behavioural distances

In this section, we give the formal definitions of the weak behavioural distances of [18, 30].

The definition of weak transitions $\stackrel{\alpha}{\Longrightarrow}$, which abstract away non-observable actions, is complicated by the fact that transitions take processes to distributions. Following [16], we need to generalise transitions, so that they take sub-distributions to sub-distributions. With an abuse of notation, we use π, π' to range also over sub-distributions, admitting $\sum_{t \in \mathsf{T}(\Sigma)} \pi(t) \leq 1$. For a term t and a distribution π, we write $t \xrightarrow{\hat{\tau}} \pi$ if $t \xrightarrow{\tau} \pi$ or $\pi = \delta(t)$. Then, for $a \in A$, we write $t \xrightarrow{\hat{a}} \pi$ if $t \xrightarrow{a} \pi$. Relation $\xrightarrow{\hat{\alpha}}$ is extended to model transitions from sub-distributions to sub-distributions. For a sub-distribution $\pi = \sum_{i \in I} p_i \delta(t_i)$, we write $\pi \xrightarrow{\hat{\alpha}} \pi'$ if there is a set $J \subseteq I$ with $t_j \xrightarrow{\hat{\alpha}} \pi_j$ for all $j \in J$, $t_i \not\xrightarrow{\hat{\alpha}}$, for all $i \in I \setminus J$, and $\pi' = \sum_{j \in J} p_j \pi_j$. If $\alpha \neq \tau$ then this definition entails that only some terms in the support of π have the $\xrightarrow{\hat{\alpha}}$ transition. Then, we define the weak transition relation $\stackrel{\hat{\tau}}{\Longrightarrow}$ as the transitive and reflexive closure of $\xrightarrow{\hat{\tau}}$, i.e. $\stackrel{\hat{\tau}}{\Longrightarrow} = (\xrightarrow{\hat{\tau}})^*$, while for $a \in A$ we let $\stackrel{\hat{a}}{\Longrightarrow}$ denote $\stackrel{\hat{\tau}}{\Longrightarrow}\xrightarrow{\hat{a}}\stackrel{\hat{\tau}}{\Longrightarrow}$.

Weak bisimilarity metric [18] (resp. weak similarity quasimetric [30]) is defined as a pseudometric (resp. pseudoquasimetric) measuring the tolerance of the probabilistic weak bisimilarity (resp. probabilistic weak similarity).

▶ **Definition 4** (Pseudoquasimetrics and pseudometrics). A function $d\colon \mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma) \to [0,1]$ is said to be a *1-bounded pseudoquasimetric* when:
(i) $d(t,t) = 0$, for all $t \in \mathsf{T}(\Sigma)$, and
(ii) $d(t,t') \leq d(t,t'') + d(t'',t')$, for all $t, t', t'' \in \mathsf{T}(\Sigma)$.
If it is also symmetric, i.e. $d(t,t') = d(t',t)$, for all $t, t' \in \mathsf{T}(\Sigma)$, then it is said to be a *1-bounded pseudometric*.

We need to lift these two definitions to (sub)distributions. To this end, as in [30], we rely on the notions of *matching* [37] (also known as *coupling*) and *Kantorovich lifting* [25]. The original formulation in [18] is technically different, but equivalent [15].

▶ **Definition 5** (Matching). A *matching* for a pair of distributions (π, π') is a distribution ω in the product space $\mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma)$ with $\sum_{t' \in \mathsf{T}(\Sigma)} \omega(t,t') = \pi(t)$, for $t \in \mathsf{T}(\Sigma)$, and $\sum_{t \in \mathsf{T}(\Sigma)} \omega(t,t') = \pi'(t')$, for $t' \in \mathsf{T}(\Sigma)$. Let $\Omega(\pi, \pi')$ be the set of all matchings for (π, π').

▶ **Definition 6** (Kantorovich lifting). Let $d\colon \mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma) \to [0,1]$ be a pseudo(quasi)metric. The *Kantorovich lifting* of d is the function $\mathbf{K}(d)\colon \Delta(\mathsf{T}(\Sigma)) \times \Delta(\mathsf{T}(\Sigma)) \to [0,1]$ defined as:

$$\mathbf{K}(d)(\pi, \pi') \;=\; \min_{\omega \in \Omega(\pi,\pi')} \sum_{t,t' \in \mathsf{T}(\Sigma)} \omega(t,t') \cdot d(t,t').$$

Note that since we are considering only distributions with finite support, the minimum over the set of matchings $\Omega(\pi, \pi')$ is well defined.

Now, we are ready to define our behavioural distances. They are parametric on a *discount factor* $\lambda \in (0,1]$ which mitigates the (bi)simulation tolerance on future activities [12, 17].

▶ **Definition 7** (Weak behavioural distances). Let $|\pi|$ be an abbreviation for $\sum_{t \in \mathsf{T}(\Sigma)} \pi(t)$. We say that a pseudoquasimetric $d\colon \mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma) \to [0,1]$ is a *weak simulation quasimetric* if for all $s, t \in \mathsf{T}(\Sigma)$, with $d(s,t) < 1$, whenever $s \xrightarrow{\alpha} \pi_s$ there is a sub-distribution π_t such that $t \stackrel{\hat{\alpha}}{\Longrightarrow} \pi_t$ and $\lambda \cdot \mathbf{K}(d)(\pi_s, \pi_t + (1 - |\pi_t|)\mathsf{0}) \leq d(s,t)$. Moreover, if d is a pseudometric, then d is a *weak bisimulation metric*.

In the previous definition, if $|\pi_t| < 1$ then, with probability $1 - |\pi_t|$, there is no way to simulate the behaviour of any process different from $\mathsf{0}$ in the support of π_s. We remark that the kernel of a weak bisimulation pseudometric is a weak probabilistic bisimulation [18] and the kernel of a weak simulation pseudoquasimetric is a weak probabilistic simulation [30].

Crucial results are the existence of both the minimal weak bisimulation metric [18], called *weak bisimilarity metric*, and denoted with $\mathbf{d}^{\mathsf{m}}$, and the minimal weak simulation quasimetric [30], called *weak similarity quasimetric*, and denoted with $\mathbf{d}^{\mathsf{q}}$.

3 Uniform continuity for rooted (quasi)metric semantics

In this section, we show that the operators in Table 1–3 allow for compositional reasoning with respect to a *rooted* variant of our weak behavioural distances. We start by recalling the notion of uniform continuity, whose intuitive meaning was discussed in the Introduction.

▶ **Definition 8** (Modulus of continuity)**.** Let $T = (\Sigma, Act, R)$ be a TSS, $f \in \Sigma$ an n-ary operator, and $d\colon \mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma) \to [0,1]$ a function. A mapping $\omega_f\colon [0,1]^n \to [0,1]$ is a *modulus of continuity* for f with respect to d when:

- $d(f(s_1,\ldots,s_n), f(t_1,\ldots,t_n)) \leq \omega_f(d(s_1,t_1),\ldots,d(s_n,t_n))$, for all processes $s_i, t_i \in \mathsf{T}(\Sigma)$;
- ω_f is continuous at $(0,\ldots,0)$, i.e. $\lim_{(\epsilon_1,\ldots,\epsilon_n)\to(0,\ldots,0)} \omega(\epsilon_1,\ldots,\epsilon_n) = \omega(0,\ldots,0)$;
- $\omega_f(0,\ldots,0) = 0$.

▶ **Definition 9** (Uniformly continuous operator [20])**.** Let $T = (\Sigma, A, R)$ be a TSS and $d\colon \mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma) \to [0,1]$. We say that an operator $f \in \Sigma$ is:

- *uniformly continuous with respect to d* if f admits some modulus of continuity wrt. d;
- *L-Lipschitz continuous with respect to d*, for $L \in \mathbb{R}_{\geq 1}$, if $\omega_f(\epsilon_1,\ldots,\epsilon_n) = L \cdot \sum_{i=1}^{n} \epsilon_i$ is a modulus of continuity for f with respect to d;
- *non-expansive with respect to d* if f is 1-Lipschitz continuous with respect to d;
- *non-extensive with respect to d* if $\omega_f(\epsilon_1,\ldots,\epsilon_n) = \max_{i=1}^{n} \epsilon_i$ is a modulus of continuity for f with respect to d.

As expected, since τ-transitions may solve nondeterminism, $\mathbf{d}^{\mathsf{m}}$ is not uniformly continuous with respect to $+$, thus requiring to introduce a rooted version for $\mathbf{d}^{\mathsf{m}}$. For instance, $\mathbf{d}^{\mathsf{m}}(\tau.a.\mathsf{0}, a.\mathsf{0}) = 0$ but $\mathbf{d}^{\mathsf{m}}(\tau.a.\mathsf{0} + b.\mathsf{0}, a.\mathsf{0} + b.\mathsf{0}) = \lambda$, thus implying that no modulus of continuity for operator $+$ with respect to $\mathbf{d}^{\mathsf{m}}$ can be defined. Interestingly, in the metric context also the asymmetric simulation-like approach requires rootedness. Indeed, $\mathbf{d}^{\mathsf{q}}$ is not continuous with respect to $+_p$. For instance, $\mathbf{d}^{\mathsf{q}}(\tau.a.b.\mathsf{0}, a.b.\mathsf{0}) = 0$, but $\mathbf{d}^{\mathsf{q}}(\tau.a.b.\mathsf{0} +_p a.\mathsf{0}, a.b.\mathsf{0} +_p a.\mathsf{0}) = \lambda^2(1-p)$, thus implying that no modulus of continuity for $+_p$ wrt. $\mathbf{d}^{\mathsf{q}}$ can be defined. In fact, transition $\tau.a.b.\mathsf{0} +_p a.\mathsf{0} \xrightarrow{\tau} \delta(a.b.\mathsf{0})$ can be simulated only by $a.b.\mathsf{0} +_p a.\mathsf{0} \stackrel{\tau}{\Longrightarrow} \delta(a.b.\mathsf{0} +_p a.\mathsf{0})$, then $\lambda\,\mathbf{K}(\mathbf{d}^{\mathsf{q}})(\delta(a.b.\mathsf{0}), \delta(a.b.\mathsf{0} +_p a.\mathsf{0})) = \lambda \mathbf{d}^{\mathsf{q}}(a.b.\mathsf{0}, a.b.\mathsf{0} +_p a.\mathsf{0}) = \lambda^2(1-p)$. Notice that we also have that $\mathbf{d}^{\mathsf{m}}(\tau.a.b.\mathsf{0}, a.b.\mathsf{0}) = 0$ and $\mathbf{d}^{\mathsf{m}}(\tau.a.b.\mathsf{0} +_p a.\mathsf{0}, a.b.\mathsf{0} +_p a.\mathsf{0}) \geq \lambda^2(1-p)$. This implies that $\mathbf{d}^{\mathsf{m}}$, like $\mathbf{d}^{\mathsf{q}}$, is not uniformly continuous with respect to $+_p$.

▶ **Definition 10** (Rooted behavioural distances)**.** We say that a pseudoquasimetric $r\colon \mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma) \to [0,1]$ is a *rooted simulation quasimetric* if there exists a weak simulation quasimetric d such that for all $s, t \in \mathsf{T}(\Sigma)$, with $r(s,t) < 1$, whenever $s \xrightarrow{\alpha} \pi_s$ there is a distribution π_t such that $t \xrightarrow{\alpha} \pi_t$ and $\lambda \cdot \mathbf{K}(d)(\pi_s, \pi_t) \leq r(s,t)$. Moreover, if both r and d are pseudometrics, then r is a *rooted bisimulation metric*.

Table 1 Non-extensive operators

$$\frac{}{\varepsilon \xrightarrow{\checkmark} \delta(\mathsf{0})} \qquad \frac{}{\alpha.\bigoplus_{i=1}^{n}[p_i]x_i \xrightarrow{\alpha} \sum_{i=1}^{n} p_i\delta(x_i)} \qquad \frac{x \xrightarrow{\alpha} \mu}{x+y \xrightarrow{\alpha} \mu} \qquad \frac{y \xrightarrow{\alpha} \nu}{x+y \xrightarrow{\alpha} \nu}$$

$$\frac{x \xrightarrow{\alpha} \mu \quad y \not\xrightarrow{\alpha}}{x +_p y \xrightarrow{\alpha} \mu} \qquad \frac{x \not\xrightarrow{\alpha} \quad y \xrightarrow{\alpha} \nu}{x +_p y \xrightarrow{\alpha} \nu} \qquad \frac{x \xrightarrow{\alpha} \mu \quad y \xrightarrow{\alpha} \nu}{x +_p y \xrightarrow{\alpha} \mu \oplus_p \nu} \qquad \frac{x \xrightarrow{\alpha} \mu \text{ and } \alpha \notin \{\beta, \overline{\beta}\}}{(\nu\,\beta)\,x \xrightarrow{\alpha} \mu}$$

▶ **Theorem 11.** *There exists a rooted simulation quasimetric* $\mathbf{r}^{\mathsf{q}}$ *(resp. rooted bisimulation metric* $\mathbf{r}^{\mathsf{m}}$*) such that* $\mathbf{r}^{\mathsf{q}}(s,t) \leq r(s,t)$ *(resp.* $\mathbf{r}^{\mathsf{m}}(s,t) \leq r(s,t)$*) for all rooted simulation quasimetrics (resp. rooted bisimulation metrics)* r *and all processes* $s,t \in \mathcal{T}(\Sigma)$*.*

We call $\mathbf{r}^{\mathsf{q}}$ *rooted similarity quasimetric*, and $\mathbf{r}^{\mathsf{m}}$ *rooted bisimilarity metric.*

In the following, for each operator, we compute a suitable upper bound on the rooted simulation and bisimulation tolerance between processes composed by that operator, then we use this bound to infer its compositionality property. Basically, our goal is to express a bound on the rooted (bi)simulation tolerance between composed processes $f(s_1,\ldots,s_n)$ and $f(t_1,\ldots,t_n)$ in terms of the tolerance between the components s_i and t_i.

Non-extensive operators. Consider the TSS $T_{\mathrm{NExt}} = (\Sigma_{\mathrm{NExt}}, Act, R_{\mathrm{NExt}})$ given by the rules R_{NExt} in Table 1. We show that all operators in Table 1 are non-extensive.

▶ **Proposition 12.** *Assume any TSS* $T = (\Sigma, A, R)$ *with* $T_{NExt} \sqsubseteq T$ *and* $s_i, t_i \in \mathcal{T}(\Sigma)$*. For* $\mathsf{j} \in \{\mathsf{q},\mathsf{m}\}$*, let us define* $r^{\mathsf{j}}_{(s,t,t')} = \min(\mathbf{r}^{\mathsf{j}}(s,t), \mathbf{r}^{\mathsf{j}}(s,t'))$ *and* $r^{\mathsf{j}}_{(q,s,t,t')} = \min(\mathbf{r}^{\mathsf{j}}(s,t), q(\mathbf{r}^{\mathsf{j}}(s,t)) + (1-q)\mathbf{r}^{\mathsf{j}}(s,t'))$*. Then, we have:*

(a) $\mathbf{r}^{\mathsf{j}}(\alpha.\bigoplus_{i=1}^{n}[p_i]s_i, \alpha.\bigoplus_{i=1}^{n}[p_i]t_i) \leq \lambda \cdot \sum_{i=1}^{n} p_i \mathbf{r}^{\mathsf{j}}(s_i,t_i)$ *with* $\mathsf{j} \in \{\mathsf{q},\mathsf{m}\}$*;*
(b) $\mathbf{r}^{\mathsf{q}}(s_1+s_2, t_1+t_2) \leq \max(r^{\mathsf{q}}_{(s_1,t_1,t_2)}, r^{\mathsf{q}}_{(s_2,t_1,t_2)})$*;*
(c) $\mathbf{r}^{\mathsf{m}}(s_1+s_2, t_1+t_2) \leq \max(r^{\mathsf{m}}_{(s_1,t_1,t_2)}, r^{\mathsf{m}}_{(s_2,t_1,t_2)}, r^{\mathsf{m}}_{(t_1,s_1,s_2)}, r^{\mathsf{m}}_{(t_2,s_1,s_2)})$*;*
(d) $\mathbf{r}^{\mathsf{q}}(s_1 +_p s_2, t_1 +_p t_2) \leq \max(r^{\mathsf{q}}_{(p,s_1,t_1,t_2)}, r^{\mathsf{q}}_{((1-p),s_2,t_2,t_1)})$*;*
(e) $\mathbf{r}^{\mathsf{m}}(s_1 +_p s_2, t_1 +_p t_2) \leq \max(r^{\mathsf{m}}_{(p,s_1,t_1,t_2)}, r^{\mathsf{m}}_{((1-p),s_2,t_2,t_1)}), r^{\mathsf{m}}_{(p,t_1,s_1,s_2)}, r^{\mathsf{m}}_{((1-p),t_2,s_2,s_1)})$*;*
(f) $\mathbf{r}^{\mathsf{j}}((\nu\,\alpha)\,s, (\nu\,\alpha)\,t) \leq \mathbf{r}^{\mathsf{j}}(s,t)$ *with* $\mathsf{j} \in \{\mathsf{q},\mathsf{m}\}$*.*

As expected, the asymmetry leads to have upper bounds for $\mathbf{r}^{\mathsf{q}}$ below those for $\mathbf{r}^{\mathsf{m}}$. For instance, by Proposition 12.2 we get $\mathbf{r}^{\mathsf{q}}(a.\mathsf{0}+a.\mathsf{0}, a.\mathsf{0}+b.\mathsf{0}) \leq \max(\min(0,1),\min(0,1)) = 0$, while by Proposition 12.3 $\mathbf{r}^{\mathsf{m}}(a.\mathsf{0}+a.\mathsf{0}, a.\mathsf{0}+b.\mathsf{0}) \leq \max(\min(0,1),\min(0,1),\min(0,0),\min(1,1)) = 1$.

Note that in Proposition 12 we have $T_{\mathrm{NExt}} \sqsubseteq T$ (Definition 3), namely processes s_i and t_i are obtained by using arbitrary operators, not necessarily only operators in Σ_{NExt}. Thus, these bounds hold independently from T. The following result follows from Proposition 12.

▶ **Theorem 13.** *The operators in Table 1 are non-extensive with respect to* $\mathbf{r}^{\mathsf{q}}$ *and* $\mathbf{r}^{\mathsf{m}}$*.*

Non-expansive operators. We proceed to show that all operators in Table 2 are non-expansive. Consider the TSS $T_{\mathrm{NExp}} = (\Sigma_{\mathrm{NExp}}, Act, R_{\mathrm{NExp}})$ with $T_{\mathrm{NExt}} \sqsubseteq T_{\mathrm{NExp}}$ and R_{NExp} containing the rules in Table 2, besides those in Table 1.

Table 2 Non-expansive operators, where the operator $|$ assumes a function $^- : Act \setminus \{\tau, \surd\} \to Act \setminus \{\tau, \surd\}$ with $\overline{\overline{a}} = a$, and the operator $||_B$ is defined for $B \subseteq Act \setminus \{\tau, \surd\}$

$$\frac{x \xrightarrow{\alpha} \mu}{x \mid y \xrightarrow{\alpha} \mu \mid \delta(y)} \qquad \frac{y \xrightarrow{\alpha} \nu}{x \mid y \xrightarrow{\alpha} \delta(x) \mid \nu} \qquad \frac{x \xrightarrow{a} \mu \quad y \xrightarrow{\overline{a}} \nu \quad a \in Act \setminus \{\tau, \surd\}}{x \mid y \xrightarrow{\tau} \mu \mid \nu}$$

$$\frac{x \xrightarrow{\surd} \mu \quad y \xrightarrow{\surd} \nu}{x \mid y \xrightarrow{\surd} \delta(0)} \qquad \frac{x \xrightarrow{\alpha} \mu \quad a \neq \surd}{x; y \xrightarrow{\alpha} \mu; \delta(y)} \qquad \frac{x \xrightarrow{\surd} \mu \quad y \xrightarrow{\alpha} \nu}{x; y \xrightarrow{\alpha} \nu} \qquad \frac{x \xrightarrow{\surd} \mu \quad y \xrightarrow{\surd} \nu}{x \mid\mid_B y \xrightarrow{\surd} \delta(0)}$$

$$\frac{x \xrightarrow{a} \mu \quad y \xrightarrow{a} \nu \quad a \in B}{x \mid\mid_B y \xrightarrow{a} \mu \mid\mid_B \nu} \qquad \frac{x \xrightarrow{\alpha} \mu \quad \alpha \notin (B \cup \{\surd\})}{x \mid\mid_B y \xrightarrow{\alpha} \mu \mid\mid_B \delta(y)} \qquad \frac{y \xrightarrow{\alpha} \nu \quad \alpha \notin (B \cup \{\surd\})}{x \mid\mid_B y \xrightarrow{\alpha} \delta(x) \mid\mid_B \nu}$$

▶ **Proposition 14.** *Assume any TSS $T = (\Sigma, A, R)$ with $T_{NExp} \sqsubseteq T$ and $s_i, t_i \in \mathcal{T}(\Sigma)$. For $\mathsf{j} \in \{\mathsf{q}, \mathsf{m}\}$ we have:*

(a) $\mathbf{r}^{\mathsf{j}}(s_1; s_2, t_1; t_2) \leq \begin{cases} 1 & \text{if } \mathbf{r}^{\mathsf{j}}(s_1, t_1)=1 \\ \max\{\mathbf{r}^{\mathsf{j}}(s_1, t_1)+\lambda(1-\frac{\mathbf{r}^{\mathsf{j}}(s_1,t_1)}{\lambda})\mathbf{r}^{\mathsf{j}}(s_2, t_2)\,,\ \mathbf{r}^{\mathsf{j}}(s_2, t_2)\} & \text{if } \mathbf{r}^{\mathsf{j}}(s_1, t_1)\in[0, 1) \end{cases}$

(b) $\mathbf{r}^{\mathsf{j}}(s_1 \mid s_2, t_1 \mid t_2) \leq r^{\mathsf{j}}_{\mathsf{synch}}$

(c) $\mathbf{r}^{\mathsf{j}}(s_1 \parallel_B s_2, t_1 \parallel_B t_2) \leq \begin{cases} r^{\mathsf{j}}_{\mathsf{synch}} & \text{if } B \neq \emptyset \\ r^{\mathsf{j}}_{\mathsf{asynch}} & \text{otherwise} \end{cases}$

with

$$r^{\mathsf{j}}_{\mathsf{synch}} = \begin{cases} 1 & \text{if } \mathbf{r}^{\mathsf{j}}(s_1, t_1) = 1 \vee \mathbf{r}^{\mathsf{j}}(s_2, t_2) = 1 \\ \mathbf{r}^{\mathsf{j}}(s_1, t_1) + \mathbf{r}^{\mathsf{j}}(s_2, t_2) - \frac{\mathbf{r}^{\mathsf{j}}(s_1,t_1)\mathbf{r}^{\mathsf{j}}(s_2,t_2)}{\lambda} & \text{otherwise} \end{cases}$$

$$r^{\mathsf{j}}_{\mathsf{asynch}} = \begin{cases} 1 & \text{if } \mathbf{r}^{\mathsf{j}}(s_1, t_1)=1 \vee \mathbf{r}^{\mathsf{j}}(s_2, t_2)=1 \\ \max\{\mathbf{r}^{\mathsf{j}}(s_1, t_1) + \lambda^2\mathbf{r}^{\mathsf{j}}(s_2, t_2) - \lambda\mathbf{r}^{\mathsf{j}}(s_1, t_1)\mathbf{r}^{\mathsf{j}}(s_2, t_2), & \\ \quad \mathbf{r}^{\mathsf{j}}(s_2, t_2) + \lambda^2\mathbf{r}^{\mathsf{j}}(s_1, t_1) - \lambda\mathbf{r}^{\mathsf{j}}(s_1, t_1)\mathbf{r}^{\mathsf{j}}(s_2, t_2)\} & \text{otherwise.} \end{cases}$$

Let us explain first Proposition 14.1. If $\mathbf{r}^{\mathsf{j}}(s_1, t_1) = 1$ then the maximal distance between s_1 and t_1 extends to $s_1; s_2$ and $t_1; t_2$. If $\mathbf{r}^{\mathsf{j}}(s_1, t_1) < 1$ then $\mathbf{r}^{\mathsf{j}}(s_1; s_2\,,\ t_1; t_2)$ is the maximum between the values given by the two different scenarios:

(i) the first one is that s_1 and t_1 evolve followed by s_2 and t_2, thus implying that we observe the distance $\mathbf{r}^{\mathsf{j}}(s_1, t_1)$ between s_1 and t_1 plus the distance $\mathbf{r}^{\mathsf{j}}(s_2, t_2)$ between s_2 and t_2, weighted by the likelihood that s_1 and t_1 exhibit the same behaviour, which is at most $(1 - \mathbf{r}^{\mathsf{j}}(s_1, t_1)/\lambda)$, and discounted by λ, since s_2 and t_2 are delayed by at least one step;

(ii) the second scenario is that s_1 and t_1 terminate immediately, so that we can observe only the distance $\mathbf{r}^{\mathsf{j}}(s_2, t_2)$ between s_2 and t_2, with no discount.

Consider now Proposition 14.2. If $\mathbf{r}^{\mathsf{j}}(s_1, t_1) = 1$ or $\mathbf{r}^{\mathsf{j}}(s_2, t_2) = 1$ then the upper bound is immediate. Otherwise, we obtain $\mathbf{r}^{\mathsf{j}}(s_1 \mid s_2, t_1 \mid t_2)$ by summing the distances $\mathbf{r}^{\mathsf{j}}(s_1, t_1)$ and $\mathbf{r}^{\mathsf{j}}(s_2, t_2)$ and, then, by subtracting $\frac{\mathbf{r}^{\mathsf{j}}(s_1,t_1)\mathbf{r}^{\mathsf{j}}(s_2,t_2)}{\lambda}$, which allows us to weight one of the two distances, say $\mathbf{r}^{\mathsf{j}}(s_2, t_2)$ by the likelihood that the other two processes exhibit the same behaviour, namely $(1 - \frac{\mathbf{r}^{\mathsf{j}}(s_1,t_1)}{\lambda})$. Finally, consider Proposition 14.3. If processes can synchronise, then the upper bound is the same as Proposition 14.2. Otherwise, either s_1 and t_1 evolve and s_2 and t_2 are delayed, or, symmetrically, s_2 and t_2 evolve and s_1 and t_1 are delayed. The distance between the delayed processes is therefore discounted and we get a bound slightly below that of Proposition 14.2.

Table 3 Lipschitz continuous operators

$$\frac{x \xrightarrow{a} \mu \quad a \neq \surd}{x^{n+1} \xrightarrow{a} \mu;\delta(x^n)} \qquad \frac{x \xrightarrow{\surd} \mu}{x^{n+1} \xrightarrow{\surd} \mu} \qquad \frac{}{x^0 \xrightarrow{\surd} \delta(0)} \qquad \frac{x \xrightarrow{\surd} \mu \quad x \xrightarrow{a} \nu \quad a \neq \surd \quad n > m}{x^n \xrightarrow{a} \nu;\delta(x^m)}$$

$$\frac{x \xrightarrow{a} \mu \quad a \neq \surd}{!^{n+1}x \xrightarrow{a} \mu \,||_\emptyset\, \delta(!^n x)} \qquad \frac{x \xrightarrow{\surd} \mu}{!^{n+1}x \xrightarrow{\surd} \mu} \qquad \frac{}{!^0 x \xrightarrow{\surd} \delta(0)} \qquad \frac{x \xrightarrow{a} \mu \quad a \neq \surd}{x^\omega \xrightarrow{a} \mu;\delta(x^\omega)}$$

$$\frac{x \xrightarrow{a} \mu \quad a \neq \surd}{x^* y \xrightarrow{a} \mu;\delta(x^* y)} \qquad \frac{y \xrightarrow{a} \nu}{x^* y \xrightarrow{a} \nu} \qquad \frac{x \xrightarrow{a} \mu \quad a \neq \surd}{!x \xrightarrow{a} \mu \,|||\, \delta(!x)} \qquad \frac{x \xrightarrow{a} \mu \quad a \neq \surd}{!_p x \xrightarrow{a} \mu \oplus_p (\mu \,|||\, \delta(!_p x))}$$

Notice that also the processes s_i and t_i in Proposition 14 are obtained by using arbitrary operators, not necessarily in Σ_{NExp}. The following result follows from Proposition 14.

▶ **Theorem 15.** *The operators in Table 2 are non-expansive with respect to* $\mathbf{r}^{\mathsf{q}}$ *and* $\mathbf{r}^{\mathsf{m}}$.

Clearly, the results in Proposition 14 can be generalised to more complex terms. For instance, we give two generalizations of Proposition 14.2 that will be used in the case study presented in the next section.

▶ **Proposition 16.** *Assume processes* $s_1, t_1, \ldots, s_n, t_n$, *with* $n \geq 2$. *We have:*

$$\mathbf{r}^{\mathsf{m}}(s_1|\ldots|s_n, t_1|\ldots|t_n) \;\leq \sum_{\emptyset \subset I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} \prod_{i \in I} \mathbf{r}^{\mathsf{m}}(s_i, t_i) \;\; .$$

▶ **Proposition 17.** *Assume processes* $s_1, t_1, \ldots, s_n, t_n$, *with* $n \geq 2$. *If* $\mathbf{r}^{\mathsf{m}}(s_i, t_i) = r$ *for all* $i = 1, \ldots n$, *then we have:*

$$\mathbf{r}^{\mathsf{m}}(s_1|\ldots|s_n, t_1|\ldots|t_n) \;\leq\; -\sum_{i=1}^{n} \binom{n}{i} (-r)^i \;\; .$$

Lipschitz continuous operators. Now we show that all operators in Table 3 are Lipschitz continuous. Consider the TSS $T_{\mathrm{LC}} = (\Sigma_{\mathrm{LC}}, Act, R_{\mathrm{LC}})$ with $T_{\mathrm{NExp}} \sqsubseteq T_{\mathrm{LC}}$ and R_{LC} containing the rules in Table 3, besides those in Table 1–2.

▶ **Proposition 18.** *Assume any TSS* $T = (\Sigma, A, R)$ *with* $T_{LC} \sqsubseteq T$ *and* $s, s_i, t, t_i \in \mathcal{T}(\Sigma)$. *For* $\mathsf{j} \in \{\mathsf{q}, \mathsf{m}\}$ *we have:*

(a) $\mathbf{r}^{\mathsf{j}}(s^n, t^n) \leq \begin{cases} \mathbf{r}^{\mathsf{j}}(s,t)\frac{1-(\lambda-\mathbf{r}^{\mathsf{j}}(s,t))^n}{1-(\lambda-\mathbf{r}^{\mathsf{j}}(s,t))} & \text{if } \mathbf{r}^{\mathsf{j}}(s,t) \in (0,1) \\ \mathbf{r}^{\mathsf{j}}(s,t) & \text{if } \mathbf{r}^{\mathsf{j}}(s,t) \in \{0,1\} \end{cases}$

(b) $\mathbf{r}^{\mathsf{j}}(!^n s, !^n t) \leq \begin{cases} \mathbf{r}^{\mathsf{j}}(s,t)\frac{1-(\lambda^2-\lambda\mathbf{r}^{\mathsf{j}}(s,t))^n}{1-(\lambda^2-\lambda\mathbf{r}^{\mathsf{j}}(s,t))} & \text{if } \mathbf{r}^{\mathsf{j}}(s,t) \in (0,1) \\ \mathbf{r}^{\mathsf{j}}(s,t) & \text{if } \mathbf{r}^{\mathsf{j}}(s,t) \in \{0,1\} \end{cases}$

(c) $\mathbf{r}^{\mathsf{j}}(s^\omega, t^\omega) \leq \begin{cases} \mathbf{r}^{\mathsf{j}}(s,t)\frac{1}{1-(\lambda-\mathbf{r}^{\mathsf{j}}(s,t))} & \text{if } \mathbf{r}^{\mathsf{j}}(s,t) \in (0,1) \\ \mathbf{r}^{\mathsf{j}}(s,t) & \text{if } \mathbf{r}^{\mathsf{j}}(s,t) \in \{0,1\} \end{cases}$

(d) $\mathbf{r}^{\mathsf{j}}(!s, !t) \leq \begin{cases} \mathbf{r}^{\mathsf{j}}(s,t)\frac{1}{1-(\lambda^2-\lambda\mathbf{r}^{\mathsf{j}}(s,t))} & \text{if } \mathbf{r}^{\mathsf{j}}(s,t) \in (0,1) \\ \mathbf{r}^{\mathsf{j}}(s,t) & \text{if } \mathbf{r}^{\mathsf{j}}(s,t) \in \{0,1\} \end{cases}$

(e) $\mathbf{r}^{\mathsf{j}}(s_1{}^* s_2, t_1{}^* t_2) \leq \max(\mathbf{r}^{\mathsf{j}}(s_1{}^\omega, t_1{}^\omega), \mathbf{r}^{\mathsf{j}}(s_2, t_2))$

(f) $\mathbf{r}^{\mathsf{j}}(!_p s, !_p t) \leq \begin{cases} \mathbf{r}^{\mathsf{j}}(s,t)\frac{1}{1-(1-p)(\lambda^2-\lambda\mathbf{r}^{\mathsf{j}}(s,t))} & \text{if } \mathbf{r}^{\mathsf{j}}(s,t) \in (0,1) \\ \mathbf{r}^{\mathsf{j}}(s,t) & \text{if } \mathbf{r}^{\mathsf{j}}(s,t) \in \{0,1\}. \end{cases}$

The bound in Proposition 18.1 is obtained by applying $n-1$ times the bound in Proposition 14 for operator $\_\,;\_$, the rationale being that the pLTS associated to s^n is isomorphic to that of process $s;\ldots;s$ with n occurrences of s. Similarly, the bound in Proposition 18.2 is obtained by applying $n-1$ times the bound in Proposition 14 for operator $\_\,||_\emptyset\,\_$, the rationale being that $!^n s$ denotes a pLTS isomorphic to that of process $s\,||_\emptyset \ldots ||_\emptyset\, s$ with n occurrences of s. The bounds in Proposition 18.3 and Proposition 18.4 are obtained by taking the limits for the bounds in Proposition 18.1 and Proposition 18.2, respectively. Proposition 18.5 is obtained by observing that the (bi)simulation tolerance between processes ${s_1}^* s_2$ and ${t_1}^* t_2$ is less than or equal to the maximum of the tolerance bound $\mathbf{r}^{\mathsf{j}}({s_1}^\omega, {t_1}^\omega)$ (infinite iteration of s_1 and t_1 such that s_2 and t_2 never evolve), and the tolerance bound $\mathbf{r}^{\mathsf{j}}(s_2,t_2)$ (s_2 and t_2 evolve immediately). The case where s_1 and t_1 iterate n-times and then s_2 and t_2 evolve leads always to a tolerance bound $\mathbf{r}^{\mathsf{j}}({s_1}^n,{t_1}^n) + (\lambda - \mathbf{r}^{\mathsf{j}}(s_1,t_1))^n \mathbf{r}^{\mathsf{j}}(s_2,t_2) \leq \max(\mathbf{r}^{\mathsf{j}}({s_1}^\omega,{t_1}^\omega), \mathbf{r}^{\mathsf{j}}(s_2,t_2))$. Finally, Proposition 18.6 can be understood by observing that $!_p s$ behaves as $!^{n+1}s$ with probability $p(1-p)^n$. Hence, by Proposition 18.2 we get $\mathbf{r}^{\mathsf{j}}(!_p s, !_p t) \leq \sum_{n=0}^{\infty} p(1-p)^n \mathbf{r}^{\mathsf{j}}(!^{n+1}s, !^{n+1}t) \leq \sum_{n=0}^{\infty} p(1-p)^n r^{\mathsf{j}}_{!^{n+1}} = \mathbf{r}^{\mathsf{j}}(s,t)/(1-(1-p)(\lambda^2 - \lambda \mathbf{r}^{\mathsf{j}}(s,t)))$.

Notice also that the processes s, t, s_i, t_i in Proposition 18 are obtained by using arbitrary operators, not necessarily in Σ_{LC}. The following result follows from Proposition 18.

▶ **Theorem 19.** *The operators*

(i) *finite iteration* $\_^n$
(ii) *finite replication* $!^n\_$
(iii) *probabilistic replication* $!_p\_$

are Lipschitz continuous with respect to $\mathbf{r}^{\mathsf{q}}$ *and* $\mathbf{r}^{\mathsf{m}}$ *for any* $\lambda \in (0,1]$. *The operators*

(i) *infinite iteration* $\_^\omega$
(ii) *nondeterministic Kleene-star iteration* $\_^*\_$
(iii) *infinite replication* $!\_$

are Lipschitz continuous with respect to $\mathbf{r}^{\mathsf{q}}$ *and* $\mathbf{r}^{\mathsf{m}}$ *for any* $\lambda \in (0,1)$.

Notice that discounting the distance observed at step n by λ^n is necessary to have compositionality of the operators $\_^\omega$, $\_^*\_$, and $!\_$.

4 A case study: Group Key Update

A *group key* is a secret key shared by a group of principals to secure their *multicast communications*. Group key update protocols were originally adopted to secure LANs [32]. Nowadays they are widely used in different contexts, such as: audio and video conferencing in Computer Supported Co-operative Work (CSCW), Virtual Private Networks (VPN), distributed databases, instant messaging applications, etc.

A crucial problem when dealing with key-secured communications is *rekeying*, i.e. the process of distributing new keys to the principals. Rekeying is necessary when a member joins the group, to prevent it to access the information exchanged in the past (*backward confidentiality*), and when a member leaves the group, to prevent it to access future data (*forward confidentiality*). Rekeying is managed either by a third trusted party or by a member acting as *group owner*. In our example, we abstract from these two solutions by assuming a unique *key manager entity* which takes care of rekeying.

We assume a set $\mathcal{N}$ of member IDs. For each members $i \in \mathcal{N}$, the probabilities of leaving and joining the group are $l(i)$ and $j(i)$, respectively. Furthermore, each member can leave/join the group at most n times. Notice that high values of n, $l(i)$ or $j(i)$ cause frequent rekeying, with obvious consequences on:

Table 4 An abstract group key update protocol.

$$
\begin{array}{lcl}
\mathit{Group}(l,j) & = & (\nu\,(\mathit{Act} \setminus \mathsf{newK}))\,\big(\mathit{Manager} \mid \prod_{i\in\mathcal{N}} \mathit{Member}(i,l(i),j(i))\big) \\
\mathit{Manager} & = & \mathit{Connected} \mid \big(\sum_{I\subseteq\mathcal{N}} \mathsf{act}_I; \mathit{Mng}(I)\big)^{\omega} \\
\mathit{Connected} & = & \overline{\mathsf{act}}_{\mathcal{N}}; \Big(\sum_{I\subseteq\mathcal{N}} \mathsf{sync}_I; \overline{\mathsf{act}}_I; \varepsilon\Big)^{\omega} \\
\mathit{Mng}(I) & = & \big(\sum_{i\in I} \mathsf{leave}_i; \overline{\mathsf{newK}}; \mathit{SendNewKey}(I\setminus i); \overline{\mathsf{sync}}_{I\setminus i}; \varepsilon\big) + \\
 & & \big(\sum_{i\in(\mathcal{N}\setminus I)} \mathsf{join}_i; \overline{\mathsf{newK}}; \mathit{SendNewKey}(I\cup i); \overline{\mathsf{sync}}_{I\cup i}; \varepsilon\big) \\
\mathit{SendNewKey}(\{i_1,\ldots,i_k\}) & = & \overline{(\mathsf{i}_1,\mathsf{Key})}; \ldots; \overline{(\mathsf{i}_k,\mathsf{Key})}; \varepsilon \\
\mathit{Member}(i,p,q) & = & \mathit{State}(i) \mid \big(\mathit{MembIn}(i,p) + \mathit{MembOut}(i,q)\big)^{n} \\
\mathit{State}(i) & = & \overline{\mathsf{in}}_i; \big(\overline{\mathsf{in}}_i{}^{*}; \mathsf{change}_i; \overline{\mathsf{out}}_i{}^{*}; \mathsf{change}_i; \varepsilon\big)^{*} \\
\mathit{MembIn}(i,p) & = & \mathsf{in}_i; \big((\mathsf{i},\mathsf{Key}) + \tau; \big((\overline{\mathsf{leave}}_i; \overline{\mathsf{change}}_i; \varepsilon) \oplus_p \varepsilon\big)\big) \\
\mathit{MembOut}(i,q) & = & \mathsf{out}_i; \tau; \big((\overline{\mathsf{join}}_i; \overline{\mathsf{change}}_i; (\mathsf{i},\mathsf{Key}); \varepsilon) \oplus_q \varepsilon\big)
\end{array}
$$

(a) the number of attacks aiming at compromising the group key,
(b) degradation of communication service,
(c) battery consumption.

Thus, in the following, a group key protocol in which members never leave/join the group (i.e. $l(i) = j(i) = 0$, for any $i \in \mathcal{N}$) will be called *ideal*.

Our goal is to show that the theory developed in the previous section represents an effective instrument to estimate the distance between the ideal protocol and possible variations of the protocol obtained by playing with the parameters n, $l(i)$ and $j(i)$, for $i \in \mathcal{N}$.

Table 4 reports an abstract representation of the rekeying process in terms of our general process algebra. Since cryptographic details are not relevant for our purposes, we protect communications via the restriction operator $(\nu\,\alpha)\,\_$. We observe only the signal newK denoting the rekeying event. For simplicity, in the protocol, we will write $I \setminus i$ instead of $I \setminus \{i\}$, and $I \cup i$ instead of $I \cup \{i\}$.

The process $\mathit{Group}(l,j)$ represents a group, in its initial configuration, containing all members in $\mathcal{N}$, each of which can leave/enter the group at most n times. This process consists of the parallel composition of the process *Manager* together with a process for each member in $\mathcal{N}$. The process *Manager* has two parallel components: (a) the process *Connected*, which determines those members which are currently part of the group, and (b) a process that upon reception of a signal act_I, with $I \subseteq \mathcal{N}$, it behaves as $\mathit{Mng}(I)$, i.e. the process managing the group with members in the set I. Initially, all members join the group. Thus, *Connected* starts by activating the process $\mathit{Mng}(\mathcal{N})$. Then, whenever *Connected* receives a signal sync_I (from some $\mathit{Mng}(J)$) it activates $\mathit{Mng}(I)$ by sending the signal act_I. Notice that, in this case, there is a member i such that either $I = J \setminus \{i\}$, because i has left the group, or $I = J \cup \{i\}$, because i has joined the group.

The process $\mathit{Mng}(I)$ behaves as follows. Whenever it senses a signal leave_i (resp. join_i) denoting that a connected member $i \in I$ (resp. an unconnected member $i \in \mathcal{N} \setminus I$) is leaving (resp. joining) the group, it performs the following actions:

(i) it signals the generation of a new key,
(ii) it broadcasts the new key to all members in $J = I \setminus i$ (resp. $J = I \cup i$), namely the members of the new group,
(iii) it communicates to *Connected* the new set of current members in the group via a signal sync_J.

The process $Member(i,p,q)$ consists of the parallel composition of the process $State(i)$, which stores the state of member i, together with either the process $MembIn(i,p)$, which describes the behaviour of i when it is in the group, or the process $MembOut(i,q)$, which describes the behaviour of i when it is out of the group. The signal in_i (resp. out_i) is used by $State(i)$ to activate $MembIn(i,p)$ (resp. $MembOut(i,q)$). Via process $MembIn(i,p)$, the member i may either receive the new key from the manager or leave the group with a probability p. Similarly, via process $MembOut(i,q)$, the member i may decide to join the group with probability q. If i succeeds in joining the group then a new group key is sent to i and all current members of the group. This completes the explanation of the protocol.

Now, let $\mathbf{p}$ denote the constant function $\mathbf{p}\colon \mathcal{N} \to [0,1]$, with $\mathbf{p}(i) = p$ for all $i \in \mathcal{N}$. Similarly, we define $\mathbf{q}$. Thus, $Group(\mathbf{p},\mathbf{q})$ denotes the instance of the protocol where all members have the same leave/join probability, whereas $Group(\mathbf{0},\mathbf{0})$ denotes the *ideal* group, where rekeying never occurs as the no principal leaves or join the group.

We start our analysis by providing an upper bound of the distance between the behaviours of an arbitrary member $i \in \mathcal{N}$, when varying leave/join probabilities. For that we need a technical lemma to estimate the distance between two occurrences of probabilistic prefix dealing with the same processes, but different probabilities.

▶ **Lemma 20.** *For all $s,t \in \mathcal{T}(\Sigma)$ we have $\mathbf{r}^{\mathsf{m}}(a.(s \oplus_p t), a.(s \oplus_q t)) \leq |p-q|$.*

▶ **Proposition 21.** *Let $p,q,p',q' \in [0,1]$, then*

$$\mathbf{r}^{\mathsf{m}}(Member(i,p,q), Member(i,p',q')) \;\leq\; \max(|p-p'|,|q-q'|)\frac{1-(1-r)^n}{1+r}.$$

Proof. By Lemma 20 and compositionality results in Proposition 12.1 and Prop. 12.2 we get $\mathbf{r}^{\mathsf{m}}(MembIn(i,p), MembIn(i,p')) \leq |p-p'|$ and $\mathbf{r}^{\mathsf{m}}(MembOut(i,q), MembOut(i,q')) \leq |q-q'|$. Then, the thesis follows by applying the compositionality results in Proposition 12.3, Proposition 14.2, Proposition 18.1 ◀

We can generalise this result by taking into account all members in $\mathcal{N}$.

▶ **Proposition 22.** *Given arbitrary functions $l,j,l',j'\colon \mathcal{N} \to [0,1]$, we have:*

$$\mathbf{r}^{\mathsf{m}}(Group(l,j), Group(l',j')) \leq \sum_{I\subseteq\mathcal{N}, I\neq\emptyset} (-1)^{|I|+1} \prod_{i\in I} r(i)\frac{1-(1-r(i))^n}{1+r(i)}$$

with $r(i) = \max(|l(i)-l'(i)|, |j(i)-j'(i)|)$.

Proof. By Proposition 21, Proposition 16 and the compositionality results of Proposition 14.2 and Proposition 12.6. ◀

Proposition 22 estimates the distance between an arbitrary group and the ideal one.

▶ **Corollary 23.** *For any $l,j,r\colon \mathcal{N} \to [0,1]$ such that $r(i) = \max(l(i),j(i))$, we have:*

$$\mathbf{r}^{\mathsf{m}}(Group(\mathbf{0},\mathbf{0}), Group(l,j)) \;\leq\; \sum_{I\subseteq\mathcal{N}, I\neq\emptyset} (-1)^{|I|+1} \prod_{i\in I} r(i)\frac{1-(1-r(i))^n}{1+r(i)} \quad .$$

We can also compare two instances of the protocol when assuming homogeneous probabilities (i.e. all members leaves/join the group with the same probabilities.)

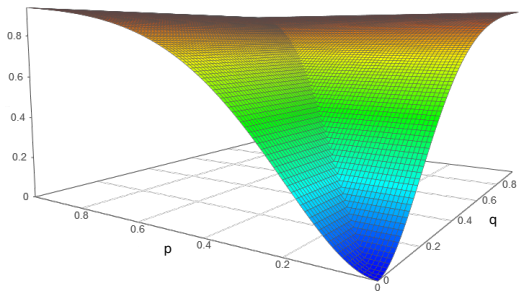


Figure 1 Upper bound to $\mathbf{r}^{\mathsf{m}}(\mathit{Group}(\mathbf{0},\mathbf{0}), \mathit{Group}(\mathbf{p},\mathbf{q}))$

▶ **Proposition 24.** *For arbitrary probabilities $p, p', q, q' \in [0,1]$ we have:*

$$\mathbf{r}^{\mathsf{m}}(\mathit{Group}(\mathbf{p},\mathbf{q}), \mathit{Group}(\mathbf{p}',\mathbf{q}')) \leq -\sum_{i=1}^{|\mathcal{N}|} \binom{|\mathcal{N}|}{i} \left(-r\frac{1-(1-r)^n}{1+r}\right)^i$$

with $r = \max(|p-p'|, |q-q'|)$.

Proof. By Proposition 21, Proposition 17 and the compositionality results of Proposition 12.6 and Proposition 14.2. ◀

This allows us to compare a group with homogeneous probabilities with the ideal group.

▶ **Corollary 25.** *For arbitrary probabilities $p, q \in [0,1]$ we have:*

$$\mathbf{r}^{\mathsf{m}}(\mathit{Group}(\mathbf{0},\mathbf{0}), \mathit{Group}(\mathbf{p},\mathbf{q})) \leq -\sum_{i=1}^{|\mathcal{N}|} \binom{|\mathcal{N}|}{i} \left(-(\max(p,q)\frac{1-(1-(\max(p,q))^n}{1+(\max(p,q)}\right)^i.$$

For instance, assume an instance of the protocol with $\mathcal{N} = \{1,\ldots,4\}$ and $n = 3$. Then we have $\mathbf{r}^{\mathsf{m}}(\mathit{Group}(\mathbf{0},\mathbf{0}), \mathit{Group}(\mathbf{p},\mathbf{q})) \leq 4r - 6r^2 + 4r^3 - r^4$, with $r = \max(p,q)\frac{1-(1-\max(p,q))^3}{1+\max(p,q)}$. The upper bound is reported in Fig. 1. Notice that the surface is symmetric, meaning that the upper bounds for $p \geq q$ and $q \geq p$ coincide (because they depend on $\max(p,q)$).

Figures 2 and 3 describe the evolution of the upper bound of $\mathbf{r}^{\mathsf{m}}(\mathit{Group}(\mathbf{0},\mathbf{0}), \mathit{Group}(\mathbf{p},\mathbf{p}))$, i.e. when leave and join probability are the same ($p = q$). In Figure 2 we fix 4 members and we vary n in the set $\{3,5,7,9,11\}$. We can observe that $\mathbf{r}^{\mathsf{m}}(\mathit{Group}(\mathbf{0},\mathbf{0}), \mathit{Group}(\mathbf{p},\mathbf{p}))$ grows with n, in particular for group with low values of p and q. In Figure 3, we fix n equals to 3 and vary the number of members in the set $\{4,5,6,7,8\}$. We can observe that $\mathbf{r}^{\mathsf{m}}(\mathit{Group}(\mathbf{0},\mathbf{0}), \mathit{Group}(\mathbf{p},\mathbf{p}))$ grows with the size of the group, in particular in group with high values of p (and q).

Summarising, we can assert that the protocol under analysis has good efficiency in groups with low dynamicity, regardless of the size of the group.

5 Conclusions, related and future work

We showed that uniform continuity is an effective property to achieve compositional reasoning with respect to rooted (quasi)metric semantics. We considered all standard operators of

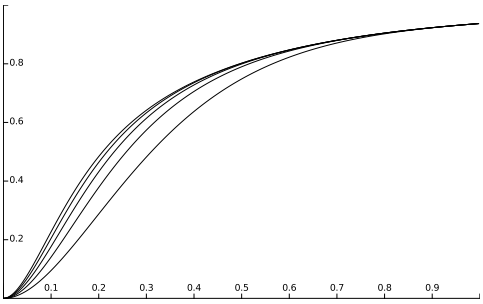


Figure 2 4 members and $n \in \{3, 5, 7, 9, 11\}$.

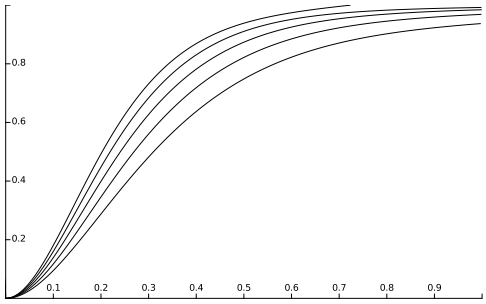


Figure 3 $n = 3$ and $4 \leq$ members ≤ 8.

probabilistic process algebra and provided suitable upper bounds on the distance between processes composed by these operators. This allows us to infer their uniform continuity with respect to both rooted bisimilarity metric and rooted similarity quasimetric. Interestingly, the rootedness condition, introduced to deal with nondeterministic and probabilistic choice, is crucial when dealing with similarity quasimetric. We exemplified how these semantic theories can be used to pursue compositional reasoning over a non-trivial protocol.

The current paper is the ideal continuation of [20]. In that paper, the authors show that uniform continuity captures the essential intuition of compositional reasoning when dealing with probabilistic processes. The proposal of [20] generalises and extends earlier proposals in [17, 2] to capture recursive operators. The focus of all these papers is on strong bisimulation metrics. We remark that, following [35, 17, 14], we have considered (bi)simulation-inspired (quasi)metric for the pLTS model. However, the literature offers also different approaches to estimate the distance between processes. In [19] a spectrum of distances between processes is obtained by applying the theory of quantitative Ehrenfeucht-Fraïssé games to transition systems. This theory allows to generate different notions of distance by means of different generalisations of a suitable distance over traces. Paper [11] studies distances between processes in the semantic model of Metric Transition Systems. In [3, 8] trace metrics for the model of Markov Chains are defined as total variation distances on the cones generated by the execution traces. In [29] the distance between systems is defined by means of a probabilistic approximated bisimulation. This paper provides a technique to compute upper bounds based on compositional algebraic laws.

As *future work*, we will extend the analysis of concrete process algebra operators to general SOS rule and specification formats. A SOS rule and specification format ensuring uniform continuity of operators with respect to strong bisimilarity metric has been proposed in [21, 22], the idea being that process arguments of operators are copied only finitely many times along their evolution. In order to achieve the same result in the weak case, it is necessary to strengthen the format of [21] by preventing that process replication can arise by τ-transitions. After that, we intend to extend our approach to the weak versions of other notions of distance, such as convex bisimulation metric [13], trace metric [19], and total-variation distance based metrics [31]. Finally, another possible research direction is develop a timed-variant of our technique to deal with timed aspects of systems as in [27, 28].

Acknowledgements. We thank the anonymous reviewers for valuable comments.

References

1 Luca Aceto, Bard Bloom, and Fritz W. Vaandrager. Turning SOS Rules into Equations. *Information & Computation*, 111(1):1–52, 1994. doi:10.1006/inco.1994.1040.

2 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. Computing Behavioral Distances, Compositionally. In *38th International Symposium on Mathematical Foundations of Computer Science*, volume 8087 of *LNCS*, pages 74–85. Springer, 2013. doi:10.1007/978-3-642-40313-2_9.

3 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. Converging from Branching to Linear Metrics on Markov Chains. In *12th International Colloquium on Theoretical Aspects of Computing*, volume 9399 of *LNCS*, pages 349–367. Springer, 2015. doi:10.1007/978-3-319-25150-9_21.

4 Christel Baier, Holger Hermanns, and Joost-Pieter Katoen. Probabilistic Weak Simulation is Decidable in Polynomial Time. *Information Processing Letters*, 89(3):123–130, 2004. doi:10.1016/j.ipl.2003.10.001.

5 Christel Baier, Joost-Pieter Katoen, Holger Hermanns, and Boudewijn R. Haverkort. Simulation for Continuous-Time Markov Chains. In *13th International Conf. on Concurrency Theory*, volume 2421 of *LNCS*, pages 338–354, 2002. doi:10.1007/3-540-45694-5_23.

6 Falk Bartels. *On Generalised Coinduction and Probabilistic Specification Formats.* PhD thesis, VU University Amsterdam, 2004.

7 Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation Can't Be Traced. *Journal of the ACM*, 42:232–268, 1995. doi:10.1145/200836.200876.

8 Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský, and Tatjana Petrov. Linear Distances between Markov Chains. In *27th International Conference on Concurrency Theory*, LIPIcs, pages 20:1–20:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.20.

9 Pedro R. D'Argenio, Daniel Gebler, and Matias D. Lee. Axiomatizing Bisimulation Equivalences and Metrics from Probabilistic SOS Rules. In *17th International Conference on Foundations of Software Science and Computation Structures*, volume 8412 of *LNCS*, pages 289–303. Springer, 2014. doi:10.1007/978-3-642-54830-7_19.

10 Pedro R. D'Argenio, Daniel Gebler, and Matias D. Lee. A General SOS Theory for the Specification of Probabilistic Transition Systems. *Information & Computation*, 249:76–109, 2016. doi:10.1016/j.ic.2016.03.009.

11 Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and Branching System Metrics. *IEEE Trans. Software Eng.*, 35(2):258–273, 2009. doi:10.1109/TSE.2008.106.

12 Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Discounting the Future in Systems Theory. In *30th Int. Colloquium on Automata, Languages and Programming*, volume 2719 of *LNCS*, pages 1022–1037. Springer, 2003. doi:10.1007/3-540-45061-0_79.

13 Luca de Alfaro, Rupak Majumdar, Vishwanath Raman, and Mariëlle Stoelinga. Game Relations and Metrics. In *22nd IEEE Symposium on Logic in Computer Science*, pages 99–108. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.22.

14 Yuxin Deng, Tom Chothia, Catuscia Palamidessi, and Jun Pang. Metrics for Action-labelled Quantitative Transition Systems. *ENTCS*, 153(2):79–96, 2006. doi:10.1016/j.entcs.2005.10.033.

15 Yuxin Deng and Wenjie Du. The Kantorovich Metric in Computer Science: A Brief Survey. *ENTCS*, 253(3):73–82, 2009. doi:10.1016/j.entcs.2009.10.006.

16 Yuxin Deng, Rob J. van Glabbeek, Matthew Hennessy, and Carroll Morgan. Characterising Testing Preorders for Finite Probabilistic Processes. *Logical Methods in Computer Science*, 4(4), 2008. doi:10.2168/LMCS-4(4:4)2008.

17 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for Labelled Markov Processes. *Theoretical Computer Science*, 318(3):323–354, 2004. doi:10.1016/j.tcs.2003.09.013.

18 Josée Desharnais, Radha Jagadeesan, Vineet Gupta, and Prakash Panangaden. The Metric Analogue of Weak Bisimulation for Probabilistic Processes. In *17th IEEE Symposium on Logic in Computer Science*, pages 413–422. IEEE Computer Society, 2002. doi:10.1109/LICS.2002.1029849.

19 Uli Fahrenberg and Axel Legay. The Quantitative Linear-time-branching-time Spectrum. *Theoretical Computer Science*, 538:54–69, 2014. doi:10.1016/j.tcs.2013.07.030.

20 Daniel Gebler, Kim G. Larsen, and Simone Tini. Compositional Bisimulation Metric Reasoning with Probabilistic Process Calculi. *Logical Methods in Computer Science*, 12(4), 2016. doi:10.2168/LMCS-12(4:12)2016.

21 Daniel Gebler and Simone Tini. Fixed-point Characterization of Compositionality Properties of Probabilistic Processes Combinators. In *Combined 21th International Workshop on Expressiveness in Concurrency and 11th Workshop on Structural Operational Semantics*, volume 160 of *EPTCS*, pages 63–78. OPA, 2014. doi:10.4204/EPTCS.160.7.

22 Daniel Gebler and Simone Tini. SOS Specifications of Probabilistic Systems by Uniformly Continuous Operators. In *26th Conference on Concurrency Theory*, volume 42 of *LIPIcs*, pages 155–168. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

23 Hans Hansson and Bengt Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994. doi:10.1007/BF01211866.

24 Bengt Jonsson, Kim G. Larsen, and Wang Yi. Probabilistic Extensions of Process Algebras. In *Handbook of Process Algebra*, pages 685–710. Elsevier, 2001.

25 Leonid V. Kantorovich. On the transfer of masses. *Doklady Akademii Nauk*, 37(2):227–229, 1942. Original article in Russian, translation in Management Science, $5 : 1 - 4(1959)$.

26 Robert M. Keller. Formal Verification of Parallel Programs. *Communications of the ACM*, 19(7):371–384, 1976. doi:10.1145/360248.360251.

27 Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. Time and Probability-based Information Flow Analysis. *IEEE Transactions on Software Engineering*, 36(5):719–734, 2010. doi:10.1109/TSE.2010.4.

28 Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. Weak bisimulation for Probabilistic Timed Automata. *Theoretical Computer Science*, 411(50):4291–4322, 2010. doi:10.1016/j.tcs.2010.09.003.

29 Ruggero Lanotte and Massimo Merro. Semantic Analysis of Gossip Protocols for Wireless Sensor Networks. In *22nd International Conference on Concurrency Theory*, volume 6901 of *LNCS*, pages 156–170. Springer, 2011. doi:10.1007/978-3-642-23217-6_11.

30 Ruggero Lanotte, Massimo Merro, and Simone Tini. Weak Simulation Quasimetric in a Gossip Scenario. In *37th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, volume 10321 of *LNCS*, pages 139–155. Springer, 2017. doi:10.1007/978-3-319-60225-7_10.

31 Matteo Mio. Upper-Expectation Bisimilarity and Łukasiewicz μ-Calculus. In *17th International Conference on Foundations of Software Science and Computation Structures*, volume 8412 of *LNCS*, pages 335–350. Springer, 2014. doi:10.1007/978-3-642-54830-7_22.

32 Sandro Rafaeli and David Hutchison. A Survey of Key Management for Secure Group Communication. *ACM Comput. Surv.*, 35(3):309–329, 2003. doi:10.1145/937503.937506.

33 Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995.

34 William J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

35 Frank van Breugel and J. Worrell. A Behavioural Pseudometric for Probabilistic Transition Systems. *Theor. Comput. Sci.*, 331(1):115–142, 2005. doi:10.1016/j.tcs.2006.05.021.

36 Rob J. van Glabbeek and Peter W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996. doi:10.1145/233551.233556.

37 Cédric Villani. *Optimal Transport: Old and New.* Springer, 2009. doi:10.1007/978-3-540-71050-9.

Clique-Width for Graph Classes Closed under Complementation*

Alexandre Blanché[1], Konrad K. Dabrowski[2], Matthew Johnson[3], Vadim V. Lozin[4], Daniël Paulusma[5], and Viktor Zamaraev[6]

1 **École normale supérieure de Rennes, Rennes, France**
alexandre.blanche@ens-rennes.fr
2 **Durham University, Durham, UK**
konrad.dabrowski@durham.ac.uk
3 **Durham University, Durham, UK**
matthew.johnson2@durham.ac.uk
4 **University of Warwick, Coventry, UK**
v.lozin@warwick.ac.uk
5 **Durham University, Durham, UK**
daniel.paulusma@durham.ac.uk
6 **University of Warwick, Coventry, UK**
v.zamaraev@warwick.ac.uk

Abstract

Clique-width is an important graph parameter due to its algorithmic and structural properties. A graph class is hereditary if it can be characterized by a (not necessarily finite) set $\mathcal{H}$ of forbidden induced subgraphs. We initiate a systematic study into the boundedness of clique-width of hereditary graph classes closed under complementation. First, we extend the known classification for the $|\mathcal{H}| = 1$ case by classifying the boundedness of clique-width for every set $\mathcal{H}$ of self-complementary graphs. We then completely settle the $|\mathcal{H}| = 2$ case. In particular, we determine one new class of $(H, \overline{H})$-free graphs of bounded clique-width (as a side effect, this leaves only six classes of (H_1, H_2)-free graphs, for which it is not known whether their clique-width is bounded). Once we have obtained the classification of the $|\mathcal{H}| = 2$ case, we research the effect of forbidding self-complementary graphs on the boundedness of clique-width. Surprisingly, we show that for a set $\mathcal{F}$ of self-complementary graphs on at least five vertices, the classification of the boundedness of clique-width for $(\{H, \overline{H}\} \cup \mathcal{F})$-free graphs coincides with the one for the $|\mathcal{H}| = 2$ case if and only if $\mathcal{F}$ does not include the bull (the only non-empty self-complementary graphs on fewer than five vertices are P_1 and P_4, and P_4-free graphs have clique-width at most 2). Finally, we discuss the consequences of our results for Colouring.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases clique-width, self-complementary graph, forbidden induced subgraph

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.73

1 Introduction

Many graph-theoretic problems that are computationally hard for general graphs may still be solvable in polynomial time if the input graph can be decomposed into large parts of "similarly

* This paper received support from EPSRC (EP/K025090/1 and EP/L020408/1) and the Leverhulme Trust (RPG-2016-258).

behaving" vertices. Such decompositions may lead to an algorithmic speed up and are often defined via some type of graph construction. One particular type is to use vertex labels and to allow certain graph operations, which ensure that vertices labelled alike will always keep the same label and thus behave identically. The clique-width $\mathrm{cw}(G)$ of a graph G is the minimum number of different labels needed to construct G using four such operations (see Section 2 for details). Clique-width has been studied extensively both in algorithmic and structural graph theory. The main reason for its popularity is that, indeed, many well-known NP-hard problems [14, 25, 35, 40], such as Colouring and Hamilton Cycle, become polynomial-time solvable on any graph class $\mathcal{G}$ of *bounded* clique-width, that is, for which there exists a constant c, such that every graph in $\mathcal{G}$ has clique-width at most c. Graph Isomorphism is also polynomial-time solvable on such graph classes [30]. Having bounded clique-width is equivalent to having bounded rank-width [39] and having bounded NLC-width [33], two other well-known width-parameters. However, despite these close relationships, clique-width is a notoriously difficult graph parameter, and our understanding of it is still very limited. For instance, no polynomial-time algorithms are known for computing the clique-width of very restricted graph classes, such as unit interval graphs, or for deciding whether a graph has clique-width at most 4.[1] In order to get a better understanding of clique-width and to identify new "islands of tractability" for central NP-hard problems, many graph classes of bounded and unbounded clique-width have been identified; see, for instance, the Information System on Graph Classes and their Inclusions [24], which keeps a record of such graph classes. In this paper we study the following research question:

What kinds of properties of a graph class ensure that its clique-width is bounded?

We refer to the surveys [31, 34] for examples of such properties. Here, we consider graph complements. The *complement* $\overline{G}$ of a graph G is the graph with vertex set V_G and edge set $\{uv \mid uv \notin E(G)\}$ and has clique-width $\mathrm{cw}(\overline{G}) \leq 2\,\mathrm{cw}(G)$ [15]. This result implies that a graph class $\mathcal{G}$ has bounded clique-width if and only if the class consisting of all complements of graphs in $\mathcal{G}$ has bounded clique-width. Due to this, we initiate a *systematic* study of the boundedness of clique-width for graph classes $\mathcal{G}$ *closed under complementation*, that is, for every graph $G \in \mathcal{G}$, its complement $\overline{G}$ also belongs to $\mathcal{G}$.

To get a handle on graph classes closed under complementation, we restrict ourselves to graph classes that are not only closed under complementation but also under vertex deletion. This is a natural assumption, as deleting a vertex does not increase the clique-width of a graph. A graph class closed under vertex deletion is said to be *hereditary* and can be characterized by a (not necessarily finite) set $\mathcal{H}$ of forbidden induced subgraphs. Over the years many results on the (un)boundedness of clique-width of hereditary graph classes have appeared. We briefly survey some of these results below.

A hereditary graph class of graphs is monogenic or H-free if it can be characterized by one forbidden induced subgraph H, and bigenic or (H_1, H_2)-free if it can be characterized by two forbidden induced subgraphs H_1 and H_2. It is well known (see [23]) that a class of H-free graphs has bounded clique-width if and only if H is an induced subgraph of P_4.[2] By combining known results [3, 5, 7, 8, 9, 10, 11, 17, 18, 21, 38] with new results for bigenic graph classes, Dabrowski and Paulusma [23] classified the (un)boundedness of clique-width of (H_1, H_2)-free graphs for all but 13 pairs (H_1, H_2) (up to an equivalence relation). Afterwards,

[1] It is known that computing clique-width is NP-hard in general [27] and that deciding whether a graph has clique-width at most 3 is polynomial-time solvable [13].

[2] We refer to Section 2 for all the notation used in this section.

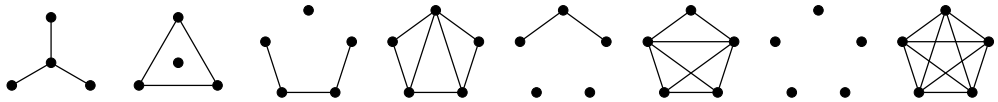


Figure 1 Graphs H for which the clique-width of $(H, \overline{H})$-free graphs is bounded ($s = 5$ is shown).

five new classes of (H_1, H_2)-free graphs were identified by Dross et al. [16] and recently, another one was identified by Dabrowski et al. [19]. This means that only seven cases (H_1, H_2) remained open. Other systematic studies were performed for H-free weakly chordal graphs [5], H-free chordal graphs [5] (two open cases), H-free triangle-free graphs [19] (two open cases), H-free bipartite graphs [22], H-free split graphs [4] (two open cases), and $\mathcal{H}$-free graphs where $\mathcal{H}$ is any set of 1-vertex extensions of the P_4 [6] or any set of graphs on at most four vertices [7]. Clique-width results or techniques for these graph classes impacted upon each other and could also be used for obtaining new results for bigenic graph classes.

Our Contribution. Recall that we investigate the clique-width of hereditary graph classes closed under complementation. A graph that contains no induced subgraph isomorphic to a graph in a set $\mathcal{H}$ is said to be *$\mathcal{H}$-free*. We first consider the $|\mathcal{H}| = 1$ case. The class of H-free graphs is closed under complementation if and only if H is a self-complementary graph, that is, $H = \overline{H}$. Self-complementary graphs have been extensively studied; see [26] for a survey. From the aforementioned result for P_4-free graphs, we find that the only self-complementary graphs H for which the class of H-free graphs has bounded clique-width are $H = P_1$ and $H = P_4$. In Section 3 we prove the following generalization of this result.

▶ **Theorem 1.** *Let $\mathcal{H}$ be a set of non-empty self-complementary graphs. Then the class of $\mathcal{H}$-free graphs has bounded clique-width if and only if either $P_1 \in \mathcal{H}$ or $P_4 \in \mathcal{H}$.*

We now consider the $|\mathcal{H}| = 2$ case. Let $\mathcal{H} = \{H_1, H_2\}$. Due to Theorem 1 we may assume $H_2 = \overline{H_1}$ and H_1 is not self-complementary. The class of $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graphs was one of the seven remaining bigenic graph classes, and the only bigenic graph class closed under complementation, for which boundedness of clique-width was open. We settle this case by proving in Section 4 that the clique-width of this class is bounded. In the same section we combine this new result with known results to prove the following theorem, which, together with Theorem 1, shows to what extent the property of being closed under complementation helps with bounding the clique-width for bigenic graph classes (see also Figure 1).

▶ **Theorem 2.** *For a graph H, the class of $(H, \overline{H})$-free graphs has bounded clique-width if and only if H or $\overline{H}$ is an induced subgraph of $K_{1,3}$, $P_1 + P_4$, $2P_1 + P_3$ or sP_1 for some $s \geq 1$.*

For the $|\mathcal{H}| = 3$ case, where $\{H_1, H_2, H_3\} = \mathcal{H}$, we observe that a class of (H_1, H_2, H_3)-free graphs is closed under complementation if and only if either every H_i is self-complementary, or one H_i is self-complementary and the other two graphs H_j and H_k are complements of each other. By Theorem 1, we only need to consider $(H_1, \overline{H_1}, H_2)$-free graphs, where H_1 is not self-complementary, H_2 is self-complementary, and neither H_1 nor H_2 is an induced subgraph of P_4. The next two smallest self-complementary graphs H_2 are the C_5 and the bull (see also Figure 2). Observe that any self-complementary graph on n vertices must contain $\frac{1}{2}\binom{n}{2}$ edges and this number must be an integer, so $n = 4q$ or $n = 4q + 1$ for some integer $q \geq 0$. There are exactly ten non-isomorphic self-complementary graphs on eight vertices [41] and we depict these in Figure 3.

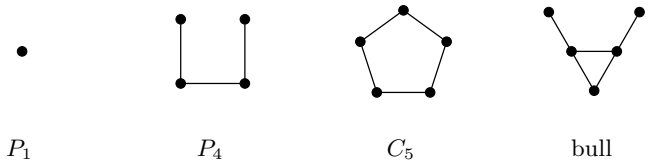


Figure 2 The four non-empty self-complementary graphs on less than eight vertices [41].

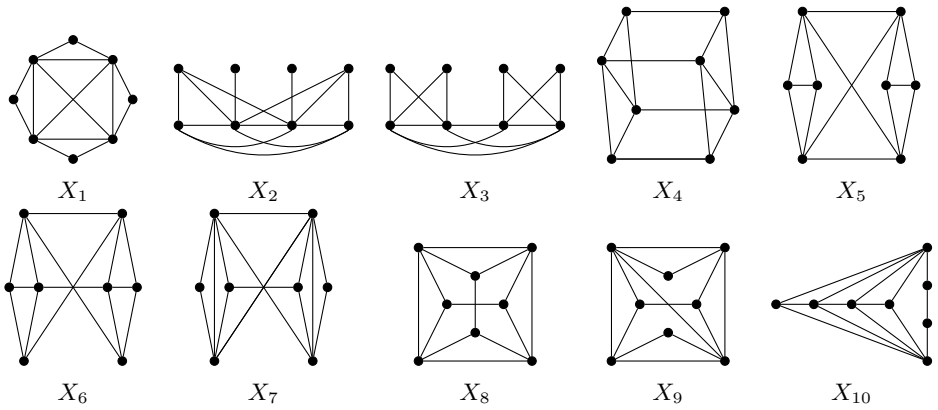


Figure 3 The ten self-complementary graphs on eight vertices [41].

It is known that split graphs, or equivalently, $(2P_2, \overline{2P_2}, C_5)$-free graphs have unbounded clique-width [38]. In Section 5 we determine three new hereditary graph classes of unbounded clique-width, which imply that the class of $(H, \overline{H}, C_5)$-free graphs has unbounded clique-width if $H \in \{K_{1,3}+P_1, 2P_2, 3P_1+P_2, S_{1,1,2}\}$. By combining this with known results, we discovered that the classification of boundedness of clique-width for $(H, \overline{H}, C_5)$-free graphs coincides with the one of Theorem 2. This raised the question of whether the same is true for other sets of self-complementary graphs $\mathcal{F} \neq \{C_5\}$. If $\mathcal{F}$ contains the bull, then the answer is negative: by Theorem 2, the class of $(S_{1,1,2}, \overline{S_{1,1,2}})$-free graphs and the class of $(2P_2, C_4)$-free graphs both have unbounded clique-width, but both the class of $(S_{1,1,2}, \overline{S_{1,1,2}}, \text{bull})$-free graphs and even the class of $(P_5, \overline{P_5}, \text{bull})$-free graphs have bounded clique-width [6]. However, also in Section 5, we prove that the bull is the *only* exception (apart from the trivial cases when $H' \in \{P_1, P_4\}$ which yield bounded clique-width of $(H, \overline{H}, H')$-free graphs for any graph H).

▶ **Theorem 3.** *Let $\mathcal{F}$ be a set of self-complementary graphs on at least five vertices not equal to the bull. For a graph H, the class of $(\{H, \overline{H}\} \cup \mathcal{F})$-free graphs has bounded clique-width if and only if H or $\overline{H}$ is an induced subgraph of $K_{1,3}$, P_1+P_4, $2P_1+P_3$ or sP_1 for some $s \geq 1$.*

Consequences. Due to our result for $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graphs, we can update the summary of [19] for the clique-width of bigenic graph classes and reduce the number of open cases from seven to six.

▶ **Open Problem 4.** *Have (H_1, H_2)-free graphs bounded or unbounded clique-width when:*

(i) $H_1 = 3P_1$ *and* $\overline{H_2} \in \{P_1 + S_{1,1,3}, S_{1,2,3}\}$;
(ii) $H_1 = 2P_1 + P_2$ *and* $\overline{H_2} \in \{P_1 + P_2 + P_3, P_1 + P_5\}$;
(iii) $H_1 = P_1 + P_4$ *and* $\overline{H_2} \in \{P_1 + 2P_2, P_2 + P_3\}$.

Another consequence of our result for $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graphs is that Colouring is polynomial-time solvable for this graph class. This result was used by Blanché et al. [2]:

▶ **Theorem 5** ([2]). *Let $H, \overline{H} \notin \{(s+1)P_1 + P_3, sP_1 + P_4 \mid s \geq 2\}$. Then* Colouring *is polynomial-time solvable for $(H, \overline{H})$-free graphs if H or $\overline{H}$ is an induced subgraph of $K_{1,3}$, $P_1 + P_4$, $2P_1 + P_3$, $P_2 + P_3$, P_5, or $sP_1 + P_2$ for some $s \geq 0$ and it is* NP*-complete otherwise.*

Comparing Theorems 2 and 5 shows that there are graph classes of unbounded clique-width closed under complementation for which Colouring is polynomial-time solvable. Nevertheless, on many graph classes, polynomial-time solvability of NP-hard problems stems from the underlying property of having bounded clique-width. The present paper illustrates this for the Colouring problem, since Theorem 28 implies that Colouring is solvable in polynomial time on $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graphs. By updating the summary of [16] (see also [29]), we find that there are twelve classes of (H_1, H_2)-free graphs, for which Colouring could still potentially be solved in polynomial time by showing that their clique-width is bounded.

Future Work. Apart from settling the classification of boundedness of clique-width for (H_1, H_2)-free graphs by addressing Open Problem 4, we aim to continue our study of boundedness of clique-width for graph classes closed under complementation. In particular, to complete the classification for $\mathcal{H}$-free graphs when $|\mathcal{H}| = 3$, we still need to determine those graphs H for which $(H, \overline{H}, \text{bull})$-free graphs have bounded clique-width (there are several cases left).

2 Preliminaries

The *disjoint union* $(V(G) \cup V(H), E(G) \cup E(H))$ of two vertex-disjoint graphs G and H is denoted by $G + H$ and the disjoint union of r copies of a graph G is denoted by rG. For a subset $S \subseteq V(G)$, we let $G[S]$ denote the subgraph of G induced by S. If $S = \{s_1, \ldots, s_r\}$ then, to simplify notation, we may also write $G[s_1, \ldots, s_r]$ instead of $G[\{s_1, \ldots, s_r\}]$. We write $G \setminus S = G[V(G) \setminus S]$; if $S = \{v\}$, we may write $G \setminus v$ instead. We write $G' \subseteq_i G$ to indicate that G' is an induced subgraph of G. The graphs C_r, K_r, $K_{1,r-1}$ and P_r denote the cycle, complete graph, star and path on r vertices, respectively. The graphs K_3 and $K_{1,3}$ are also called the *triangle* and *claw*. The graph $S_{h,i,j}$, for $1 \leq h \leq i \leq j$, denotes the *subdivided claw*, that is, the tree with only one vertex x of degree 3 and exactly three leaves, which are of distance h, i and j from x, respectively. Observe $S_{1,1,1} = K_{1,3}$. We let $\mathcal{S}$ be the class of graphs each connected component of which is either a subdivided claw or a path.

For a set of graphs $\mathcal{H}$, a graph G is *$\mathcal{H}$-free* (or $(\mathcal{H})$-free) if it has no induced subgraph isomorphic to a graph in $\mathcal{H}$. If $\mathcal{H} = \{H_1, \ldots, H_p\}$ for some integer p, then we may write $(H_1, \ldots, H_p)$-free instead of $(\{H_1, \ldots, H_p\})$-free, or, if $p = 1$, we may simply write H_1-free. For a graph $G = (V, E)$, the set $N(u) = \{v \in V \mid uv \in E\}$ denotes the *neighbourhood* of $u \in V$. A graph is *bipartite* if its vertex set can be partitioned into two (possibly empty) independent sets. A graph is *split* if its vertex set can be partitioned into a (possibly empty) independent set and a (possibly empty) clique. Split graphs have been characterized as follows.

▶ **Lemma 6** ([28]). *A graph G is split if and only if it is $(2P_2, C_4, C_5)$-free.*

Let X be a set of vertices in a graph $G = (V, E)$. A vertex $y \in V \setminus X$ is *complete* to X if it is adjacent to every vertex of X and *anti-complete* to X if it is non-adjacent to every

vertex of X. Similarly, a set of vertices $Y \subseteq V \setminus X$ is *complete* (resp. *anti-complete*) to X if every vertex in Y is complete (resp. anti-complete) to X. We say that the edges between two disjoint sets of vertices X and Y form a *matching* (resp. *co-matching*) if each vertex in X has at most one neighbour (resp. non-neighbour) in Y and vice versa (if each vertex has exactly one such neighbour, we say that the matching is *perfect*). A vertex $y \in V \setminus X$ *distinguishes* X if y has both a neighbour and a non-neighbour in X. The set X is a *module* of G if no vertex in $V \setminus X$ distinguishes X. A module X is *non-trivial* if $1 < |X| < |V|$, otherwise it is *trivial*. A graph is *prime* if it has only trivial modules.

To help reduce the amount of case analysis needed to prove Theorems 2 and 3, we will use the following lemma (proof omitted).

▶ **Lemma 7.** *Let $H \in \mathcal{S}$. Then H is $(K_{1,3} + P_1, 2P_2, 3P_1 + P_2, S_{1,1,2})$-free if and only if H is an induced subgraph of $K_{1,3}$, $P_1 + P_4$, $2P_1 + P_3$ or sP_1 for some $s \geq 1$.*

The *clique-width* $\mathrm{cw}(G)$ of a graph G is the minimum number of labels needed to construct G by using the following four operations:

1. creating a new graph consisting of a single vertex v with label i;
2. taking the disjoint union of two labelled graphs G_1 and G_2;
3. joining each vertex with label i to each vertex with label j ($i \neq j$);
4. renaming label i to j.

For an induced subgraph G' (or vertex set $X \subseteq V(G)$) of a graph G, the *subgraph complementation* operation replaces every edge present in G' (resp. $G[X]$) by a non-edge, and vice versa. For two disjoint vertex subsets S and T in G, the *bipartite complementation* operation replaces every edge with one end-vertex in S and the other one in T by a non-edge and vice versa. Let $k \geq 0$ be a constant and let γ be some graph operation. A class $\mathcal{G}'$ is (k, γ)-*obtained* from a class $\mathcal{G}$ if:

1. every graph in $\mathcal{G}'$ is obtained from a graph in $\mathcal{G}$ by performing γ at most k times, and
2. for every $G \in \mathcal{G}$ there exists at least one graph in $\mathcal{G}'$ that is obtained from G by performing γ at most k times.

We say that γ *preserves* boundedness of clique-width if for any finite constant k and any graph class $\mathcal{G}$, any graph class $\mathcal{G}'$ that is (k, γ)-obtained from $\mathcal{G}$ has bounded clique-width if and only if $\mathcal{G}$ has bounded clique-width.

Fact 1. Vertex deletion preserves boundedness of clique-width [36].

Fact 2. Subgraph complementation preserves boundedness of clique-width [34].

Fact 3. Bipartite complementation preserves boundedness of clique-width [34].

We need the following lemmas on clique-width, the first one of which is easy to show.

▶ **Lemma 8.** *The clique-width of a graph of maximum degree at most 2 is at most 4.*

▶ **Lemma 9** ([23])**.** *Let H be a graph. The class of H-free graphs has bounded clique-width if and only if $H \subseteq_i P_4$.*

▶ **Lemma 10** ([37])**.** *Let $\{H_1, \ldots, H_p\}$ be a finite set of graphs. If $H_i \notin \mathcal{S}$ for all $i \in \{1, \ldots, p\}$ then the class of $(H_1, \ldots, H_p)$-free graphs has unbounded clique-width.*

▶ **Lemma 11** ([15])**.** *Let G be a graph and let $\mathcal{P}$ be the set of all induced subgraphs of G that are prime. Then $\mathrm{cw}(G) = \max_{H \in \mathcal{P}} \mathrm{cw}(H)$.*

3 The Proof of Theorem 1

We use the following lemma (proof omitted), which we also need for Theorem 3.

▶ **Lemma 12.** *If G is a (C_4, C_5, K_4)-free self-complementary graph, then $G \subseteq_i$ bull.*

We are now ready to prove Theorem 1. Note that this theorem also holds if $\mathcal{H}$ is infinite.

▶ **Theorem 1** (restated)**.** Let $\mathcal{H}$ be a set of non-empty self-complementary graphs. Then the class of $\mathcal{H}$-free graphs has bounded clique-width if and only if either $P_1 \in \mathcal{H}$ or $P_4 \in \mathcal{H}$.

Proof. Suppose there is a graph $H \in \mathcal{H} \cap \{P_1, P_4\}$. Then the class of $\mathcal{H}$-free graphs is a subclass of the class of P_4-free graphs, which have bounded clique-width by Lemma 9. Now suppose that $\mathcal{H} \cap \{P_1, P_4\} = \emptyset$. The only non-empty self-complementary graphs on at most five vertices that are not equal to P_1 and P_4 are the bull and the C_5 (see Figure 2). By Lemma 12, it follows that every graph in $\mathcal{H}$ contains an induced subgraph isomorphic to the bull, C_4, C_5 or K_4. Therefore the class of $\mathcal{H}$-free graphs contains the class of (bull, C_4, C_5, K_4)-free graphs, which has unbounded clique-width by Lemma 10. ◀

4 The Proof of Theorem 2

In this section we prove Theorem 2 by combining known results with the new result that $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graphs have bounded clique-width. We prove this result in the following way. We first prove two useful structural lemmas, namely Lemmas 13 and 14, which we will use repeatedly throughout the proof. Next, we prove Lemmas 15 and 16, which state that if a $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graph G contains an induced C_5 or C_6, respectively, then G has bounded clique-width. We do this by partitioning the vertices outside this cycle into sets, depending on their neighbourhood in the cycle. We then analyse the edges within these sets and between pairs of such sets. After a lengthy case analysis, we find that G has bounded clique-width in both these cases. By Fact 2 it only remains to analyse $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graphs that are also $(C_5, C_6, \overline{C_6})$-free. Next, in Lemma 17, we show that if such graphs are prime, then they are either K_7-free or $\overline{K_7}$-free. In Lemma 27 we use the fact that $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graphs are χ-bounded to deal with the case where a graph in the class is K_7-free. Finally, we combine all these results together to obtain the new result (Theorem 28). We omit the proofs of Lemmas 13–16.

▶ **Lemma 13.** *Let G be a $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graph whose vertex set can be partitioned into two sets X and Y, each of which is a clique or an independent set. Then by deleting at most one vertex from each of X and Y, it is possible to obtain subsets such that the edges between them form a matching or a co-matching.*

▶ **Lemma 14.** *Let G be a $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graph whose vertex set can be partitioned into a clique X and an independent set Y. Then by deleting at most three vertices from each of X and Y, it is possible to obtain subsets that are either complete or anti-complete to each other.*

▶ **Lemma 15.** *The class of $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graphs containing an induced C_5 has bounded clique-width.*

▶ **Lemma 16.** *The class of $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graphs containing an induced C_6 has bounded clique-width.*

▶ **Lemma 17.** *Every prime $(2P_1 + P_3, \overline{2P_1 + P_3}, C_6, \overline{C_6})$-free graph is K_7-free or $\overline{K_7}$-free.*

Proof. Let G be a prime $(2P_1 + P_3, \overline{2P_1 + P_3}, C_6, \overline{C_6})$-free graph. Suppose, for contradiction, that G contains an induced K_7 and an induced $\overline{K_7}$. We will show that in this case the graph G is not prime. Note that any induced K_7 and induced $\overline{K_7}$ in G can share at most one vertex. We may therefore assume that G contains a clique C on at least six vertices and a vertex-disjoint independent set I on at least six vertices. Furthermore, we may assume that C is a maximum clique in $G \setminus I$ and I is a maximum independent set in $G \setminus C$ (if not, then replace C or I with a bigger clique or independent set, respectively).

By Lemma 14, there exist sets $R_1 \subset C$ and $R_2 \subset I$ each of size at most 3 such that $C' = C \setminus R_1$ is either complete or anti-complete to $I' = I \setminus R_2$. Without loss of generality, we may assume that R_1 and R_2 are minimal, in the sense that the above property does not hold if we remove any vertex from R_1 or R_2. Note that the class of prime $(2P_1+P_3, \overline{2P_1 + P_3}, C_6, \overline{C_6})$-free graphs containing an induced K_7 and an induced $\overline{K_7}$ is closed under complementation. Therefore, complementing G if necessary (in which case the sets I and C will be swapped, and the sets R_1 and R_2 will be swapped), we may assume that C' is anti-complete to I'.

▶ **Claim 18.** $|R_1| \leq 1$ and $|R_2| \leq 1$.

By construction, R_1 and R_2 each contain at most three vertices and I' and C' each contain at least three vertices. Since R_1 (resp. R_2) is minimal, every vertex of R_1 (resp. R_2) has at least one neighbour in I' (resp. C').

Choose $i_1, i_2 \in I'$ arbitrarily and suppose, for contradiction, that $y \in R_2$ is not complete to C'. Then y must have a neighbour $c_1 \in C'$ and a non-neighbour $c_2 \in C'$, so $G[i_1, i_2, y, c_1, c_2]$ is a $2P_1 + P_3$, a contradiction. Therefore R_2 is complete to C'. If $y, y' \in R_2$ then for arbitrary $c_1 \in C'$, the graph $G[i_1, i_2, y, c_1, y']$ is a $2P_1 + P_3$, a contradiction. It follows that $|R_2| \leq 1$.

Choose $c_1, c_2 \in C'$ arbitrarily. Suppose, for contradiction, that $x \in R_1$ has two non-neighbours $i_1, i_2 \in I'$. Recall that x must have a neighbour $i_3 \in I'$, so $G[i_1, i_2, i_3, x, c_1]$ is a $2P_1 + P_3$, a contradiction. Therefore every vertex of R_1 has at most one non-neighbour in I'. Suppose, for contradiction, that $x, x' \in R_1$. Since I' contains at least three vertices, there must be a vertex $i_1 \in I'$ that is a common neighbour of x and x'. Now $G[x, x', c_1, i_1, c_2]$ is a $\overline{2P_1 + P_3}$, a contradiction. It follows that $|R_1| \leq 1$. This completes the proof of Claim 18.

Note that Claim 18 implies that $|C'| \geq 5$ and $|I'| \geq 5$. Let A be the set of vertices in $V \setminus (C \cup I)$ that are complete to C'. If $x \in A$ is adjacent to $y \in R_1$ then by Claim 18 $C \cup \{x\}$ is a bigger clique than C, contradicting the maximality of C. It follows that A is anti-complete to R_1. If $x, y \in A$ are adjacent then by Claim 18, $(C \cup \{x, y\}) \setminus R_1$ is a bigger clique than C, contradicting the maximality of C. It follows that A is an independent set. Furthermore, by the maximality of I and the definition of A, every vertex in $V \setminus (C \cup I \cup A)$ has a neighbour in I and non-neighbour in C'.

▶ **Claim 19.** Let x be a vertex in $V \setminus (C \cup I \cup A)$. Then either x is complete to I', or x has exactly one neighbour in I.

Suppose, for contradiction, that x has a non-neighbour z in I', and two neighbours $y, y' \in I$. Now x cannot have another non-neighbour $z' \in I \setminus \{z\}$, otherwise $G[z, z', y, x, y']$ would be a $2P_1 + P_3$. Therefore x must be complete to $I \setminus \{z\}$. In particular, since $|I'| \geq 5$, this means that x has two neighbours in I', say y_1 and y_2 (not necessarily distinct from y and y'). Recall that x must have a non-neighbour $c_1 \in C'$. Now $G[c_1, z, y_1, x, y_2]$ is a $2P_1+P_3$. This contradiction completes the proof of Claim 19.

By Claim 19 we can partition the vertex set $V \setminus (C \cup I \cup A)$ into subsets $V_{I'}$ and V_x for every $x \in I$, where $V_{I'}$ is the set of vertices that are complete to I', and V_x is the set of vertices whose unique neighbour in I is x. Let $U_x = V_x \cup \{x\}$.

▶ **Claim 20.** For all $x \in I'$, U_x is anti-complete to C'.

Suppose $x \in I'$. Clearly x is anti-complete to C'. Suppose, for contradiction, that $y \in U_x \setminus \{x\} = V_x$ has a neighbour $z \in C'$ and choose $u, v \in I' \setminus \{x\}$. Then $G[u, v, x, y, z]$ is a $2P_1 + P_3$. This contradiction completes the proof of Claim 20.

▶ **Claim 21.** For every $x \in I$, the set U_x is a clique.

Note that $x \in I$ is adjacent to all other vertices of U_x, by definition. If $y, z \in V_x$ are non-adjacent then $(I \setminus \{x\}) \cup \{y, z\}$ would be a bigger independent set than I, a contradiction.

▶ **Claim 22.** If $x, y \in I$ are distinct, then U_x is anti-complete to U_y.

Clearly x is anti-complete to U_y and y is anti-complete to U_x. Suppose, for contradiction, that $x' \in U_x \setminus \{x\}$ is adjacent to $y' \in U_y \setminus \{y\}$. Choose $u, v \in I \setminus \{x, y\}$. Then $G[u, v, x, x', y']$ is a $2P_1 + P_3$. This contradiction completes the proof of Claim 22.

▶ **Claim 23.** For every $x \in I'$, the set U_x is complete to $V_{I'}$.

By definition, x is complete to $V_{I'}$. Suppose, for contradiction that $x' \in U_x \setminus \{x\}$ is non-adjacent to $y \in V_{I'}$. As $y \notin A$, the vertex y must have a non-neighbour $c_1 \in C'$ and note that x' is non-adjacent to c_1 by Claim 20. Choose $u, v \in I' \setminus \{x\}$. Then $G[c_1, x', u, y, v]$ is a $2P_1 + P_3$. This contradiction proves Claim 23.

Suppose $x \in I'$. Claim 21 implies that U_x is a clique, Claim 20 that U_x is anti-complete to C' and Claim 23 that U_x is complete to $V_{I'}$. Furthermore for all $y \in I \setminus \{x\}$, Claim 22 implies that U_x is anti-complete to U_y. We conclude that given any two vertices $x, y \in I'$, no vertex in $V \setminus (A \cup R_1 \cup U_x \cup U_y)$ can distinguish the set $U_x \cup U_y$. In the remainder of the proof, we will show there exist $x, y \in I'$ such that no vertex of $A \cup R_1$ distinguishes the set $U_x \cup U_y$, meaning that $U_x \cup U_y$ is a non-trivial module, contradicting the assumption that G is prime.

▶ **Claim 24.** If $u \in A \cup R_1$ then either u is anti-complete to U_x for all $x \in I'$ or else u is complete to U_x for all but at most one $x \in I'$.

Suppose, for contradiction, that the claim does not hold for a vertex $u \in A \cup R_1$. Then u must have a neighbour $x' \in U_x$ for some $x \in I'$ and must have non-neighbours $y' \in U_y$ and $z' \in U_z$ for some $y, z \in I'$ with $y \neq z$. Since $|I'| \geq 5$, we may also assume that $x \notin \{y, z\}$. Choose $c_1 \in C'$ arbitrarily. By Claim 20, c_1 is non-adjacent to x', y' and z'. It follows that $G[y', z', c_1, u, x']$ is a $2P_1 + P_3$. This contradiction completes the proof of Claim 24.

Let A^* denote the set of vertices in $A \cup R_1$ that have a neighbour in U_x for some $x \in I'$.

▶ **Claim 25.** The set A^* is complete to all, except possibly two, sets $U_x, x \in I'$.

Suppose, for contradiction, that there are three different vertices $x, y, z \in I'$ such that A^* is complete to none of the sets U_x, U_y, and U_z. By Claim 24 and the definition of A^*, every vertex in A^* is complete to at least two of the sets U_x, U_y, U_z. Therefore there exist three vertices $u, v, w \in A^*$ such that:

- u is not adjacent to some vertex $x' \in U_x$, but is complete to U_y and U_z;
- v is not adjacent to some vertex $y' \in U_y$, but is complete to U_x and U_z;
- w is not adjacent to some vertex $z' \in U_z$, but is complete to U_x and U_y.

Therefore $G[u, y', w, x', v, z']$ is a C_6. This contradiction completes the proof of Claim 25.

Now, as $|I'| \geq 5$, Claims 24 and 25 imply there exist two distinct vertices $x, y \in I'$ such that every vertex of $A \cup R_1$ is either complete or anti-complete to $U_x \cup U_y$. Hence $U_x \cup U_y$ is a non-trivial module in G, contradicting the fact that G is prime. This completes the proof. ◀

The *chromatic number* $\chi(G)$ of a graph G is the minimum positive integer k such that G is k-colourable. The *clique number* $\omega(G)$ of G is the size of a largest clique in G. A class $\mathcal{C}$ of graphs is *χ-bounded* if there is a function f such that $\chi(G) \leq f(\omega(G))$ for all $G \in \mathcal{C}$.

▶ **Lemma 26** ([32]). *For every natural number k the class of P_k-free graphs is χ-bounded.*

▶ **Lemma 27.** *For $k \geq 1$, $(K_k, 2P_1 + P_3, \overline{2P_1 + P_3})$-free graphs have bounded clique-width.*

Proof. Fix a constant $k \geq 1$ and let G be a $(K_k, 2P_1 + P_3, \overline{2P_1 + P_3})$-free graph. By Lemma 16, we may assume that G is C_6-free. Since G is $(2P_1 + P_3)$-free, it is P_7-free, so by Lemma 26 it has chromatic number at most ℓ for some constant ℓ. This means that we can partition the vertices of G into ℓ independent sets $V_1, \ldots, V_\ell$ (some of which may be empty). By Lemma 13, deleting finitely many vertices (which we may do by Fact 2), we may assume that for all distinct $i, j \in \{1, \ldots, \ell\}$, the edges between V_i and V_j form a matching or a co-matching. Since G is C_6-free, if the vertices between V_i and V_j form co-matching, this co-matching can contain at most two non-edges. Therefore, by deleting finitely many vertices (which we may do by Fact 2), we may assume that the edges between V_i and V_j form a matching or V_i and V_j are complete to each other. By deleting finitely many vertices (which we may do by Fact 2), we may assume that each set V_i is either empty or contains at least five vertices.

Suppose the edges from V_i to V_j and the edges from V_i to V_k form a matching and that there is a vertex $x \in V_i$ that has a neighbour $y \in V_j$ and a neighbour $z \in V_k$. Then y must be adjacent to z, otherwise for $x', x'' \in V_i \setminus \{x\}$ the graph $G[x', x'', y, x, z]$ would be a $2P_1 + P_3$, a contradiction. If V_j is complete to V_k then for $y', y'' \in V_j$, $z' \in V_k$ and $x', x'' \in V_i \setminus (N(y') \cup N(y'') \cup N(z'))$ (such vertices exist since each of y', y'' and z' have at most one neighbour in V_i and V_i contains at least five vertices) we have $G[x', x'', y', z', y'']$ is a $2P_1 + P_3$, a contradiction. Therefore the edges between V_j and V_k form a matching.

Now for each $i, j \in \{1, \ldots, \ell\}$ with $i < j$, if V_i is complete to V_j, then by Fact 2 we may apply a bipartite complementation between V_i and V_j. Let G' be the resulting graph. The previous paragraph implies if x has two neighbours y and z in G' then y is adjacent to z in G, so G' is P_3-free. So G' is a disjoint union of cliques, and thus has clique-width at most 2. ◀

We are now ready to prove our main result.

▶ **Theorem 28.** *The class of $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graphs has bounded clique-width.*

Proof. Let G be a $(2P_1 + P_3, \overline{2P_1 + P_3})$-free graph. By Lemma 11, we may assume that G is prime. If G contains an induced C_6 then we are done by Lemma 16. If G contains an induced $\overline{C_6}$ then we are done by Lemma 16 and Fact 2. We may therefore assume that G is also $(C_6, \overline{C_6})$-free. By Lemma 17, we may assume that G is either K_7-free or $\overline{K_7}$-free. By Fact 2, we may assume that G is K_7-free. Lemma 27 completes the proof. ◀

Combining Theorem 28 with the current state-of-the-art for classifying the boundedness of clique-width for (H_1, H_2)-free graphs (see [20]) yields Theorem 2 (proof omitted).

▶ **Theorem 2** (restated). For a graph H, the class of $(H, \overline{H})$-free graphs has bounded clique-width if and only if H or $\overline{H}$ is an induced subgraph of $K_{1,3}$, $P_1 + P_4$, $2P_1 + P_3$ or sP_1 for some $s \geq 1$.

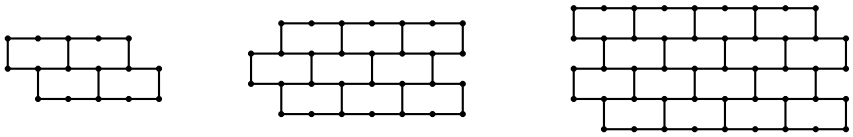


Figure 4 Walls of height 2, 3 and 4, respectively.

5 New Classes of Unbounded Clique-Width and Proof of Theorem 3

In this section we first identify three new graph classes of unbounded clique-width. To do so, we need the notion of a *wall*. Figure 4 shows three walls of different height (see e.g. [12] for a formal definition). The class of walls is well known to have unbounded clique-width; see for example [34]. A *k-subdivided wall* is the graph obtained from a wall after subdividing each edge exactly k times for some constant $k \geq 0$. The following lemma is well known.

▶ **Lemma 29** ([37]). *Let $k \geq 0$. The class of k-subdivided walls has unbounded clique-width.*

Dabrowski et al. [17] showed that $(4P_1, \overline{3P_1 + P_2})$-free graphs have unbounded clique-width. However, their construction was not C_5-free. We give an alternative construction that neither contains an induced C_5 nor an induced copy of any larger self-complementary graph. Namely, we first consider a graph H' that is a 1-subdivided wall. By Lemma 29, such graphs have unbounded clique-width. Let V_1 be the set of vertices in H' that are also present in H. Let V_2 be the set of vertices obtained from subdividing vertical edges in H, and let V_3 be the set of vertices obtained from subdividing horizontal edges. Note that V_1, V_2 and V_3 are independent sets. Furthermore, every vertex in V_1 has at most one neighbour in V_2 and at most two neighbours in V_3, while every vertex in $V_2 \cup V_3$ has at most two neighbours, each of which is in V_1. Let H'' be the graph obtained from H' by applying complementations on V_1, V_2 and V_3. By Fact 2, such graphs have unbounded clique-width. We claim that H'' is $(\{4P_1, \overline{3P_1 + P_2}\} \cup \mathcal{F})$-free, where $\mathcal{F}$ is the set of all self-complementary graphs on at least five vertices that are not equal to the bull (proof omitted). This leads to the following theorem.

▶ **Theorem 30.** *Let $\mathcal{F}$ be the set of all self-complementary graphs on at least five vertices that are not equal to the bull. The class of $(\{4P_1, \overline{3P_1 + P_2}\} \cup \mathcal{F})$-free graphs has unbounded clique-width.*

By Lemma 12, any self-complementary graph on at least five vertices not equal to the bull has an induced subgraph isomorphic to C_4, C_5 or K_4, so such graphs are automatically excluded from the class specified in our next theorem. Its proof, which we omitted, is based on observing that the construction of Brandstädt et al. [7] for proving that $(C_4, K_{1,3}, K_4, \overline{2P_1 + P_2})$-free graphs have unbounded clique-width is, in fact, also C_5-free.

▶ **Theorem 31.** *$(C_4, C_5, K_{1,3}, K_4, \overline{2P_1 + P_2})$-free graphs have unbounded clique-width.*

For our third result we need two lemmas. Given natural numbers k, ℓ, let $Rb(k, \ell)$ denote the smallest number such that if every edge of a $K_{Rb(k,\ell),Rb(k,\ell)}$ is coloured red or blue then it will contain a monochromatic $K_{k,\ell}$. It is known that $Rb(k, \ell)$ always exists [1].

▶ **Lemma 32** ([1]). *$Rb(2, 2) = 5$.*

Let $G = (V, E)$ be a split graph. By definition, G has a *split partition*, that is, a partition of V into two (possibly empty) sets C and I, where C is a clique and I is an independent set. A split graph G may have multiple split partitions. For self-complementary split graphs we can show the following (proof omitted).

▶ **Lemma 33.** *Let G be a self-complementary split graph on n vertices. If n is even, then G has a unique split partition and in this partition the clique and independent set are of equal size. If n is odd, then there exists a vertex v such that $G \setminus v$ is also a self-complementary split graph.*

▶ **Theorem 34.** *Let $\mathcal{F}$ be the set of all self-complementary graphs on at least five vertices that are not equal to the bull. The class of $(\{C_4, 2P_2\} \cup \mathcal{F})$-free graphs has unbounded clique-width.*

Proof. First note that the only self-complementary graph on five vertices apart from the bull is the C_5. Since $C_5 \in \mathcal{F}$, by Lemma 6, we may remove all graphs that are not split from $\mathcal{F}$, apart from C_5; in particular, this means that we remove $X_4, \ldots, X_{10}$ from $\mathcal{F}$ (see also Figure 3). By Lemma 33, if $G \in \mathcal{F}$ has an odd number of vertices, but is not equal to C_5, then $G \setminus v \in \mathcal{F}$ for some vertex $v \in V(G)$. Let $\mathcal{F}'$ be the set of self-complementary split graphs on at least eight vertices that have an even number of vertices. It follows that the class of $\mathcal{F}'$-free split graphs is equal to the class of $(\{C_4, 2P_2\} \cup \mathcal{F})$-free graphs.

Consider a 2-subdivided wall H and note that it is (C_4, C_8)-free; recall that 2-subdivided walls have unbounded clique-width by Lemma 29. Note that H is a bipartite graph, and fix a bipartition (A, B) of H. Let H' be the graph obtained from H by applying a complementation to A and note that H' is a split graph. In H', every vertex in B has a non-neighbour in A and every vertex in A has a neighbour in B, so (A, B) is the unique split partition of H'. By Fact 2, the family of graphs H' produced in this way also has unbounded clique-width. It remains to show that H' is $\mathcal{F}'$-free.

First note that X_1 (see also Figure 3) is the graph obtained from the bipartite graph C_8 by complementing one of the independent sets in the bipartition. Since H is C_8-free and X_1 has a unique split partition (by Lemma 33), it follows that H' is X_1-free. Note that H is C_4-free and so H' does not contain two vertices x, x' in the clique A and two vertices y, y' in the independent set B such that $\{x, x'\}$ is complete to $\{y, y'\}$. Now suppose $G \in \mathcal{F}' \setminus \{X_1\}$. Recall that by Lemma 33, G has a unique split partition (C, I), and this partition has the property that $|C| = |I|$. Therefore, if we can show that G contains two vertices $x, x' \in C$ and two vertices $y, y' \in I$ with $\{x, x'\}$ complete to $\{y, y'\}$ then H' must be G-free and the proof is complete. It is easy to verify that this is the case if $G \in \{X_2, X_3\}$ (see also Figure 3 and recall that $X_4, \ldots, X_{10} \notin \mathcal{F}'$). Otherwise, G has at least ten vertices so $|C|, |I| \geq 5$. By Lemma 32, there must be two vertices $x, x' \in C$ and two vertices $y, y' \in I$ with $\{x, x'\}$ either complete or anti-complete to $\{y, y'\}$. In the first case we are done. In the second case we note that complementing G will swap the sets C and I and make $\{x, x'\}$ complete to $\{y, y'\}$, returning us to the previous case. We conclude that H' is indeed $\mathcal{F}'$-free. ◀

We are now ready to prove Theorem 3. Note that this theorem holds even if $\mathcal{F}$ is infinite.

▶ **Theorem 3** (restated)**.** Let $\mathcal{F}$ be a set of self-complementary graphs on at least five vertices not equal to the bull. For a graph H, the class of $(\{H, \overline{H}\} \cup \mathcal{F})$-free graphs has bounded clique-width if and only if H or $\overline{H}$ is an induced subgraph of $K_{1,3}$, $P_1 + P_4$, $2P_1 + P_3$ or sP_1 for some $s \geq 1$.

Proof. Let H be a graph. By Theorem 2, if H or $\overline{H}$ is an induced subgraph of $K_{1,3}$, $P_1 + P_4$, $2P_1 + P_3$ or sP_1 for some $s \geq 1$, then the class of $(\{H, \overline{H}\} \cup \mathcal{F})$-free graphs has bounded clique-width. Consider a graph $F \in \mathcal{F}$. Since F contains at least five vertices and is not isomorphic to the bull, Lemma 12 implies that F contains an induced subgraph isomorphic to C_4, C_5 or K_4, and so $F \notin \mathcal{S}$. Therefore the class of $(\{H, \overline{H}\} \cup \mathcal{F})$-free graphs contains the class of $(H, \overline{H}, C_4, C_5, K_4)$-free graphs. If $H \notin \mathcal{S}$ and $\overline{H} \notin \mathcal{S}$, then the class of $(H, \overline{H}, C_4, C_5, K_4)$-free graphs has unbounded clique-width by Lemma 10. By Fact 2, we

may therefore assume that $H \in \mathcal{S}$. By Lemma 7, we may assume H contains $K_{1,3}+P_1$, $2P_2$, $3P_1+P_2$ or $S_{1,1,2}$ as an induced subgraph, otherwise we are done. In this case, the class of $(\{H,\overline{H}\}\cup\mathcal{F})$-free graphs contains the class of $(K_{1,3},K_4,C_4,C_5)$-free, $(\{2P_2,C_4\}\cup\mathcal{F})$-free, $(\{4P_1,\overline{3P_1+P_2}\}\cup\mathcal{F})$-free or $(K_{1,3},\overline{2P_1+P_2},C_4,C_5,K_4)$-free graphs, respectively. These classes have unbounded clique-width by Theorems 31, 34, 30 and 31, respectively. This completes the proof. ◀

References

1 Lowell W. Beineke and Allen J. Schwenk. On a bipartite form of the Ramsey problem. *Congressus Numerantium*, XV:17–22, 1975.

2 Alexandre Blanché, Konrad K. Dabrowski, Matthew Johnson, and Daniël Paulusma. Hereditary graph classes: When the complexities of Colouring and Clique Cover coincide. *CoRR*, abs/1607.06757, 2016.

3 Rodica Boliac and Vadim V. Lozin. On the clique-width of graphs in hereditary classes. *Proc. ISAAC 2002, LNCS*, 2518:44–54, 2002.

4 Andreas Brandstädt, Konrad K. Dabrowski, Shenwei Huang, and Daniël Paulusma. Bounding the clique-width of H-free split graphs. *Discrete Applied Mathematics*, 211:30–39, 2016.

5 Andreas Brandstädt, Konrad K. Dabrowski, Shenwei Huang, and Daniël Paulusma. Bounding the clique-width of H-free chordal graphs. *Journal of Graph Theory*, (in press), 2017.

6 Andreas Brandstädt, Feodor F. Dragan, Hoàng-Oanh Le, and Raffaele Mosca. New graph classes of bounded clique-width. *Theory of Computing Systems*, 38(5):623–645, 2005.

7 Andreas Brandstädt, Joost Engelfriet, Hoàng-Oanh Le, and Vadim V. Lozin. Clique-width for 4-vertex forbidden subgraphs. *Theory of Computing Systems*, 39(4):561–590, 2006.

8 Andreas Brandstädt, Tilo Klembt, and Suhail Mahfud. P_6- and triangle-free graphs revisited: structure and bounded clique-width. *Discrete Mathematics and Theoretical Computer Science*, 8(1):173–188, 2006.

9 Andreas Brandstädt, Hoàng-Oanh Le, and Raffaele Mosca. Gem- and co-gem-free graphs have bounded clique-width. *International Journal of Foundations of Computer Science*, 15(1):163–185, 2004.

10 Andreas Brandstädt, Hoàng-Oanh Le, and Raffaele Mosca. Chordal co-gem-free and (P_5,gem)-free graphs have bounded clique-width. *Discrete Applied Mathematics*, 145(2):232–241, 2005.

11 Andreas Brandstädt and Suhail Mahfud. Maximum weight stable set on graphs without claw and co-claw (and similar graph classes) can be solved in linear time. *Information Processing Letters*, 84(5):251–259, 2002.

12 Julia Chuzhoy. Improved bounds for the flat wall theorem. *Proc. SODA 2015*, pages 256–275, 2015.

13 Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce A. Reed, and Udi Rotics. Polynomial-time recognition of clique-width ≤ 3 graphs. *Discrete Applied Mathematics*, 160(6):834–865, 2012.

14 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

15 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1–3):77–114, 2000.

16 Konrad K. Dabrowski, François Dross, and Daniël Paulusma. Colouring diamond-free graphs. *Journal of Computer and System Sciences*, (to appear).

17 Konrad K. Dabrowski, Petr A. Golovach, and Daniël Paulusma. Colouring of graphs with Ramsey-type forbidden subgraphs. *Theoretical Computer Science*, 522:34–43, 2014.

18 Konrad K. Dabrowski, Shenwei Huang, and Daniël Paulusma. Bounding clique-width via perfect graphs. *Journal of Computer and System Sciences*, (in press).

19 Konrad K. Dabrowski, Vadim V. Lozin, and Daniël Paulusma. Clique-width and well-quasi ordering of triangle-free graph classes. *Proc. WG 2017, LNCS*, (to appear).

20 Konrad K. Dabrowski, Vadim V. Lozin, and Daniël Paulusma. Well-quasi-ordering versus clique-width: New results on bigenic classes. *Order*, (to appear).

21 Konrad K. Dabrowski, Vadim V. Lozin, Rajiv Raman, and Bernard Ries. Colouring vertices of triangle-free graphs without forests. *Discrete Mathematics*, 312(7):1372–1385, 2012.

22 Konrad K. Dabrowski and Daniël Paulusma. Classifying the clique-width of H-free bipartite graphs. *Discrete Applied Mathematics*, 200:43–51, 2016.

23 Konrad K. Dabrowski and Daniël Paulusma. Clique-width of graph classes defined by two forbidden induced subgraphs. *The Computer Journal*, 59(5):650–666, 2016.

24 H. N. de Ridder et al. *Information System on Graph Classes and their Inclusions*, 2001–2013. `http://www.graphclasses.org`.

25 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. *Proc. WG 2001, LNCS*, 2204:117–128, 2001.

26 Alastair Farrugia. Self-complementary graphs and generalisations: a comprehensive reference manual. Master's thesis, University of Malta, 1999.

27 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-Complete. *SIAM Journal on Discrete Mathematics*, 23(2):909–939, 2009.

28 Stéphane Földes and Peter Ladislaw Hammer. Split graphs. *Congressus Numerantium*, XIX:311–315, 1977.

29 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of colouring graphs with forbidden subgraphs. *Journal of Graph Theory*, 84(4):331–363, 2017.

30 Martin Grohe and Pascal Schweitzer. Isomorphism testing for graphs of bounded rank width. *Proc. FOCS 2015*, pages 1010–1029, 2015.

31 Frank Gurski. The behavior of clique-width under graph operations and graph transformations. *Theory of Computing Systems*, 60(2):346–376, 2017.

32 András Gyárfás. Problems from the world surrounding perfect graphs. *Applicationes Mathematicae*, 19(3-4):413–441, 1987.

33 Öjvind Johansson. Clique-decomposition, NLC-decomposition, and modular decomposition - relationships and results for random graphs. *Congressus Numerantium*, 132:39–60, 1998.

34 Marcin Kamiński, Vadim V. Lozin, and Martin Milanič. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157(12):2747–2761, 2009.

35 Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126(2–3):197–221, 2003.

36 Vadim V. Lozin and Dieter Rautenbach. On the band-, tree-, and clique-width of graphs with bounded vertex degree. *SIAM Journal on Discrete Mathematics*, 18(1):195–206, 2004.

37 Vadim V. Lozin and Dieter Rautenbach. The tree- and clique-width of bipartite graphs in special classes. *Australasian Journal of Combinatorics*, 34:57–67, 2006.

38 Johann A. Makowsky and Udi Rotics. On the clique-width of graphs with few P_4's. *International Journal of Foundations of Computer Science*, 10(03):329–348, 1999.

39 Sang-Il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

40 Michaël Rao. MSOL partitioning problems on graphs of bounded treewidth and clique-width. *Theoretical Computer Science*, 377(1–3):260–267, 2007.

41 Ronald C. Read. On the number of self-complementary graphs and digraphs. *Journal of the London Mathematical Society*, s1-38(1):99–104, 1963.

Computing the Maximum Using (min,+) Formulas

Meena Mahajan[1], Prajakta Nimbhorkar[2], and Anuj Tawari[3]

1 **The Institute of Mathematical Sciences, HBNI, Chennai, India**
meena@imsc.res.in
2 **Chennai Mathematical Institute, India**
prajakta@cmi.ac.in
3 **The Institute of Mathematical Sciences, HBNI, Chennai, India**
anujvt@imsc.res.in

Abstract

We study computation by formulas over $(\min, +)$. We consider the computation of $\max\{x_1, \ldots, x_n\}$ over $\mathbb{N}$ as a difference of $(\min, +)$ formulas, and show that size $n + n \log n$ is sufficient and necessary. Our proof also shows that any $(\min, +)$ formula computing the minimum of all sums of $n - 1$ out of n variables must have $n \log n$ leaves; this too is tight. Our proofs use a complexity measure for $(\min, +)$ functions based on minterm-like behaviour and on the entropy of an associated graph.

1998 ACM Subject Classification F.1.1 Models of Computation; F.1.3 Complexity Measures and Classes; F.2.3 Tradeoffs between Complexity Measures

Keywords and phrases formulas, circuits, lower bounds, tropical semiring

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.74

1 Introduction

A $(\min, +)$ formula is a formula (tree) in which the leaves are labeled by variables or constants. The internal nodes are gates labeled by either min or +. A min gate computes the minimum value among its inputs while a + gate simply adds the values computed by its inputs. Such formulas can compute any function expressible as the minimum over several linear polynomials with non-negative integer coefficients.

In this work, we consider the following problem: Suppose we are given n input variables $x_1, x_2, \ldots, x_n$ and we want to find a formula which computes the maximum value taken by these variables, $\max(x_1, x_2, \ldots, x_n)$. If variables are restricted to take non-negative integer values, t is easy to show that no $(\min, +)$ formula can compute max. Suppose now we strengthen this model by allowing minus gates as well. Now we have a very small linear sized formula: $\max(x_1, x_2, \ldots, x_n) = 0 - \min(0 - x_1, 0 - x_2, \ldots, 0 - x_n)$. It is clear that minus gates add significant power to the model of $(\min, +)$ formulas. But how many minuses do we actually need? It turns out that only one minus gate, at the top, is sufficient. Here is one such formula: (Sum of all variables) - $\min_i$ (Sum of all variables except x_i). The second expression above can be computed by a $(\min, +)$ formula of size $n \log n$ using recursion. So, we can compute max using min, + and one minus gate at the top, at the cost of a slightly super-linear size. Can we do any better? We show that this simple difference formula is indeed the best we can achieve for this model.

The main motivation behind studying this question is the following question asked in [8]: Does there exist a naturally occuring function f for which $(\min, +)$ circuits are super-polynomially weaker than $(\max, +)$ circuits? There are two possibilities:

1. Show that max can be implemented using a small $(\min, +)$ circuit.
2. Come up with an explicit function f which has *small* $(\max, +)$ circuits but requires *large* $(\min, +)$ circuits.

Since we show that no $(\min, +)$ formula (or circuit) can compute max, option 1 is ruled out. In the weaker model of formulas instead of circuits, we show that any difference of $(\min, +)$ formulas computing max should have size at least $n \log n$. This yields us a separation between $(\max, +)$ formulas and difference of $(\min, +)$ formulas.

Background

Many dynamic programming algorithms correspond to $(\min, +)$ circuits over an appropriate semiring. Notable examples include the Bellman-Ford-Moore (BFM) algorithm for the single-source-shortest-path problem (SSSP) [2, 5, 14], the Floyd-Warshall (FW) algorithm for the All-Pairs-Shortest-Path (APSP) problem [4, 18], and the Held-Karp (HK) algorithm for the Travelling Salesman Problem (TSP) [6]. All these algorithms are just recursively constructed $(\min, +)$ circuits. For example, both the BFM and the FW algorithms give $O(n^3)$ sized $(\min, +)$ circuits while the HK algorithm gives a $O(n^2 \cdot 2^n)$ sized $(\min, +)$ circuit. Matching lower bounds were proved for TSP in [7], for APSP in [8], and for SSSP in [10]. So, proving tight lower bounds for circuits over $(\min, +)$ can help us understand the power and limitations of dynamic programming. We refer the reader to [8, 9] for more results on $(\min, +)$ circuit lower bounds.

Note that algorithms for problems like computing the diameter of a graph are naturally expressed using $(\min, \max, +)$ circuits. This makes the cost of converting a max gate to a $(\min, +)$ circuit or formula an interesting measure.

A related question arises in the setting of counting classes defined by arithmetic circuits and formulas. Circuits over $\mathbb{N}$, with specific resource bounds, count accepting computation paths or proof-trees in a related resource-bounded Turing machine model defining a class $\mathcal{C}$. The counting function class is denoted $\#\mathcal{C}$. The difference of two such functions in a class $\#\mathcal{C}$ is a function in the class Diff$\mathcal{C}$. On the other hand, circuits with the same resource bounds, but over $\mathbb{Z}$, or equivalently, with subtraction gates, describe the function class Gap$\mathcal{C}$. For most complexity classes $\mathcal{C}$, a straightforward argument shows that that Diff$\mathcal{C}$ and Gap$\mathcal{C}$ coincide. See [1] for further discussion on this. In this framework, we restrict attention to computation over $\mathbb{N}$ and see that as a member of a Gap class over $(\min, +)$, max has linear-size formulas, whereas as a member of a Diff class, it requires $\Omega(n \log n)$ size.

Our results and techniques

We now formally state our results and briefly comment on the techniques used to prove them.

1. For $n \geq 2$, no $(\min, +)$ formula over $\mathbb{N}$ can compute $\max(x_1, x_2, \ldots, x_n)$. (Theorem 10)
   The proof is simple: apply a carefully chosen restriction to the variables and show that the $(\min, +)$ formula does not output the correct value of max on this restriction.
2. $\max(x_1, x_2, \ldots, x_n)$ can be computed by a difference of two $(\min, +)$ formulas with total size $n + n\lceil \log n \rceil$. More generally, the function computing the sum of the topmost k values amongst the n variables can be computed by a difference of two $(\min, +)$ formulas with total size $n + n(\lceil \log n \rceil)^{\min\{k, n-k\}}$. (Theorem 11)
   Note that the sum of the topmost k values can be computed by the following formula: (Sum of all variables) - $\min_S$ (Sum of all variables except those in S). Here S ranges over all possible subsets of $\{x_1, x_2, \ldots, x_n\}$ of cardinality $n - k$. Using recursion, we obtain the claimed size bound.

3. Let F_1, F_2 be $(\min, +)$ formulas over $\mathbb{N}$ such that $F_1 - F_2 = \max(x_1, x_2, \ldots, x_n)$. Then F_1 must have at least n leaves and F_2 at least $n \log n$ leaves. (Theorem 13)
   A major ingredient in our proof is the definition of a measure for functions computable by constant-free $(\min, +)$ formulas, and relating this measure to formula size. The measure involves terms analogous to minterms of a monotone Boolean function, and uses the entropy of an associated graph under the uniform distribution on its vertices. In the setting of monotone Boolean functions, this technique was used in in [15] to give formula size lower bounds. We adapt that technique to the $(\min, +)$ setting.
   The same technique also yields the following lower bound: Also, any $(\min, +)$ formula computing the minimum over the sums of $n-1$ variables must have at least $n \log n$ leaves. This is tight. (Lemma 12 and Corollary 18)

2 Preliminaries

2.1 Notation

Let X denote the set of variables $\{x_1, \ldots, x_n\}$. We use $\tilde{x}$ to denote $(x_1, x_2, \ldots, x_n, 1)$.

We use e_i to denote the $(n+1)$-dimensional vector with a 1 in the ith coordinate and zeroes elsewhere. For $i \in [n]$, we also use e_i to denote an assignment to the variables $x_1, x_2, \ldots, x_n$ where x_i is set to 1 and all other variables are set to 0.

▶ **Definition 1.** For $0 \leq r \leq n$, the n-variate functions Sum_n, MinSum_n^r and MaxSum_n^r are as defined below.

$$\begin{aligned}
\mathsf{Sum}_n &= \sum_{i=1}^{n} x_i \\
\mathsf{MinSum}_n^r &= \min\left\{\sum_{i\in S} x_i \mid S \subseteq n, |S| = r\right\} \\
\mathsf{MaxSum}_n^r &= \max\left\{\sum_{i\in S} x_i \mid S \subseteq n, |S| = r\right\}
\end{aligned}$$

Note that MinSum_n^0 and MaxSum_n^0 are the constant function 0, and MinSum_n^1 and MaxSum_n^1 are just the min and max respectively.

▶ **Observation 2.** *For* $1 \leq r < n$, $\mathsf{MinSum}_n^n = \mathsf{MaxSum}_n^n = \mathsf{Sum}_n = \mathsf{MinSum}_n^r + \mathsf{MaxSum}_n^{n-r}$.

2.2 Formulas

A $(\min, +)$ formula is a directed tree. Each leaf of a formula has a label from $X \cup \mathbb{N}$; that is, it is labeled by a variable x_i or a constant $\alpha \in \mathbb{N}$. Each internal node has exactly two children and is labeled by one of the two operations min or $+$. The output node of the formula computes a function of the input variables in the natural way. The input nodes of a formula are also referred to as gates.

If all leaves of a formula are labeled from X, we say that the formula is constant-free.

A $(\min, +, -)$ formula is similarly defined; the operation at an internal node may also be $-$, in which case the children are ordered and the node computes the difference of their values.

We define the size of a formula as the number of leaves in the formula. For a formula F, we denote by $L(F)$ its size, the number of leaves in it. For a function f, we denote by $L(f)$ the smallest size of a formula computing f. By $L_{cf}(f)$ we denote the smallest size of a constant-free formula computing f.

2.3 Graph Entropy

The notion of the entropy of a graph or hypergraph, with respect to a probability distribution on its vertices, was first defined by Körner in [11]. In that and subsequent works (e.g. [12, 13, 3, 15]), equivalent characterizations of graph entropy were established and are often used now as the definition itself, see for instance [16, 17]. In this paper, we use graph entropy only with respect to the uniform distribution, and simply call it graph entropy. We use the following definition, which is exactly the definition from [17] specialised to the uniform distribution.

▶ **Definition 3.** Let G be a graph with vertex set $V(G) = \{1, \ldots, n\}$.

The *vertex packing polytope* $VP(G)$ of the graph G is the convex hull of the characteristic vectors of independent sets of G.

The entropy of G is defined as

$$H(G) = \min_{\vec{a} \in VP(G)} \sum_{i=1}^{n} \frac{1}{n} \log \frac{1}{a_i} \quad .$$

It can easily be seen that $H(G)$ is a non-negative real number, and moreover, $H(G) = 0$ if and only if G has no edges. We list non-trivial properties of graph entropy that we use.

▶ **Lemma 4** ([12, 13]). *Let $F = (V, E(F))$ and $G = (V, E(G))$ be two graphs on the same vertex set. The following hold:*

1. ***Monotonocity.*** *If $E(F) \subseteq E(G)$, then $H(F) \leq H(G)$*
2. ***Subadditivity.*** *Let Q be the graph with vertex set V and edge set $E(F) \cup E(G)$. Then $H(Q) \leq H(F) + H(G)$.*

▶ **Lemma 5** (see for instance [16, 17]). *The following hold:*

1. *Let K_n be the complete graph on n vertices. Then $H(K_n) = \log n$.*
2. *Let G be a graph on n vertices, whose edges induce a bipartite graph on m (out of n) vertices. Then $H(G) \leq \frac{m}{n}$.*

3 Transformations and Easy bounds

We consider the computation of $\max\{x_1, \ldots, x_n\}$ over $\mathbb{N}$ using $(\min, +)$ formulas.

To start with, we describe some properties of $(\min, +)$ formulas that we use repeatedly. The first property, Proposition 7 below, is expressing the function computed by a formula as a depth-2 polynomial where $+$ plays the role of multiplication and min plays the role of addition. The next properties, Proposition 8 and 9 below, deal with removing redundant sub-expressions created by the constant zero or moving common parts aside.

▶ **Definition 6.** Let F be a $(\min, +)$ formula with leaves labeled from $X \cup \mathbb{N}$. For each gate $v \in F$, we construct a set $S_v \subseteq \mathbb{N}^{n+1}$ as described below.

We construct the sets inductively based on the depth of v.

1. Case 1. v is a leaf labeled α for some $\alpha \in \mathbb{N}$. Then $S_v = \{\alpha \cdot e_{n+1}\}$. (Recall, e_i is the unit vector with 1 at the ith coordinate and zero elsewhere).
2. Case 2: v is a leaf labeled x_i for some $i \in [n]$. Then $S_v = \{e_i\}$.
3. Case 3: $v = \min\{u, w\}$. Then $S_v = S_u \cup S_w$.
4. Case 4: $v = u + w$. Then $S_v = \{\tilde{a} + \tilde{b} \mid \tilde{a} \in S_u,\ \tilde{b} \in S_w\}$ (coordinate-wise addition).

Let r be the output gate of F. We denote by $S(F)$ the set S_r so constructed.

It is straightforward to see that if F has no constants (so Case 1 is never invoked), then a_{n+1} remains 0 throughout the construction of the sets S_v. Hence if F is constant-free, then for each $\tilde{a} \in S(F)$, $a_{n+1} = 0$.

By construction, the set $S(F)$ describes the function computed by F. Thus we have the following:

▶ **Proposition 7.** *Let F be a formula with* min *and* $+$ *gates, with leaves labeled by elements of $\{x_1, \ldots, x_n\} \cup \mathbb{N}$. For each gate $v \in F$, let f_v denote the function computed at v.*

Then $f_v = \min\{\langle \tilde{a} \cdot \tilde{x} \rangle \mid \tilde{a} \in S_v\}$.

The following proposition is an easy consequence of the construction in Definition 6.

▶ **Proposition 8.** *Let F be a $(\min, +)$ formula over $\mathbb{N}$. Let G be the formula obtained from F by replacing all constants by the constant 0. Let H be the constant-free formula obtained from G by eliminating 0s from G through repeated replacements of $0 + A$ by A, $\min\{0, A\}$ by 0. Then*

1. *$L(H) \leq L(G) = L(F)$,*
2. *$S(G) = \{\tilde{b} \mid b_{n+1} = 0, \exists \tilde{a} \in S(F), \forall i \in [n], a_i = b_i\}$, and*
3. *G and H compute the same function $\min\{\langle \tilde{b} \cdot \tilde{x} \rangle \mid \tilde{b} \in S(G)\}$.*

(Note: It is not the claim that $S(G) = S(H)$. Indeed, this may not be the case. eg. let $F = x + \min\{1, x + y\}$. Then $S(F) = \{101, 210\}$, $S(G) = \{100, 210\}$, $S(H) = \{100\}$, However, the functions computed are the same.)

The next proposition shows how to remove "common" contributors to $S(F)$ without increasing the formula size.

▶ **Proposition 9.** *Let F be a $(\min, +)$ formula computing a function f.*

If, for some $i \in [n]$, $a_i > 0$ for every $\tilde{a} \in S(F)$, then $f - x_i$ can be computed by a $(\min, +)$ formula F' of size at most size(F).

If $a_{n+1} > 0$ for every $\tilde{a} \in S(F)$, then $f - 1$ can be computed by a $(\min, +)$ formula F' of size at most size(F).

In both cases, $S(F') = \{\tilde{b} \mid \exists \tilde{a} \in S(F), \tilde{b} = \tilde{a} - e_i\}$.

Proof. First consider $i \in [n]$. Let X be the subset of nodes in F defined as follows:

$$X = \{v \in F \mid \forall \tilde{a} \in S_v : a_i > 0\}$$

Clearly, the output gate r of F belongs to X. By the construction of the sets S_v, whenever a min node v belongs to X, both its children belong to X, and whenever a $+$ node belongs to X, at least one of its children belongs to X. We pick a set $T \subseteq X$ as follows. Include r in T. For each min node in T, include both its children in T. For each $+$ node in T, include in T one child that belongs to X (if both children are in X, choose any one arbitrarily). This gives a sub-formula of F where all leaves are labeled x_i. Replace these occurrences of x_i in F by 0 to get formula F'. It is easy to see that $S(F') = \{\tilde{a} - e_i \mid \tilde{a} \in S\}$. Hence F' computes $f - x_i$.

For $i = a_{n+1}$, the same process as above yields a subformula where each leaf is labeled by a positive constant. Subtracting 1 from the constant at each leaf in T gives the formula computing $f - 1$. ◀

It is intuitively clear that no $(\min, +)$ formula can compute max. A formal proof using Proposition 7 appears below.

▶ **Theorem 10.** *For $n \geq 2$, no $(min, +)$ formula over $\mathbb{N}$ can compute $\max\{x_1, \ldots, x_n\}$.*

Proof. Suppose, to the contrary, some formula C computes max. Then its restriction D to $x_1 = X$, $x_2 = Y$, $x_3 = x_4 = \ldots = x_n = 0$, correctly computes $\max\{X, Y\}$. Since all leaves of D are labeled from $\{x_1, x_2\} \cup \mathbb{N}$, the set $S(D)$ is a set of triples. Let $S \subseteq \mathbb{N}^3$ be this set. For all $X, Y \in \mathbb{N}$, $\max\{X, Y\}$ equals $E(X, Y) = \min\{AX + BY + C \mid (A, B, C) \in S\}$.

Let K denote the maximum value taken by C in any triple in S. If for some $B, C \in \mathbb{N}$, the triple $(0, B, C)$ belongs to S, then $E(K + 1, 0) \leq C \leq K < K + 1 = \max\{0, K + 1\}$. So for all $(A, B, C) \in S$, $A \neq 0$, so $A \geq 1$. Similarly, for all $(A, B, C) \in S$, $B \geq 1$. Hence for all $(A, B, C) \in S$, $A + B \geq 2$.

Now $E(1, 1) = \min\{A + B + C \mid (A, B, C) \in S\} \geq 2 > 1 = \max\{1, 1\}$. So $E(X, Y)$ does not compute $\max(X, Y)$ correctly. ◀

However, if we also allow the subtraction operation at internal nodes, it is very easy to compute the maximum in linear size; $\max(x_1, \ldots, x_n) = -\min\{-x_1, -x_2, \ldots, -x_n\}$. Here $-a$ is implemented as $0 - a$, and if we allow only variables, not constants, at leaves, we can compute $-a$ as $(x_1 - x_1) - a$.

Thus the subtraction operation adds significant power. How much? Can we compute the maximum with very few subtraction gates? It turns out that the max function can be computed as the difference of two $(\min, +)$ formulas. Equivalently, there is a $(\min, +, -)$ formula with a single $-$ gate at the root, that computes the max function. This formula is not linear in size, but it is not too big either; we show that it has size $O(n \log n)$. A simple generalisation allows us to compute the sum of the largest k values.

▶ **Theorem 11.** *For each $n \geq 1$, and each $0 \leq k \leq n$, the function MaxSum_n^k can be computed by a difference of two $(\min, +)$ formulas with total size $n + n(\lceil \log n \rceil)^{\min\{k, n-k\}}$.*

In particular, the function $\max\{x_1, \ldots, x_n\}$ can be computed by a difference of two $(min, +)$ formulas with total size $n + n\lceil \log n \rceil$.

Proof. Note that $\mathsf{MaxSum}_n^k = \mathsf{Sum}_n - \mathsf{MinSum}_n^{n-k}$. Lemma 12 below shows that MinSum_n^{n-k} can be computed by a formula of size $n(\lceil \log n \rceil)^{\min\{k, n-k\}}$ for $0 \leq k \leq n$. Since Sum_n can be computed by a formula of size n, the claimed upper bound for MaxSum_n^k follows. ◀

▶ **Lemma 12.** *For all n, k such that $n \geq 1$ and $0 \leq k < n$, the functions MinSum_n^k, MinSum_n^{n-k} can be computed by a $(\min, +)$ formula of size $n(\lceil \log n \rceil)^k$.*

Hence the functions MinSum_n^k, MinSum_n^{n-k} can be computed by $(\min, +)$ formulas of size $n(\lceil \log n \rceil)^{\min\{k, n-k\}}$.

Proof. We prove the upper bound for MinSum_n^{n-k}. The bound for MinSum_n^k follows from an essentially identical argument.

We prove this by induction on k.

- Base Case: $k = 0$. For every $n \geq 1$, $\mathsf{MinSum}_n^{n-k} = \mathsf{Sum}_n$ and can be computed with size n.
- Inductive Hypothesis: For all $k' < k$, and all $n > k'$, $\mathsf{MinSum}_n^{n-k'}$ can computed in size $n(\lceil \log n \rceil)^{k'}$.
- Inductive Step: We want to prove the claim for k, where $k \geq 1$, and for all $n > k$. We proceed by induction on n.
  - Base Case: $n = k + 1$. $\mathsf{MinSum}_n^{n-k} = \mathsf{MinSum}_n^1$ is the minimum of the n variables, and can be computed in size n.
  - Inductive Hypothesis: For all $k < m < n$, MinSum_m^{m-k} can be computed in size $m(\lceil \log m \rceil)^k$.

- Inductive Step: Let $m' = \lfloor n/2 \rfloor$, $m'' = \lceil n/2 \rceil$, Let X, X_l, X_r denote the sets of variables $\{x_1, \ldots, x_n\}$, $\{x_1, \ldots, x_{m'}\}$, $\{x_{m'+1}, \ldots, x_n\}$. Note that $|X_l| = m'$, $|X_r| = m''$, $m' + m'' = n$. Let p denote $\lceil \log n \rceil$. Note that $\lceil \log m' \rceil = \lceil \log m'' \rceil = p - 1$.
To compute MinSum_n^{n-k} on X, we first compute, for various values of t, $\mathsf{MinSum}_{m'}^{m'-t}$ on X_l, $\mathsf{MinSum}_{m''}^{m''-(k-t)}$ on X_r, and add them up. We then take the minimum of these sums. Note that if $m' = t$ or $m'' = k - t$, then that summand is simply 0 and we only compute the other summand. Now $\mathsf{MinSum}_n^{n-k}(X)$ can be computed as

$$\min \left\{ \mathsf{MinSum}_{m'}^{m'-t}(X_l) + \mathsf{MinSum}_{m''}^{m''-(k-t)}(X_r) \mid \max\{0, k - m''\} \leq t \leq \min\{m', k\} \right\}$$

For all the sub-expressions appearing in the above construction, we can use inductively constructed formulas. Using the inductive hypotheses (both for $t < k$ and for $t = k$, $m'' < n$), we see that the number of leaves in the resulting formula is given by

$$\begin{aligned}
& \sum_{t=\max\{0,k-m''\}}^{\min\{m',k\}} \left[m'(p-1)^t + m''(p-1)^{k-t} \right] \\
\leq\ & \sum_{t=0}^{k} \left[m'(p-1)^t + m''(p-1)^{k-t} \right] \\
=\ & \left[\sum_{t=0}^{k} m'(p-1)^t \right] + \left[\sum_{t=0}^{k} m''(p-1)^t \right] \\
=\ & (m' + m'') \left[\sum_{t=0}^{k} (p-1)^t \right] \\
\leq\ & n \left[(p-1) + 1 \right]^k = np^k
\end{aligned}$$

◀

In the rest of this paper, our goal is to prove a matching lower bound for the max function. Note that the constructions in Theorem 11 and Lemma 12 yield formulas that do not use constants at any leaves. Intuitively, it is clear that if a formula computes the maximum correctly for all natural numbers, then constants cannot help. So the lower bound should hold even in the presence of constants, and indeed our lower bound does hold even if constants are allowed.

4 The main lower bound

In this section, we prove the following theorem:

▶ **Theorem 13.** *Let F_1, F_2 be $(min, +)$ formulas over $\mathbb{N}$ such that $F_1 - F_2 = \max(x_1, \ldots, x_n)$. Then $L(F_1) \geq n$, and $L(F_2) \geq n \log n$.*

The proof proceeds as follows: we first transform F_1 and F_2 over a series of steps to formulas G_1 and G_2 no larger than F_1 and F_2, such that $G_1 - G_2$ equals $F_1 - F_2$ and hence still computes max, and G_1 and G_2 have some nice properties. These properties immediately imply that $L(F_1) \geq L(G_1) \geq n$. We further transform G_2 to a constant-free formula H no larger than G_2. We then define a measure for functions computable by constant-free $(\min, +)$ formulas, relate this measure to formula size, and use the properties of G_2 and H to show that the function h computed by H has large measure and large formula size.

Transformation 1. For $b \in \{1,2\}$, let S_b denote the set $S(F_b)$. For $i \in [n+1]$, let A_i be the minimum value appearing in the ith coordinate in any tuple in $S_1 \cup S_2$. Let $\tilde{A}$ denote the tuple $(A_1, \ldots, A_n, A_{n+1})$. By repeatedly invoking Proposition 9, we obtain formulas G_b computing $F_b - \langle \tilde{A} \cdot \tilde{x} \rangle$, with $L(G_b) \leq L(F_b)$. For $b \in \{1,2\}$, let T_b denote the set $S(G_b)$.

We now establish the following properties of G_1 and G_2.

▶ **Lemma 14.** *Let F_1, F_2 be $(\min,+)$ formulas such that $F_1 - F_2$ computes* max. *Let G_1, G_2 be obtained as described above. Then*

1. $L(G_1) \leq L(F_1)$, $L(G_2) \leq L(F_2)$,
2. $\max(X) = F_1 - F_2 = G_1 - G_2$,
3. *For every $i \in [n]$, for every $\tilde{a} \in T_1$, $a_i > 0$. Hence $L(G_1) \geq n$.*
4. *For every $i \in [n]$, there exists $\tilde{a} \in T_2$, $a_i = 0$.*
5. *There exist $\tilde{a} \in T_1$, $\tilde{b} \in T_2$, $a_{n+1} = b_{n+1} = 0$.*
6. *For every $i, j \in [n]$ with $i \neq j$, for every $\tilde{a} \in T_2$, $a_i + a_j > 0$.*

Proof. 1. This follows from proposition 9.

2. Obvious.
3. Suppose for some $\tilde{a} \in T_1$ and for some $i \in [n]$, $a_i = 0$. Consider the input assignment $\tilde{d}$ where $d_i = 1 + a_{n+1}$ and $d_j = 0$ for $j \in [n] \setminus \{i\}$. Then $\max\{d_1, \ldots, d_n\} = 1 + a_{n+1}$. However, $\langle \tilde{a} \cdot \tilde{d} \rangle = a_{n+1}$. Therefore on input $\tilde{d}$, $G_1(\tilde{d}) \leq a_{n+1}$. Since $G_2 \geq 0$ on all assignments, we get $G_1(\tilde{d}) - G_2(\tilde{d}) \leq a_{n+1} < \max(\tilde{d})$, contradicting the assumption that $G_1 - G_2$ computes max.
4. This follows from the previous point and the choice of A_i for each i.
5. From the choice of A_{n+1}, we know that there is an $\tilde{a}$ in $T_1 \cup T_2$ with $a_{n+1} = 0$. Suppose there is such a tuple in exactly one of the sets T_1, T_2. Then exactly one of $G_1(\tilde{0})$, $G_2(\tilde{0})$ equals 0, and so $G_1 - G_2$ does not compute $\max(\tilde{0})$.
6. Suppose to the contrary, some $\tilde{a} \in T_2$ has $a_i = a_j = 0$. Consider the input assignment $\tilde{d}$ where $d_i = d_j = 1 + a_{n+1}$ and $d_k = 0$ for $k \in [n] \setminus \{i, j\}$. Then $\max\{d_1, \ldots, d_n\} = 1 + a_{n+1}$. Since every x_k figures in every tuple of T_1, $G_1(\tilde{d}) \geq d_i + d_j = 2a_{n+1} + 2$. But $G_2(\tilde{d}) \leq a_{n+1}$. Hence $G_1(\tilde{d}) - G_2(\tilde{d})$ does not compute $\max(\tilde{d})$.

◀

We have already shown above that $L(F_1) \geq L(G_1) \geq n$. Now the more tricky part: we need to lower bound $L(G_2)$.

Transformation 2. Let H' be the formula obtained by simply replacing every constant in G_2 by 0. Let H be the constant-free formula obtained from H' by eliminating the zeroes, repeatedly replacing $0 + A$ by A, $\min\{0, A\}$ by 0. Let h be the function computed by H. Then, $L_{cf}(h) \leq L(H) \leq L(H') = L(G_2) \leq L(F_2)$. It thus suffices to show that $L_{cf}(h) \geq n \log n$. To this end, we define a complexity measure μ, relate it to constant-free formula size, and show that it is large for the function h.

▶ **Definition 15.** For an n-variate function f computable by a constant-free $(\min,+)$ formula, we define

$$(f)_1 = \{i \mid f(e_i) \geq 1, f(0) = 0\}.$$
$$(f)_2 = \{(i,j) \mid f(e_i + e_j) \geq 1, f(e_i) = 0, f(e_j) = 0\}.$$

We define $G(f)$ to be the graph whose vertex set is $[n]$ and edge set is $(f)_2$.

The measure μ for function f is defined as follows:

$$\mu = \frac{|(f)_1|}{n} + H(G(f))$$

The following lemma relates $\mu(f)$ with $L(f)$. This relation has been used before, see for instance [15] for applications to monotone Boolean circuits. Since we have not seen an application in the setting of $(\min, +)$ formulas, we (re-)prove this in detail here; however, it is really the same proof.

▶ **Lemma 16.** *Let f be an n-variate function computable by a constant-free $(\min, +)$ formula. Then $L_{cf}(f) \geq n \cdot \mu(f)$.*

Proof. The proof is by induction on the depth of a witnessing formula F that computes f and has $L_{cf}(F) = L_{cf}(f)$.

Base case F is an input variable, say x_i. Then $(f)_1 = \{x_i\}$, and $G(f)$ is the empty graph, so $\mu(f) = \frac{1}{n}$. Hence $1 = L_{cf}(f) = n\mu(f)$.

Inductive step F is either $F' + F''$ or $\min\{F', F''\}$ for some formulas F', F'' computing functions f', f'' respectively. Since F is an optimal-size formula for f, F' and F'' are optimal-size formulas for f' and f'' as well. So $L_{cf}(f) = L(F) = L(F') + L(F'') = L_{cf}(f') + L_{cf}(f'')$.

Case a. $F = F' + F''$. Then $(f)_1 = (f')_1 \cup (f'')_1$ and $G(f) \subseteq G(f') \cup G(f'')$. Hence,

$$\begin{aligned}
\mu(f) &\leq \frac{|(f')_1 \cup (f'')_1|}{n} + H(G(f') \cup G(f'')) && \text{(Lemma 4)} \\
&\leq \frac{|(f')_1|}{n} + \frac{|(f'')_1|}{n} + H(G(f')) + H(G(f'')) && \text{(Lemma 4)} \\
&= \mu(f') + \mu(f'') \\
&\leq \frac{1}{n} \cdot L_{cf}(f') + \frac{1}{n} \cdot L_{cf}(f'') && \text{(Induction)} \\
&= \frac{1}{n} \cdot L_{cf}(f) && (L_{cf}(f) = L_{cf}(f') + L_{cf}(f''))
\end{aligned}$$

Case b. $F = \min(F', F'')$. Let $(f')_1 = A$ and $(f'')_1 = B$. Then $(f)_1 = A \cap B$ and $G(f) \subseteq G(f') \cup G(f'') \cup G(A \setminus B, B \setminus A)$. Here, $G(P, Q)$ denotes the bipartite graph G with parts P and Q. Hence,

$$\begin{aligned}
\mu(f) &\leq \frac{1}{n}(|A \cap B|) + H(G(f') \cup G(f'') \cup G(A \setminus B, B \setminus A)) && \text{(Lemma 4)} \\
&\leq \frac{1}{n}(|A \cap B|) + H(G(f')) + H(G(f'')) + H(G(A \setminus B, B \setminus A)) && \text{(Lemma 4)} \\
&\leq \frac{1}{n}(|A \cap B|) + H(G(f')) + H(G(f'')) + \frac{1}{n}(|A \setminus B| + |B \setminus A|) && \text{(Lemma 5)} \\
&\leq \frac{1}{n}(|A| + |B|) + H(G(f')) + H(G(f'')) \\
&= \mu(f') + \mu(f'') \\
&\leq \frac{1}{n} \cdot L_{cf}(f') + \frac{1}{n} \cdot L_{cf}(f'') && \text{(Induction)} \\
&= \frac{1}{n} \cdot L_{cf}(f) && (L_{cf}(f) = L_{cf}(f') + L_{cf}(f''))
\end{aligned}$$

Hence, $\mu(f) \leq \frac{1}{n} \cdot L_{cf}(f)$. ◀

Using this measure, we can now show the required lower bound.

▶ **Lemma 17.** *For the function h obtained after Transformation 2, $\mu(h) \geq \log n$.*

Proof. Recall that we replaced constants in G_2 by 0 to get H', then eliminated the 0s to get constant-free H computing h. By Proposition 8, we know that $S(H') = \{\tilde{b} \mid b_{n+1} = 0, \exists \tilde{a} \in T_2, a_i = b_i \forall i \in [n]\}$ and that $h = \min\{\tilde{x} \cdot \tilde{b} \mid \tilde{b} \in S(H')\}$.

From item 4 in Lemma 14, it follows that $(h)_1 = \emptyset$. (For every i, there is a $\tilde{b} \in S(H')$ with $b_i = 0$. So $h(e_i) \leq \langle e_i \cdot \tilde{b} \rangle = 0$.)

Since $(h)_1$ is empty, $(i,j) \in G(h)$ exactly when $h(e_i + e_j) \geq 1$. From item 6 in Lemma 14, it follows that every pair (i,j) is in $G(h)$. Thus $G(h)$ is the complete graph K_n.

From Lemma 5 we conclude that $\mu(h) = \log n$. ◀

Lemmas 16 and 17 imply that $L_{cf}(h) \geq n \log n$. Since $L_{cf}(h) \leq L(H) \leq L(H') = L(G_2) \leq L(F_2)$, we conclude that $L(F_2) \geq n \log n$.

This completes the proof of Theorem 13.

A major ingredient in this proof is using the measure μ. This yields lower bounds for constant-free formulas. For functions computable in a constant-free manner, it is hard to see how constants can help. However, to transfer a lower bound on $L_{cf}(f)$ to a lower bound on $L(f)$, this idea of "constants cannot help" needs to be formalized. The transformations described before we define μ do precisely this.

For the MinSum_n^{n-1} function, applying the measure technique immediately yields the lower bound $L_{cf}(\mathsf{MinSum}_n^{n-1}) \geq n \log n$. Transferring this lower bound to formulas with constants is a corollary of our main result, and with it we see that the upper bound from Lemma 12 is tight for MinSum_n^{n-1}.

▶ **Corollary 18.** *Any* $(\min, +)$ *formula computing* MinSum_n^{n-1} *must have size at least* $n \log n$.

Proof. Let F be any formula computing MinSum_n^{n-1}. Applying Theorem 13 to $F_1 = x_1 + \ldots + x_n$ and $F_2 = F$, we obtain $L(F) \geq n \log n$. ◀

5 Discussion

Our results hold when variables take values from $\mathbb{N}$. In the standard $(\min, +)$ semi-ring, the value ∞ is also allowed, since it serves as the identity for the min operation. The proof of our main result Theorem 13 does not carry over to this setting. The main stumbling block is the removal of the "common" part of $S(F)$. However, if we allow ∞ as a value that a variable can take, but not as a constant appearing at a leaf, then the lower bound proof still seems to work. However, the upper bound no longer works; while taking a difference, what is $\infty - \infty$?

Apart from the many natural settings where the tropical semiring $(\min, +, \mathbb{N} \cup \{\infty\}, 0, \infty)$ crops up, it is also interesting because it can simulate the Boolean semiring for monotone computation. The mapping is straightforward: $0, 1, \vee, \wedge$ in the Boolean semiring are replaced by $\infty, 0, \min, +$ respectively in the tropical semiring. Proving lower bounds for $(\min, +)$ formulas could be easier than for monotone Boolean formulas because the $(\min, +)$ formula has to compute a function correctly at all values, not just at $0, \infty$. Hence it would be interesting to extend our lower bound to this setting with ∞ as well.

Our transformations crucially use the fact that there is a minimum element, 0. Thus, we do not see how to extend these results to computations over integers. It appears that we will need to include $-\infty$, and since we are currently unable to handle even $+\infty$, there is already a barrier.

The lower bound method uses graph entropy which is always bounded above by $\log n$. Thus this method cannot give a lower bound larger than $n \log n$. It would be interesting to obtain a modified technique that can show that all the upper bounds in Theorem 11 and

Lemma 12 are tight. It would also be interesting to find a direct combinatorial proof of our lower bound result, without using graph entropy.

References

1. Eric Allender. Arithmetic circuits and counting complexity classes. In Jan Krajicek, editor, *Complexity of Computations and Proofs*, Quaderni di Matematica Vol. 13, pages 33–72. Seconda Universita di Napoli, 2004. An earlier version appeared in the Complexity Theory Column, SIGACT News 28, 4 (Dec. 1997) pp. 2-15.
2. Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1956.
3. Imre Csiszár, János Körner, László Lovász, Katalin Marton, and Gábor Simonyi. Entropy splitting for antiblocking corners and perfect graphs. *Combinatorica*, 10(1):27–40, 1990.
4. Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.
5. Lester R Ford Jr. Network flow theory. Technical Report P-923, Rand Corporation, 1956.
6. Michael Held and Richard M Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.
7. Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *Journal of the ACM (JACM)*, 29(3):874–897, 1982.
8. Stasys Jukna. Lower bounds for tropical circuits and dynamic programs. *Theory of Computing Systems*, 57(1):160–194, 2015.
9. Stasys Jukna. Tropical complexity, Sidon sets, and dynamic programming. *SIAM Journal on Discrete Mathematics*, 30(4):2064–2085, 2016.
10. Stasys Jukna and Georg Schnitger. On the optimality of Bellman–Ford–Moore shortest path algorithm. *Theoretical Computer Science*, 628:101–109, 2016.
11. János Körner. Coding of an information source having ambiguous alphabet and the entropy of graphs. In *Transactions of 6th Prague Conference on Information Theory*, pages 411–425. Academia, Prague, 1973.
12. János Körner. Fredman-Komlós bounds and information theory. *SIAM. J. on Algebraic and Discrete Methods*, 7(4):560–570, 1986.
13. János Körner and Katalin Marton. New bounds for perfect hashing via information theory. *European Journal of Combinatorics*, 9(6):523–530, 1988.
14. Edward F Moore. *The shortest path through a maze*. Bell Telephone System., 1959.
15. Ilan Newman and Avi Wigderson. Lower bounds on formula size of boolean functions using hypergraph entropy. *SIAM Journal on Discrete Mathematics*, 8(4):536–542, 1995.
16. Gábor Simonyi. Graph entropy: A survey. *Combinatorial Optimization*, 20:399–441, 1995.
17. Gábor Simonyi. Perfect graphs and graph entropy: An updated survey. In *Perfect Graphs*, pages 293–328. John Wiley and Sons, 2001.
18. Stephen Warshall. A theorem on Boolean matrices. *Journal of the ACM (JACM)*, 9(1):11–12, 1962.

Selecting Nodes and Buying Links to Maximize the Information Diffusion in a Network

Gianlorenzo D'Angelo[1], Lorenzo Severini[2], and Yllka Velaj[3]

1 **Gran Sasso Science Institute (GSSI), L'Aquila, Italy**
`gianlorenzo.dangelo@gssi.infn.it`
2 **ISI Foundation, Torino, Italy**
`lorenzo.severini@isi.it`
3 **Gran Sasso Science Institute (GSSI), L'Aquila, Italy, and**
Department of Economic Studies, University of Chieti-Pescara, Pescara, Italy
`yllka.velaj@unich.it`

Abstract

The Independent Cascade Model (ICM) is a widely studied model that aims to capture the dynamics of the information diffusion in social networks and in general complex networks. In this model, we can distinguish between active nodes which spread the information and inactive ones. The process starts from a set of initially active nodes called seeds. Recursively, currently active nodes can activate their neighbours according to a probability distribution on the set of edges. After a certain number of these recursive cycles, a large number of nodes might become active. The process terminates when no further node gets activated.

Starting from the work of Domingos and Richardson [10, 26], several studies have been conducted with the aim of shaping a given diffusion process so as to maximize the number of activated nodes at the end of the process. One of the most studied problems has been formalized by Kempe et al. and consists in finding a set of initial seeds that maximizes the expected number of active nodes under a budget constraint [14]. In this paper we study a generalization of the problem of Kempe et al. in which we are allowed to spend part of the budget to create new edges incident to the seeds. That is, the budget can be spent to buy seeds or edges according to a cost function. The problem does not admin a PTAS, unless $P = NP$. We propose two approximation algorithms: the former one gives an approximation ratio that depends on the edge costs and increases when these costs are high; the latter algorithm gives a constant approximation guarantee which is greater than that of the first algorithm when the edge costs can be small.

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

Keywords and phrases Approximation algorithms, information diffusion, complex networks, independent cascade model, network augmentation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.75

1 Introduction

When a new idea or innovation arises in a network of individuals, it can either quickly propagate to a large part of the network and be adopted by many individuals or immediately expire. Understanding the dynamics that regulate these behaviours has been one of the main goals in the field of complex network analysis and has been studied under the name of *influence spreading* or *information diffusion* analysis problem [10, 14]. The motivating application span several fields: from marketing with the aim of evaluating the success of a new product or maximizing its adoption [3, 4, 11, 21, 26], to epidemiology in order to limit the diffusion of a virus or disease [22, 23], the study of adoption of innovations [5, 27, 30],

the analysis of social networks to find influential users and to study how information flows through the network [2], and the analysis of cascading failures in power networks [1].

Different models of information diffusion have been introduced in the literature. Two widely studied models are the *Linear Threshold Model* (LTM) [13, 15, 28] and the *Independent Cascade Model* (ICM) [11, 12, 14, 15]. In both models, we can distinguish between *active*, or *affected*, nodes which spread the information and inactive ones. At the beginning of the process a small percentage of nodes of the graph is set to active in order to let the information diffusion process start. These nodes are called *seeds*. Recursively, currently affected nodes can infect their neighbours with some probability. After a certain number of these cascading cycles, a large number of nodes might become affected in the network. In LTM the idea is that a node becomes active as more of its neighbours become active. Formally, each node u has a threshold t_u chosen uniformly at random in the interval $[0, 1]$. The threshold represents the weighted fraction of neighbours of u that must become active in order for u to become active. During the process, a node u becomes active if the total weight of its active neighbours is greater than t_u. In ICM, instead, an active node u tries to influence one of its inactive neighbours but the success of node u in activating the node v only depends on the propagation probability of the edge from u to v (each edge has its own value). Regardless of its success, the same node will never get another chance to activate the same inactive neighbour. The process terminates when no further node gets activated.

An interesting question, in the analysis of the information diffusion through a network, is how to shape a given diffusion process so as to maximize or minimize the number of activated nodes at the end of the process by taking intervention actions. Many intervention actions have been studied in the literature, the most important one has been proposed by Domingos and Richardson in the field of viral marketing and asks to find a small set of "influential" seeds in a network in order to activate a large part of the network [10, 26]. The problem has been formalized by Kempe et al. [15] as follows: if we are allowed to choose at most k seeds, which ones should be selected so as to maximize the number of active nodes resulting from the diffusion process [15]. This problem admits a $(1 - \frac{1}{e})$-approximation algorithm and this factor cannot be improved, unless $P = NP$ [15]. Besides seeds selection, other intervention actions may be used to facilitate the diffusion processes, such as inserting or deleting edges and adding or deleting nodes in the network. Since in social networks and in other complex networks users can add edges incident to themselves, in this paper we consider the possibility to create a limited number of new edges incident to the initial seed nodes. In detail, we study the following generalization of the problem of Kempe et al.: we are given a cost for each possible edge that can be created in a network and a budget k, and we want to find a set of seed nodes A and a set of edges S incident to the nodes in A such that the expected number of active nodes at the end of the diffusion process is maximized and the overall cost of A and S does not exceed k, assuming that all the seeds have the same cost.

Related work. The problem of choosing initially active nodes to maximize the information diffusion in a network has been widely studied, we refer the interested reader to [15] and references therein for more detail, while the budgeted version of the problem was proposed in [24]. In the following we focus on the problem of modifying a graph in order to maximize or minimize the spread of information in a network under LTM and ICM models.

To the best of our knowledge, under LTM, the problems that have been studied are those outlined in what follows. Khalil et al. [16] consider two types of actions, adding edges to or deleting edges from the existing network to minimize the information diffusion and they show that this network structure modification problem has a supermodular objective and therefore

can be solved by algorithms with provable approximation guarantees. Zhang et al. [32] consider arbitrarily specified sets of nodes, and interventions that involve both edge and node removal from the sets. They develop algorithms with rigorous performance guarantees and good empirical performance. Kimura et al. [18] use a greedy approach to delete edges under the LTM but do not provide any rigorous approximation guarantees. Kuhlman et al. [20] propose heuristic algorithms for edge removal under a simpler deterministic variant of LTM which is not only hard, but also has no approximation guarantee. Papagelis [25] and Crescenzi et al. [6] study the problem of augmenting the graph in order to increase the connectivity or the centrality of a node, respectively and experimentally show that this increases the average number of eventual active nodes.

Under ICM, Wu et al. [31] consider intervention actions other than edge addition, edge deletion and seed selection, such as increasing the probability that a node infects its neighbours. They proved that optimizing the selection of these actions with a limited budget is NP-hard and is neither submodular nor supermodular. Sheldon et al. [29] study the problem of node addition to maximize the spread of information, and provide a counterexample showing that the objective function is not submodular. Kimura et al. [19] propose methods for efficiently finding good approximate solutions on the basis of a greedy strategy for the edge deletion problem under the ICM, but do not provide any approximation guarantees. D'Angelo et al. [7, 8] focus on the case in which the initial set of seeds is given and provide a constant approximation algorithm.

Our results. In this paper, we focus on the independent cascade model and investigate the problem of selecting a set of initial seeds and adding a small number of edges incident to the seeds, without exceeding a given budget k, in order to maximize the spread of information in terms of the expected number of nodes that eventually become active. The problem we analyse differs from above mentioned ones since we make the reasonable restriction that the edges to be added can only be incident to the seed nodes. To our knowledge, similar problems have never been studied for the independent cascade model. We refer to this problem as the *Budgeted Influence Maximization with Augmentation problem* (BIMA).

We observe that the BIMA problem is a generalization of the problem in [15] and therefore cannot be approximated within a factor greater than $1 - \frac{1}{e}$, unless $P = NP$.

We then focus on approximation algorithms. We first assume that the edge costs are all greater that a given constant c_{min} and, in Section 3, we propose two approximation algorithms that guarantee approximation factors of $1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}$ and $\left(1 - \frac{1}{e}\right) c_{\min}$, respectively. Note that these factors increase with $c_{\min}$ and the second algorithm achieves the optimal approximation of $1 - \frac{1}{e}$ when all the edge costs are equal to 1. It is somewhat expected that when the all costs are high the approximation factor gets closer to the lower bound of $1 - \frac{1}{e}$. Indeed, when the cost of buying an edge approaches the cost of buying the node at the tail of this edge, we can buy the node instead of the edge with small increase in the cost. This allows us to exploit the $1 - \frac{1}{e}$ approximation algorithm for the seed selection problem proposed in [15]. However, the challenge of our problem consists in finding a good approximation even when the cost function includes small values. In Section 4, we focus on the general case and propose an algorithm that guarantees a constant approximation ratio of about 0.0878. This algorithm outperforms the other two algorithms when the cost is allowed to be small. See Figure 1 for a summary of the approximation ratio of our algorithms.

Some of the proofs are omitted due to space constraints and can be found in the full version of the paper [9].

2 Preliminaries

A social network is represented by a weighted directed graph $G = (V, E, p, c)$, where V represents the set of nodes, E represents the set of relationships, $p : V \times V \to [0, 1]$ is the propagation probability of an edge, that is the probability that the information is propagated from u to v if $(u, v) \in E$, and $c : V \times V \to [0, 1]$ is the cost of adding an edge to E. In ICM, each node can be either *active* or *inactive*. If a node is active (or adopter of the innovation), then it is already influenced by the information under diffusion. If a node is inactive, then it is unaware of the information or not influenced. The process runs in discrete steps. At the beginning of the ICM process, few nodes are given the information, they are known as *seed nodes*. Upon receiving the information these nodes become active. In each discrete step, an active node tries to influence its inactive neighbours. The success of node u in activating the node v depends on the propagation probability of the edge (u, v), independently of the history so far. Regardless of its success, the same node will never get another chance to activate the same inactive neighbour. The process terminates when no further nodes become activated from inactive state.

We define the influence of a set $A \subseteq V$ in the graph G, denoted $\sigma(A)$, to be the expected number of active nodes in G at the end of the process, given that A is the set of seeds. Given a set S of edges not in E, we denote by $G(S)$ the graph augmented by adding the edges in S to G, i.e. $G(S) = (V, E \cup S)$. We denote by $\sigma(A, S)$ the influence of A in $G(S)$.

In this paper we look for a set of seeds A and a set of edges S, to be added to G, incident to these seeds that maximize $\sigma(A, S)$. W.l.o.g. we assume that each seed node can be selected with cost 1, while each edge $e \in (V \times V) \setminus E$ can be selected with cost $c_e \in [0, 1]$. In detail, the BIMA problem is defined as follows: given a graph $G = (V, E)$ and a budget k, find a set A of seeds and a set S of edges such that $S \subseteq (A \times V) \setminus E$, $c(A, S) \leq k$, and $\sigma(A, S)$ is maximum, where $c(A, S) = |A| + \sum_{e \in S} c_e$.

A *live-edge graph* $X = (V, E_X)$ of G is a directed graph where the set of nodes is the same set V and the set of edges E_X is a subset of E given by an edge selection process in which each edge in E belongs to E_X or not according to its propagation probability. In detail, we can assume that for each edge $e = (u, v)$ in the graph, we flip a coin of bias p_e and only the edges for which the coin indicated an activation belong to E_X. It is easy to show that the information diffusion process is equivalent to a reachability problem in live-edge graphs: given any seed set A, the distribution of active node sets after the diffusion process ends is the same as the distribution of node sets reachable from A in live-edge graphs. We denote by χ the probability space in which each sample point specifies one possible set of outcomes for all the coin flips on the edges, that is the set of all possible live-edge graphs of G. For a set of edges $S \subseteq (V \times V) \setminus E$, the set of all possible live-edge graphs of $G(S)$ is denoted by $\chi(S)$. Given two set of edges S, T, such that $S \subseteq T$, for each live-edge graph X in $\chi(S)$ we denote by $\chi(T, X)$ the set of live-edge graphs in $\chi(T)$ that have X as a subgraphs and possibly contain other edges in $T \setminus S$. In other words, a live-edge graphs in $\chi(T, X)$ has been generated with the same outcomes as X on the coin flips in the edges of $E \cup S$ and it has other outcomes for edges in $T \setminus S$. The following holds: $|\chi(T, X)| = 2^{|T \setminus S|}$, for each $Y \in \chi(T, X)$ $\mathbb{P}[Y] = \mathbb{P}[Y|X]\mathbb{P}[X]$, $\mathbb{P}[X] = \sum_{Y \in \chi(T,X)} \mathbb{P}[Y]$, and $\sum_{Y \in \chi(T,X)} \mathbb{P}[Y|X] = 1$. For a node $a \in V$ and a live-edge graph X in $\chi(S)$, let $R(a, X)$ be the set of all nodes that can be reached from a in graph X, that is for each node $u \in R(a, X)$, there exists a path from a to u consisting entirely of live edges with respect to the outcome of the coin flips that generates X. Let $R(A, X) = \bigcup_{a \in A} R(a, X)$, then $\sigma(A, S)$ can be computed as $\sigma(A, S) = \sum_{X \in \chi(S)} \mathbb{P}[X] \cdot |R(A, X)|$.

Computing $\sigma(A)$ is $\#P$-complete [4], however it has been proven by using the Chernoff bound that it can be approximated within an arbitrarily good factor by simulating the random process a polynomial number of times [15]. Therefore, in the rest of the paper we can assume that we can compute $\sigma(A)$ (and $\sigma(A, S)$) within an arbitrary bound. This reflects to an additional factor $1 + \epsilon$, for any $\epsilon > 0$, to all algorithms presented in this paper. For the sake of clarity, we omit this factor from the approximation factor of our algorithms.

Given a set of edges S, for each graph $X \in \chi(S)$ and subset of edges $T \subseteq S$, we denote by X^T the graph obtained by removing edges in T from X. Given two feasible solutions (A_1, S_1) and (A_2, S_2), such that $A_2 \subseteq A_1$ and $S_2 \subseteq S_1$, we denote with $\delta(A_1, S_1, A_2, S_2)$ the expected number of nodes affected by (A_1, S_1) and not affected by (A_2, S_2), formally: $\delta(A_1, S_1, A_2, S_2) = \sum_{X \in \chi(S_1)} \mathbb{P}[X] \cdot \left(|R(A_1, X)| - |R(A_2, X^T)|\right)$, where $T = S_1 \setminus S_2$.

▶ **Proposition 1.** *For each $A_2 \subseteq A_1 \subseteq V$ and $S_2 \subseteq S_1 \subseteq V \times V$, such that the edges in S_1 and S_2 are outgoing A_1 and A_2, respectivey, then $\delta(A_1, S_1, A_2, S_2) = \sigma(A_1, S_1) - \sigma(A_2, S_2)$.*

We observe that our problem is a generalization of the influence maximization problem in [15], indeed it is enough to set $c_e = 1$ for each $e \in (V \times V) \setminus E$. It follows that BIMA cannot be approximated within a factor greater than $1 - \frac{1}{e}$, unless $P = NP$.

3 Lower-bounded edge costs

In this section we consider the case in which the edge costs are at least a given value $c_{\min}$, that is for each $e \in V \times V, c_e \geq c_{\min}$. It can be easily shown that in this case selecting a set A of k seed nodes that maximizes $\sigma(A, \emptyset)$ guarantees an approximation factor of $c_{\min}$. Since this problem can be optimally approximated within $1 - \frac{1}{e}$ [15], we can obtain an overall $\left(1 - \frac{1}{e}\right) c_{\min}$ approximation. In what follows we give an approximation algorithm that improves over this bound for small values of $c_{\min}$. The following analysis serves also as a warm-up for the analysis of the algorithm proposed in the next section.

Our algorithm, whose pseudocode is reported in Algorithm 1, finds two candidate solutions: the first solution is obtained by a greedy algorithm at lines 2–25, the second solution is found at line 26 and is made of a single node a_M and a single edge (a_M, v_M) for which $\sigma(\{a_M\}, \{(a_M, v_M)\})$ is maximized. Then, the algorithm outputs one of the candidate solutions that maximizes the expected number of affected nodes (line 27).

The greedy phase, at each iteration, selects a solution (A, S) that adds at most one node and one edge to the current solution (A', S') and that maximizes the ratio between $\delta(A, S, A', S')$ and the marginal cost of (A, S), that is the cost of the added node or edge. In particular, it considers three possible ways of obtaining (A, S) from (A', S'):

line 3: select a seed node a that maximizes $r_1 = \delta(A' \cup \{a\}, S', A', S')$, $(A, S) = (A' \cup \{a\}, S')$;

line 4: select an edge (a, v) incident to a seed a in A' that maximizes $r_2 = \frac{\delta(A', S' \cup \{(a,v)\}, A', S')}{c_{(a,v)}}$, $(A, S) = (A', S' \cup \{(a, v)\})$;

line 5: select a seed node a not in A' and edge (a, v) incident to a that maximize $r_3 = \frac{\delta(A' \cup \{a\}, S' \cup \{(a,v)\}, A', S')}{1 + c_{(a,v)}}$, $(A, S) = (A' \cup \{a\}, S' \cup \{(a, v)\})$.

The greedy phase of the algorithm selects a solution (A, S) that maximizes the three above ratios. If (A, S) does not violate the budget, i.e. cost $c(A, S)$ is at most k, then it is chosen as new solution.

We denote by (A^*, S^*) an optimal solution to the BIMA problem. Let us consider the iterations executed by the greedy algorithm in which an element is added to (A, S). For $i \geq 1$, let us denote by j_i the index of these iterations, $j_i < j_{i+1}$, and let j_{l+1} be the index of the first iteration in which an element in (A^*, S^*) is considered (i.e. it maximizes the

Algorithm 1:
Input : A directed graph $G = (V, E)$ and an integer $k \in \mathbb{N}$
Output : A set of nodes A and a of edges $S \subseteq (A \times V) \setminus E$ such that $c(A, S) \leq k$

$A := \emptyset$; $S := \emptyset$; $U := V$; $T := (V \times V) \setminus E$;
while $T \neq \emptyset$ *or* $U \neq \emptyset$ **do**
  $r_1 = \max_{a \in U} \{\delta(A \cup \{a\}, S, A, S)\}$;
  $r_2 = \max_{(a,v) \in (A \times V) \cap T} \left\{\delta(A, S \cup \{(a, v)\}, A, S)/c_{(a,v)}\right\}$;
  $r_3 = \max_{a \in U, (a,v) \in (\{a\} \times V) \cap T} \left\{\delta(A \cup \{a\}, S \cup \{(a, v)\}, A, S)/(1 + c_{(a,v)})\right\}$;
  if $\max\{r_1, r_2, r_3\} = r_1$ **then**
    $\hat{a} = \arg\max_{a \in U} \{\delta(A \cup \{a\}, S, A, S)\}$;
    if $k - 1 \geq 0$ **then**
      $A := A \cup \{\hat{a}\}$;
      $k := k - 1$;
    $U := U \setminus \{\hat{a}\}$;
  else
    if $\max\{r_1, r_2, r_3\} = r_2$ **then**
      $(\hat{a}, \hat{v}) := \arg\max_{(a,v) \in A \times V \cap T} \left\{\delta(A, S \cup \{(a, v)\}, A, S)/c_{(a,v)}\right\}$;
      if $k - c_{(\hat{a},\hat{v})} \geq 0$ **then**
        $S := S \cup \{(\hat{a}, \hat{v})\}$;
        $k := k - c_{(\hat{a},\hat{v})}$;
      $T := T \setminus \{(\hat{a}, \hat{v})\}$;
    else
      $(\hat{a}, (\hat{a}, \hat{v})) := \arg\max_{a \in U, (a,v) \in \{a\} \times V \cap T} \left\{\delta(A \cup \{a\}, S \cup \{(a, v)\}, A, S)/(1 + c_{(a,v)})\right\}$;
      if $k - c_{(\hat{a},\hat{v})} - 1 \geq 0$ **then**
        $(A, S) := (A \cup \{\hat{a}\}, S \cup \{(\hat{a}, \hat{v})\})$;
        $U := U \setminus \{\hat{a}\}$;
        $k := k - 1 - c_{(\hat{a},\hat{v})}$;
      $T := T \setminus \{(\hat{a}, \hat{v})\}$;
$(a_M, (a_M, v_M)) := \arg\max_{a \in V, (a,v) \in (\{a\} \times V) \setminus E} \{\sigma(A \cup \{a\}, S \cup \{(a, v)\})\}$;
return $\arg\max\{\sigma(A, S), \sigma(\{a_M\}, \{(a_M, v_M)\})\}$;

above ratios) but not added to (A, S) because it violates the budget constraint. We denote by (A_i, S_i) the solution at the end of iteration j_i and by $\bar{c}_i$ the marginal cost of (A_i, S_i) as computed in the above three ratios,

$$\bar{c}_i = \begin{cases} 1 & \text{if } A_i \setminus A_{i-1} = \{a\} \text{ and } S_i = S_{i-1} \quad (\text{i.e. } \max\{r_1, r_2, r_3\} = r_1) \\ c_e & \text{if } A_i = A_{i-1} \text{ and } S_i \setminus S_{i-1} = \{(a, v)\} \quad (\text{i.e. } \max\{r_1, r_2, r_3\} = r_2) \\ 1 + c_e & \text{if } A \setminus A_{i-1} = \{a\} \text{ and } S_i \setminus S_{i-1} = \{(a, v)\} \quad (\text{i.e. } \max\{r_1, r_2, r_3\} = r_3). \end{cases}$$

The next two lemmas are the core of our analysis [17].

▶ **Lemma 2.** *After each iteration* j_i, $i = 1, 2, \ldots, l + 1$, $\sigma(A_i, S_i) - \sigma(A_{i-1}, S_{i-1}) \geq \frac{\bar{c}_i}{k} \frac{c_{\min}}{1+c_{\min}} (\sigma(A^*, S^*) - \sigma(A_{i-1}, S_{i-1}))$.

Proof. We denote by δ_i the expected number of nodes affected by solution (A_i, S_i) and not affected by solution (A_{i-1}, S_{i-1}), $\delta_i = \delta(A_i, S_i, A_{i-1}, S_{i-1})$. We first show that the value $\sigma(A^*, S^*) - \sigma(A_{i-1}, S_{i-1})$ is at most the sum, for each element in (A^*, S^*) and not in (A_{i-1}, S_{i-1}), of the expected number of nodes affected by this element and not affected by solution (A_{i-1}, S_{i-1}), that is the following inequality holds:

$$\sigma(A^*, S^*) - \sigma(A_{i-1}, S_{i-1}) \leq \sum_{a \in A_1^*} \delta(A_{i-1} \cup \{a\}, S_{i-1}, A_{i-1}, S_{i-1}) + \tag{1}$$

$$\sum_{\substack{e=(a,v) \in S^* \setminus S_{i-1} \\ \text{s.t. } a \in A_{i-1}}} \delta(A_{i-1}, S_{i-1} \cup \{e\}, A_{i-1}, S_{i-1}) + \sum_{\substack{a \in A_2^* \\ e=(a,v) \in S^* \setminus S_{i-1}}} \delta(A_{i-1} \cup \{a\}, S_{i-1} \cup \{e\}, A_{i-1}, S_{i-1}),$$

where we divided the set $A^* \setminus A_{i-1}$ into two subsets: A_1^* is the subset of $A^* \setminus A_{i-1}$ that contains the seeds a with no incident edges in S^* (i.e. $\nexists (a,v) \in S^*$), and $A_2^* = A^* \setminus (A_{i-1} \cup A_1^*)$.

The difference $\sigma(A^*, S^*) - \sigma(A_{i-1}, S_{i-1})$ is at most $\sigma(A^* \cup A_{i-1}, S^* \cup S_{i-1}) - \sigma(A_{i-1}, S_{i-1})$ and therefore upper-bounded by:

$$\begin{aligned}
&\sum_{X \in \chi(S^* \cup S_{i-1})} \mathbb{P}[X] |R(A^* \cup A_{i-1}, X)| - \sum_{X \in \chi(S_{i-1})} \mathbb{P}[X] |R(A_{i-1}, X)| \\
&= \sum_{X \in \chi(S_{i-1})} \sum_{Y \in \chi(S^* \cup S_{i-1}, X)} \mathbb{P}[Y] |R(A^* \cup A_{i-1}, Y)| - \sum_{X \in \chi(S_{i-1})} \mathbb{P}[X] |R(A_{i-1}, X)| \\
&= \sum_{X \in \chi(S_{i-1})} \mathbb{P}[X] \sum_{Y \in \chi(S^* \cup S_{i-1}, X)} \mathbb{P}[Y|X] |R(A^* \cup A_{i-1}, Y)| - \sum_{X \in \chi(S_{i-1})} \mathbb{P}[X] |R(A_{i-1}, X)| \\
&= \sum_{X \in \chi(S_{i-1})} \mathbb{P}[X] \left[\sum_{Y \in \chi(S^* \cup S_{i-1}, X)} \mathbb{P}[Y|X] |R(A^* \cup A_{i-1}, Y)| - |R(A_{i-1}, X)| \right] \\
&= \sum_{X \in \chi(S_{i-1})} \mathbb{P}[X] \left[\sum_{Y \in \chi(S^* \cup S_{i-1}, X)} \mathbb{P}[Y|X] |R(A^* \cup A_{i-1}, Y)| - \sum_{Y \in \chi(S^* \cup S_{i-1}, X)} \mathbb{P}[Y|X] |R(A_{i-1}, X)| \right] \\
&= \sum_{X \in \chi(S_{i-1})} \mathbb{P}[X] \sum_{Y \in \chi(S^* \cup S_{i-1}, X)} \mathbb{P}[Y|X] \left(|R(A^* \cup A_{i-1}, Y)| - |R(A_{i-1}, X)| \right) \qquad (2)
\end{aligned}$$

For each $X \in \chi(S_{i-1})$ and $Y \in \chi(S^* \cup S_{i-1}, X)$, the difference $|R(A^* \cup A_{i-1}, Y)| - |R(A_{i-1}, X)|$ between the nodes reachable from $A^* \cup A_{i-1}$ in Y and those reachable from A_{i-1} in X can be bounded as follows:

$$|R(A^* \cup A_{i-1}, Y)| - |R(A_{i-1}, X)| \leq \sum_{a \in A_1^*} \left(|R(A_{i-1} \cup \{a\}, X)| - |R(A_{i-1}, X)| \right) + \qquad (3)$$

$$\sum_{\substack{e=(a,v) \in Y \setminus X \\ \text{s.t. } a \in A_{i-1}}} \left(|R(A_{i-1}, X \cup \{e\})| - |R(A_{i-1}, X)| \right) + \sum_{\substack{a \in A_2^* \\ e=(a,v) \in Y \setminus X}} \left(|R(A_{i-1} \cup \{a\}, X \cup \{e\})| - |R(A_{i-1}, X)| \right).$$

Combining (2) and the first term of (3), we obtain the first term of (1). To show the second and third term of (1), observe that for a function $f : V \times V \to \mathbb{N}$ and for each $X \in \chi(S_{i-1})$,

$$\sum_{Y \in \chi(S^* \cup S_{i-1}, X)} \mathbb{P}[Y|X] \sum_{e \in Y \setminus X} f(e) \leq \sum_{e \in S^* \setminus S_{i-1}} p_e \sum_{Y \in \chi(S^* \cup S_{i-1} \setminus \{e\}, X)} \mathbb{P}[Y|X \cup \{e\}] f(e) = \sum_{e \in S^* \setminus S_{i-1}} p_e f(e).$$

This shows inequality (1).

Since the greedy phase of the algorithm selects a solution that that maximizes the ratio between the marginal increment in the objective function and the cost, the following holds:

- For each $a \in A_1^*$, $\delta(A_{i-1} \cup \{a\}, S_{i-1}, A_{i-1}, S_{i-1}) \leq \frac{\delta_i}{\bar{c}_i}$;
- For each $e = (a,v) \in S^* \setminus S_{i-1}$ such that $a \in A_{i-1}$, $\frac{\delta(A_{i-1}, S_{i-1} \cup \{e\}, A_{i-1}, S_{i-1})}{c_e} \leq \frac{\delta_i}{\bar{c}_i}$;
- For each $a \in A_2^*$ and $e = (a,v) \in S^* \setminus S_{i-1}$, $\frac{\delta(A_{i-1} \cup \{a\}, S_{i-1} \cup \{e\}, A_{i-1}, S_{i-1})}{1 + c_e} \leq \frac{\delta_i}{\bar{c}_i}$.

Since the edge costs are at least $c_{\min}$, then the number of edges in $S^* \setminus S_{i-1}$ incident to each $a \in A_2^*$ is at most $\left\lfloor \frac{k}{c_{\min}} \right\rfloor \leq \frac{k}{c_{\min}}$. Therefore, the right hand side of (1) is at most:

$$\begin{aligned}
&\sum_{a \in A_1^*} \frac{\delta_i}{\bar{c}_i} + \sum_{\substack{e=(a,v) \in S^* \setminus S_{i-1} \\ \text{s.t. } a \in A_{i-1}}} \frac{\delta_i}{\bar{c}_i} c_e + \sum_{\substack{a \in A_2^* \\ e=(a,v) \in S^* \setminus S_{i-1}}} \frac{\delta_i}{\bar{c}_i} (1 + c_e) \\
&\leq \frac{\delta_i}{\bar{c}_i} \left(|A_1^*| + \sum_{\substack{e=(a,v) \in S^* \setminus S_{i-1} \\ \text{s.t. } a \in A_{i-1}}} c_e + \frac{k}{c_{\min}} + \sum_{\substack{a \in A_2^* \\ e=(a,v) \in S^* \setminus S_{i-1}}} c_e \right) \leq \left(1 + \frac{1}{c_{\min}} \right) k \frac{\delta_i}{\bar{c}_i}.
\end{aligned}$$

Where the last inequality is due to $|A_1^*| + \sum_{\substack{e=(a,v)\in S^*\setminus S_{i-1} \\ \text{s.t. } a\in A_{i-1}}} c_e + \sum_{\substack{a\in A_2^* \\ e=(a,v)\in S^*\setminus S_{i-1}}} c_e \leq k$.

To conclude the proof, $\delta_i = \sigma(A_i, S_i) - \sigma(A_{i-1}, S_{i-1})$ follows from Proposition 1. ◀

The next lemma can be proven by induction on iterations j_i and by using Lemma 2.

▶ **Lemma 3.** *After each iteration j_i, $i = 1, 2, \ldots, l+1$,*

$$\sigma(A_i, S_i) \geq \left[1 - \prod_{\ell=1}^{i} \left(1 - \frac{\bar{c}_\ell}{k} \frac{c_{\min}}{(1 + c_{\min})}\right)\right] \sigma(A^*, S^*).$$

▶ **Theorem 4.** *Algorithm 1 achieves an approximation factor of $\frac{1}{2}\left(1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}\right)\sigma(A^*, S^*)$.*

Proof. We observe that since (A_{l+1}, S_{l+1}) violates the budget, then $c(A_{l+1}, S_{l+1}) > k$. Moreover, for a sequence of numbers $a_1, a_2, \ldots, a_n$ such that $\sum_{\ell=1}^{n} a_\ell = B$, the function $\left[1 - \prod_{i=1}^{n}\left(1 - \frac{a_i}{B\cdot\beta}\right)\right]$ achieves its minimum when $a_i = \frac{B}{n}$ and that $\left[1 - \prod_{i=1}^{n}\left(1 - \frac{a_i}{B\cdot\beta}\right)\right] \geq 1 - \left(1 - \frac{1}{n\cdot\beta}\right)^n \geq 1 - e^{-\frac{1}{\beta}}$. Therefore, by applying Lemma 3 for $i = l+1$ and observing that $\sum_{\ell=1}^{l+1} \bar{c}_\ell = c(A_{l+1}, S_{l+1})$, we obtain:

$$\begin{aligned}
\sigma(A_{l+1}, S_{l+1}) &\geq \left[1 - \prod_{\ell=1}^{l+1}\left(1 - \frac{\bar{c}_\ell}{k\left(\frac{1+c_{\min}}{c_{\min}}\right)}\right)\right]\sigma(A^*, S^*) \\
&\geq \left[1 - \prod_{\ell=1}^{l+1}\left(1 - \frac{\bar{c}_\ell}{c(A_{l+1}, S_{l+1})\left(\frac{1+c_{\min}}{c_{\min}}\right)}\right)\right]\sigma(A^*, S^*) \\
&\geq \left[1 - \left(1 - \frac{1}{(l+1)\frac{1+c_{\min}}{c_{\min}}}\right)^{l+1}\right]\sigma(A^*, S^*) \geq \left(1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}\right)\sigma(A^*, S^*).
\end{aligned}$$

By Proposition 1, follows that:

$$\sigma(A_{l+1}, S_{l+1}) = \sigma(A_l, S_l) + \delta_{l+1} \geq \left(1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}\right)\sigma(A^*, S^*). \tag{4}$$

Since $\delta_{l+1} \leq \sigma(\{a_M\}, \{(a_M, v_M)\})$, we get

$$\sigma(A_l, S_l) + \sigma(\{a_M\}, \{(a_M, v_M)\}) \geq \left(1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}\right)\sigma(A^*, S^*).$$

Hence, $\max\{\sigma(A_l, S_l), \sigma(\{a_M\}, \{(a_M, v_M)\})\} \geq \frac{1}{2}\left(1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}\right)\sigma(A^*, S^*)$. ◀

We now propose an algorithm which improves the performance guarantee of Algorithm 1. Let M a fixed integer, we consider all the solutions (A, S) with cardinality M (i.e. $|A| + |S| = M$) and cost at most k ($c(A, S) \leq k$), and we complete all these solutions by using the greedy algorithm. The pseudocode is reported in Algorithm 2.

▶ **Theorem 5.** *For $M \geq 4$ Algorithm 2 achieves an approximation factor of $1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}$.*

Algorithm 2:
Input : A directed graph $G = (V, E)$, integer $M \in \mathbb{N}$ and an integer $k \in \mathbb{N}$
Output : A set of nodes A and a of edges $S \subseteq (A \times V) \setminus E$ such that $c(A, S) \leq k$
$(A_1, S_1) := \arg\max\{\sigma(A, S) : |A| + |S| < M, c(A, S) \leq k\}$;
$A_2 := \emptyset$; $S_2 := \emptyset$; $U := V$; $T := (V \times V) \setminus E$;
foreach $A \subseteq U, S \subseteq (A \times V) \setminus E$ such that $|A| + |S| = M$ and $c(A, S) \leq k$ **do**
  $U := U \setminus A$; $T := T \setminus S$;
  Complete (A, S) by using Algorithm 1 with U and T as possible nodes and edges;
  if $\sigma(A, S) > \sigma(A_2, S_2)$ **then**
    $A_2 := A$;
    $S_2 := S$;
return $\arg\max\{\sigma(A_1, S_1), \sigma(A_2, S_2)\}$;

Proof. We assume that $|A^*| + |S^*| > M$ since otherwise Algorithm 2 finds an optimal solution. We sort the elements in (A^*, S^*) by selecting at each step the element, which can be either a seed or an edge, that maximizes the marginal increment in the number of influenced nodes. Let $Z = (A_Z, S_Z)$ be the first M elements in this order. We now consider the iteration of Algorithm 2 in which element Z is considered. We define $(A_{Z'}, S_{Z'})$ as the elements added by the algorithm to (A_Z, S_Z) and $(A, S) = (A_Z \cup A_{Z'}, S_Z \cup S_{Z'})$. By Proposition 1 follows that $\sigma(A, S) = \sigma(A_Z, S_Z) + \delta(A_Z \cup A_{Z'}, S_Z \cup S_{Z'}, A_Z, S_Z)$.

The completion of (A_Z, S_Z) to (A, S) is an application of the greedy algorithm and therefore, we can use the result from the previous theorems. Let us consider the iterations executed by the greedy algorithm during the completion of (A_Z, S_Z) to (A, S). For $i \geq 1$, let us denote by j_i the index of these iterations, $j_i < j_{i+1}$, and let j_{l+1} be the index of the first iteration in which an element in $(A^* \setminus A_Z, S^* \setminus S_Z)$ is considered but not added to $(A_{Z'}, S_{Z'})$ because it violates the budget constraint. Applying inequality (4) to the instance of the problem obtained removing the nodes covered by solution Z we get: $\delta(A_Z \cup A_{Z'}, S_Z \cup S_{Z'}, A_Z, S_Z) + \delta_{l+1} \geq \left(1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}\right) \sigma(A^* \setminus A_Z, S^* \setminus S_Z)$.

Moreover, since we ordered the elements in (A^*, S^*) and in iteration j_{l+1} and most 2 elements are selected, then $\delta_{l+1} \leq \frac{2\sigma(A_Z, S_Z)}{M}$ and

$$\begin{aligned}
\sigma(A, S) &= \sigma(A_Z, S_Z) + \delta(A_Z \cup A_{Z'}, S_Z \cup S_{Z'}, A_Z, S_Z) \\
&\geq \sigma(A_Z, S_Z) + \left(1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}\right) \sigma(A^* \setminus A_Z, S^* \setminus S_Z) - \delta_{l+1} \\
&\geq \sigma(A_Z, S_Z) + \left(1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}\right) \sigma(A^* \setminus A_Z, S^* \setminus S_Z) - \frac{2\sigma(A_Z, S_Z)}{M} \\
&\geq \left(1 - \frac{2}{M}\right) \sigma(A_Z, S_Z) + \left(1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}\right) \sigma(A^* \setminus A_Z, S^* \setminus S_Z)
\end{aligned}$$

But, $\sigma(A_Z, S_Z) + \sigma(A^* \setminus A_Z, S^* \setminus S_Z) \geq \sigma(A^*, S^*)$, and we get:

$$\sigma(A, S) \geq \left(1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}\right) \sigma(A^*, S^*) + \left(\frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}} - \frac{2}{M}\right) \sigma(A_Z, S_Z) \geq \left(1 - \frac{1}{e^{\frac{c_{\min}}{1+c_{\min}}}}\right) \sigma(A^*, S^*),$$

for $M \geq 2e^{\frac{c_{\min}}{1+c_{\min}}}$. Since $2e^{\frac{c_{\min}}{1+c_{\min}}} < 4$ for $c_{\min} \in [0, 1]$, the theorem follows. ◀

4 General case

In this section we introduce an approximation algorithm for the BIMA problem that guarantees a constant approximation ratio, i.e. it does not depend on the edge costs.

The algorithm is similar to Algorithm 1, that is it chooses the best between two candidate solutions: one solution is made of a single node a_M and a single edge (a_M, v_M) that maximizes $\sigma(\{a_M\}, \{(a_M, v_M)\})$, the other solution is found by means of a greedy algorithm that at each step maximizes the ratio between the marginal increment in the objective function of a candidate solution and its marginal cost. The main difference consists in the way in which the greedy algorithm computes a candidate solution. In fact, the algorithm presented here might select more than one edge at one time. In particular, at each iteration the greedy algorithm computes four candidate solutions starting from the current solution (A', S'). The first two solutions correspond to cases r_1 and r_2 of Algorithm 1, and differ from (A', S') by a single seed node or a single edge incident to a seed in A', respectively. To compute the two further candidate solutions, the algorithm divides the edges into two sets: those whose cost is at least b and those whose cost is smaller than b, for some given constant b, and proceeds as follows.

- Select a seed node a not in A' and an edge (a, v), which cost is $c_{(a,v)} \geq b$, incident to a that maximize $\frac{\delta(A' \cup \{a\}, S' \cup \{(a,v)\}, A', S')}{1+c_{(a,v)}}$;
- Select a seed node a not in A' and a set S of edges incident to a, whose overall cost is smaller than b (i.e. $\sum_{(a,v)\in S} c_{(a,v)} < b$), that maximize $\frac{\delta(A' \cup \{a\}, S' \cup S, A', S')}{1+\sum_{(a,v)\in S} c_{(a,v)}}$.

The greedy phase of the algorithm selects the candidate solution that gives the maximum ratio among the above four possibilities.

To compute the fourth candidate solution we need to solve an optimization problem, which we call RBIMA (where R stands for ratio). In the following, we assume that we can exploit an α-approximation algorithm for this sub-problem. At the end of this section we will introduce a suitable algorithm for RBIMA.

The analysis of the algorithm is similar to that in the previous section. In particular, the next lemma is the core of the analysis and corresponds to Lemma 2. We denote by j_i an iteration of the greedy algorithm in which an element is added to the solution, by j_{l+1} the first iteration in which an element of the optimum is not added to the solution and by $\bar{c}_i$ the marginal cost of the solution (A_i, S_i) computed at the iteration j_i.

▶ **Lemma 6.** *After each iteration j_i, $i = 1, 2, \ldots, l+1$,*

$$\sigma(A_i, S_i) - \sigma(A_{i-1}, S_{i-1}) \geq \frac{\bar{c}_i}{k} \frac{b\alpha}{b\alpha + b + \alpha + 2} (\sigma(A^*, S^*) - \sigma(A_{i-1}, S_{i-1})).$$

Proof. We denote by δ_i the expected number of nodes affected by (A_i, S_i) and not affected by (A_{i-1}, S_{i-1}), $\delta_i = \delta(A_i, S_i, A_{i-1}, S_{i-1})$. We divide the set $A^* \setminus A_{i-1}$ into two subsets: A_1^* is the subset that contains the seeds a with no incident edges in S^*, and $A_2^* = A^* \setminus (A_{i-1} \cup A_1^*)$. For each $a \in A_2^*$ let us consider the subset of S^* containing edges incident to a such that $c_e < b$. We partition this set into sets of edges whose overall cost is smaller than b in such a way that the number of sets in the partition is minimized. We denote this a partition by $\mathcal{S}_a$ and observe that the overall number of sets in $\mathcal{S}_a$, for all $a \in A_2^*$, is at most $2\frac{k}{b}$ as it is equivalent to the minimum number of bins of size b needed to pack a set of items of overall size k. By using similar arguments as in Lemma 2, we can show that

$$\sigma(A^*,S^*)-\sigma(A_{i-1},S_{i-1}) \leq \sum_{a\in A_1^*}\delta(A_{i-1}\cup\{a\},S_{i-1},A_{i-1},S_{i-1})$$
$$+\sum_{\substack{e=(a,v)\in S^*\setminus S_{i-1}\\ \text{s.t. } a\in A_{i-1}}}\delta(A_{i-1},S_{i-1}\cup\{e\},A_{i-1},S_{i-1})+ \tag{5}$$
$$\sum_{\substack{a\in A_2^*\\ e=(a,v)\in S^*\setminus S_{i-1}\\ c_e\geq b}}\delta(A_{i-1}\cup\{a\},S_{i-1}\cup\{e\},A_{i-1},S_{i-1})+\sum_{\substack{a\in A_2^*\\ S\in\mathcal{S}_a}}\delta(A_{i-1}\cup\{a\},S_{i-1}\cup S,A_{i-1},S_{i-1}).$$

Indeed, $\sigma(A^*,S^*)-\sigma(A_{i-1},S_{i-1}) \leq \sum_{X\in\chi(S_{i-1})}\mathbb{P}[X]\sum_{Y\in\chi(S^*\cup S_{i-1},X)}\mathbb{P}[Y|X](|R(A^*\cup A_{i-1},Y)|-|R(A_{i-1},X)|)$ and for each $X\in\chi(S_{i-1})$ and $Y\in\chi(S^*\cup S_{i-1},X)$,

$$\begin{aligned}|R(A^*\cup A_{i-1},Y)|-|R(A_{i-1},X)| \leq &\sum_{a\in A_1^*}(|R(A_{i-1}\cup\{a\},X)|-|R(A_{i-1},X)|)\\ &+\sum_{\substack{e=(a,v)\in Y\setminus X\\ \text{s.t. } a\in A_{i-1}}}(|R(A_{i-1},X\cup\{e\})|-|R(A_{i-1},X)|)\\ &+\sum_{\substack{a\in A_2^*\\ e=(a,v)\in Y\setminus X\\ c_e\geq b}}(|R(A_{i-1}\cup\{a\},X\cup\{e\})|-|R(A_{i-1},X)|)\\ &+\sum_{\substack{a\in A_2^*\\ S\in\mathcal{S}_a}}(|R(A_{i-1}\cup\{a\},X\cup(S\cap Y))|-|R(A_{i-1},X)|)\,.\end{aligned}$$

The following holds since the greedy phase of the algorithm selects a solution that maximizes the ratio between the marginal increment in the objective function and the cost:

- For each $a\in A_1^*$, $\delta(A_{i-1}\cup\{a\},S_{i-1},A_{i-1},S_{i-1})\leq\frac{\delta_i}{\bar{c}_i}$;
- For each $e=(a,v)\in S^*\setminus S_{i-1}$ such that $a\in A_{i-1}$, $\frac{\delta(A_{i-1},S_{i-1}\cup\{e\},A_{i-1},S_{i-1})}{c_e}\leq\frac{\delta_i}{\bar{c}_i}$;
- For each $a\in A_2^*$ and $e=(a,v)\in S^*\setminus S_{i-1}$ s.t. $c_e\geq b$, $\frac{\delta(A_{i-1}\cup\{a\},S_{i-1}\cup\{e\},A_{i-1},S_{i-1})}{1+c_e}\leq\frac{\delta_i}{\bar{c}_i}$;
- For each $a\in A_2^*$ and $S\in\mathcal{S}_a$, $\frac{\delta(A_{i-1}\cup\{a\},S_{i-1}\cup S,A_{i-1},S_{i-1})}{1+\sum_{e\in S}c_e}\leq\frac{\delta_i}{\bar{c}_i}\cdot\frac{1}{\alpha}$,

where the term $\frac{1}{\alpha}$ in the last inequality is due to the use of an α-approximation algorithm in the computation of the fourth candidate solution of the greedy phase. The number of edges incident to each $a\in A_2^*$ with cost at least b is at most $\frac{k}{b}$. Therefore, the right hand side of (5) is at most:

$$\sum_{a\in A_1^*}\frac{\delta_i}{\bar{c}_i}+\sum_{\substack{e=(a,v)\in S^*\setminus S_{i-1}\\ \text{s.t. } a\in A_{i-1}}}\frac{\delta_i}{\bar{c}_i}c_e+\sum_{\substack{a\in A_2^*\\ e=(a,v)\in S^*\setminus S_{i-1}\\ c_e\geq b}}\frac{\delta_i}{\bar{c}_i}(1+c_e)+\sum_{\substack{a\in A_2^*\\ S\in\mathcal{S}_a}}\frac{1}{\alpha}\frac{\delta_i}{\bar{c}_i}\left(1+\sum_{e\in S}c_e\right)$$
$$=\frac{\delta_i}{\bar{c}_i}\left(|A_1^*|+\sum_{\substack{e=(a,v)\in S^*\setminus S_{i-1}\\ \text{s.t. } a\in A_{i-1}}}c_e+\frac{k}{b}+\sum_{\substack{a\in A_2^*\\ e=(a,v)\in S^*\setminus S_{i-1}\\ c_e\geq b}}c_e+\frac{1}{\alpha}\frac{2k}{b}+\frac{1}{\alpha}\sum_{\substack{a\in A_2^*\\ S\in\mathcal{S}_a}}\sum_{e\in S}c_e\right)$$
$$\leq\left(1+\frac{1}{b}+\frac{2}{b\alpha}+\frac{1}{\alpha}\right)k\frac{\delta_i}{\bar{c}_i}=\left(\frac{b\alpha+\alpha+2+b}{b\alpha}\right)k\frac{\delta_i}{\bar{c}_i}.$$

To conclude, we observe that by Proposition 1 follows that $\delta_i=\sigma(A_i,S_i)-\sigma(A_{i-1},S_{i-1})$. ◀

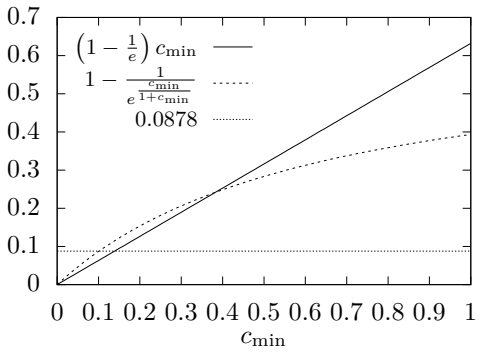


Figure 1 Approximation ratio of the three algorithm presented as a function of $c_{\min}$. Intersection points are at $c_{\min} \approx 0.1011$ and $c_{\min} \approx 0.3821$.

The statements of Lemmas 2 and 6 differ only for the coefficients $\frac{c_{\min}}{1+c_{\min}}$ and $\frac{b\alpha}{b\alpha+b+\alpha+2}$. Then, equipped with Lemma 6, we prove the next theorem, which corresponds to Theorem 4.

▶ **Theorem 7.** *The greedy algorithm achieves an approximation factor of* $1-\frac{1}{e^{\frac{b\alpha}{b\alpha+b+\alpha+2}}}$.

It remains to give an α-approximation algorithm for the RBIMA problem which consists in selecting a seed node a not in the current solution (A', S') and a set S of edges incident to a, which overall cost is smaller than b (i.e. $\sum_{(a,v)\in S} c_{(a,v)} < b$), that maximize $\frac{\delta(A'\cup\{a\}, S'\cup S, A', S')}{1+\sum_{(a,v)\in S} c_{(a,v)}}$. To this aim we define the CostIMA [8] problem as follows. Given a directed graph $G = (V, E, p, c)$, an integer $k \in \mathbb{N}$ and a seed set A, find a set of edges $S \subseteq A \times V \setminus E$ such that $c(S) \leq k$ and $\sigma(A, S)$ is maximized.

Let us consider the instances of the CostIMA problem where $A = \{a\}$ and let m^* be the maximum, over all $a \in V \setminus A'$, among the optima of these instances. It is easy to show that m^* is at least $b+1$ times the optimum of RCostIMA. It has been shown that the CostIMA problem can be approximated within a factor of $1-\frac{1}{e}$ by using a greedy algorithm [8]. Therefore we obtain an overall approximation of $\alpha = \left(1-\frac{1}{e}\right)\frac{1}{b+1}$. The next corollary follows by applying Theorem 7 with $\alpha = \left(1-\frac{1}{e}\right)\frac{1}{b+1}$ and optimizing over b.

▶ **Corollary 8.** *There exists an algorithm that achieves an approximation factor* > 0.0878 *for the* BIMA *problem.*

5 Conclusions

In Figure 1 we summarize our approximation ratios as a function of $c_{\min}$. As expected, when all the edge costs are high, it is not worth to buy them. Indeed the algorithm that selects only seeds outperforms the other algorithms and it reaches the optimal approximation of $1-\frac{1}{e}$ when all the costs are equal to 1. However, the challenge of our problem consists in finding a good approximation even when the cost function includes small values. In the general case and when the minimum edge cost can be very small ($c_{\min} < 0.1011$) the best algorithm is the constant factor algorithm given in Section 4, while when the minimum cost is in $(0.1011, 0.3821)$ the best algorithm is the one presented in Section 3.

The main open problem is the gap between the lower bound on approximation of $1-\frac{1}{e} \approx 0.6321$ and the constant approximation of 0.0878. To close this gap, we aim at improving the algorithm in Section 4 by devising a better approximation algorithm for

RCostIMA problem, which directly implies a better approximation for the BIMA problem. Other research directions that deserve further investigation include the study of the BIMA problem on different information diffusion models such as LTM or the Triggering Model [15]. Moreover, we plan to design efficient heuristics in order to assess the performance of our greedy algorithm from the experimental point of view.

References

1. C. Asavathiratham, S. Roy, B. Lesieutre, and G. Verghese. The influence model. *IEEE Control Systems*, 21(6):52–64, 2001.
2. Eytan Bakshy, Jake M. Hofman, Winter A. Mason, and Duncan J. Watts. Everyone's an influencer: Quantifying influence on twitter. In *Proc. of the 4th ACM International Conference on Web Search and Data Mining*, WSDM11, pages 65–74. ACM, 2011.
3. Frank M Bass. A new product growth for model consumer durables. *Management science*, 15(5):215–227, 1969.
4. Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD10, 2010.
5. James S. Coleman, Elihu Katz, and Herbert Menzel. *Medical innovation: A diffusion study*. The Bobbs-Merrill Company, 1966.
6. Pierluigi Crescenzi, Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj. Greedily improving our own closeness centrality in a network. *ACM Trans. Knowl. Discov. Data*, 11(1):9:1–9:32, 2016.
7. 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, Lecce, Italy, September 7-9, 2016.*, pages 269–274, 2016.
8. Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj. Recommending links through influence maximization. *arXiv preprint*, 2017. URL: `http://arxiv.org/abs/1706.04368`.
9. Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj. Selecting nodes and buying links to maximize the information diffusion in a network. *arXiv preprint*, 2017. URL: `https://arxiv.org/abs/1706.06466`.
10. Pedro Domingos and Matt Richardson. Mining the network value of customers. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD01, pages 57–66. ACM, 2001.
11. Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters*, 12(3):211–223, 2001.
12. Jacob Goldenberg, Barak Libai, and Eitan Muller. Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata. *Academy of Marketing Science Review*, 2001(9):1, 2001.
13. Mark Granovetter. Threshold models of collective behavior. *American journal of sociology*, 83(6):1420–1443, 1978.
14. David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD03, pages 137–146. ACM, 2003.
15. David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11:105–147, 2015.
16. Elias Boutros Khalil, Bistra Dilkina, and Le Song. Scalable diffusion-aware optimization of network topology. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD14, pages 1226–1235. ACM, 2014.

17 Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999.

18 Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. Solving the contamination minimization problem on networks for the linear threshold model. In *Proc. of the 10th Pacific Rim International Conference on Artificial Intelligence*, PRICA08, pages 977–984. Springer Berlin Heidelberg, 2008.

19 Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. Blocking links to minimize contamination spread in a social network. *ACM Trans. Knowl. Discov. Data*, 3(2):9:1–9:23, 2009.

20 Chris J Kuhlman, Gaurav Tuli, Samarth Swarup, Madhav V Marathe, and SS Ravi. Blocking simple and complex contagion by edge removal. In *IEEE International Conference on Data Mining*, ICDM13. IEEE, 2013.

21 Vijay Mahajan, Eitan Müller, and Frank M Bass. New product diffusion models in marketing: A review and directions for research. *Journal of Marketing*, 54:1–26, 1990.

22 Lauren Meyers. Contact network epidemiology: Bond percolation applied to infectious disease prediction and control. *Bulletin of the American Mathematical Society*, 44(1):63–86, 2007.

23 M. E. J. Newman. Spread of epidemic disease on networks. *Phys. Rev. E*, 66:016128, Jul 2002.

24 H. Nguyen and R. Zheng. On budgeted influence maximization in social networks. *IEEE Journal on Selected Areas in Communications*, 31(6):1084–1094, June 2013. doi:10.1109/JSAC.2013.130610.

25 Manos Papagelis. Refining social graph connectivity via shortcut edge addition. *ACM Trans. Knowl. Discov. Data*, 10(2):12, 2015.

26 Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD02, pages 61–70. ACM, 2002.

27 E. Rogers. *Diffusion of innovations*. Free Press, 1995.

28 Thomas C Schelling. *Micromotives and macrobehavior*. Norton & Company, 2006.

29 Daniel Sheldon, Bistra N. Dilkina, Adam N. Elmachtoub, Ryan Finseth, Ashish Sabharwal, Jon Conrad, Carla P. Gomes, David B. Shmoys, William Allen, Ole Amundsen, and William Vaughan. Maximizing the spread of cascades using network design. *CoRR*, abs/1203.3514, 2012. URL: http://arxiv.org/abs/1203.3514.

30 T. Valente. *Network models of the diffusion of innovations*. Hampton Press, 1995.

31 Xiaojian Wu, Daniel Sheldon, and Shlomo Zilberstein. Efficient algorithms to optimize diffusion processes under the independent cascade model. In *NIPS Workshop on Networks in the Social and Information Sciences*, Montreal, Quebec, Canada, 2015.

32 Y. Zhang, A. Adiga, A. Vullikanti, and B. A. Prakash. Controlling propagation at group scale on networks. In *IEEE International Conference on Data Mining*, ICDM15, pages 619–628, 2015.

K_4-Free Graphs as a Free Algebra*

Enric Cosme-Llópez[1] and Damien Pous[2]

1 **Univ. Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France**
2 **Univ. Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France**

Abstract

Graphs of treewidth at most two are the ones excluding the clique with four vertices as a minor. Equivalently, they are the graphs whose biconnected components are series-parallel.

We turn those graphs into a free algebra, answering positively a question by Courcelle and Engelfriet, in the case of treewidth two. First we propose a syntax for denoting them: in addition to series and parallel compositions, it suffices to consider the neutral elements of those operations and a unary transpose operation. Then we give a finite equational presentation and we prove it complete: two terms from the syntax are congruent if and only if they denote the same graph.

1998 ACM Subject Classification G.2.2 Graph Theory, F.4.3 Formal Languages.

Keywords and phrases Universal Algebra, Graph theory, Axiomatisation, Graph minors.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.76

1 Introduction

The notion of treewidth is a cornerstone in (algorithmic) graph theory [16]. It measures how close a graph is to a forest, and classes of graphs of bounded treewidth often enjoy good computational properties. For instance, graph homomorphism (and thus k-colouring) becomes polynomial-time [19, 7, 21], so does model-checking of Monadic Second Order (MSO) formulae, and satisfiability of MSO formulae becomes decidable, even linear [9]. (See the monograph of Courcelle and Engelfriet about monadic second order logic on graphs [13].)

Here we focus on graphs of treewidth at most two. They coincide with the *partial* 2*-trees*, with the K_4*-free* graphs (those that exclude the clique with four vertices (K_4) as a *minor*), and with the graphs whose biconnected components are *series-parallel* [18, 5].

We consider the set Gph of directed graphs with edges labelled with letters $a, b, \ldots$ in some alphabet Σ, and with two distinguished vertices, called the *input* and the *output*. We represent such graphs as usual, using an unlabelled ingoing (resp. outgoing) arrow to denote the input (resp. output). Such graphs can be composed:

- in *parallel* by putting them side by side, merging their inputs, and merging their outputs;
- in *series* by putting them one after the other and merging the output of the first one with the input of the second one.

Every letter of the alphabet gives rise to a graph consisting of two vertices (the input and the output), and a single edge from the input to the output, labelled with that letter.

If we allow only those operations, we obtain the series-parallel graphs, and it is easy to see that these form the free algebra over the signature $\langle \parallel, \cdot \rangle$, where $\parallel$ is an associative-commutative binary operation (parallel composition), and $\cdot$ is an associative binary operation

* An extended version of this abstract, including proofs, is available on HAL [8]. This work was supported by the European Research Council (ERC) under the Horizon 2020 programme (CoVeCe, grant agreement No 678157) and the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007)

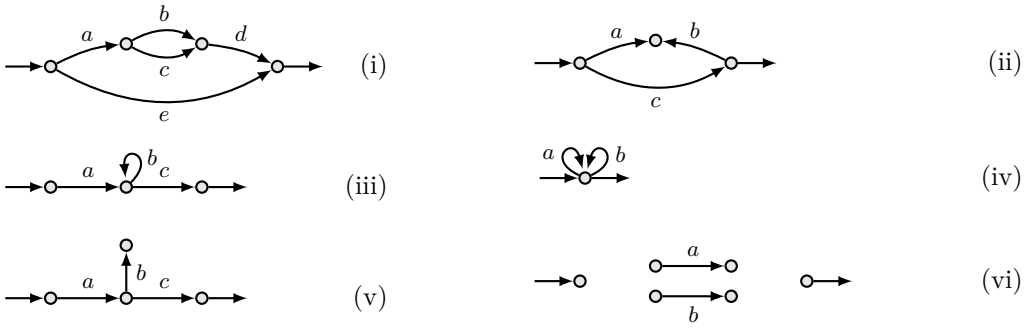


Figure 1 Some graphs of treewidth at most two

(series composition). For instance, the terms $((a{\cdot}(b \parallel c)){\cdot}d) \parallel e$ and $e \parallel (a{\cdot}((c \parallel b){\cdot}d))$ both denote the graph (i) in Figure 1; they are equal up to associativity of $\cdot$ and commutativity of $\parallel$. Parallel composition is not idempotent: $(a{\cdot}b) \parallel (a{\cdot}b)$ and $a{\cdot}b$ denote distinct graphs.

However, we cannot denote all graphs of treewidth at most two in such a way.

First the notion of treewidth does not depend on the orientation of the edges. For instance, the graph (ii) has treewidth two, yet it is not the image of a term in the previous syntax. To this end, we add a unary operation $\cdot^\circ$ to our signature, which we interpret in graphs as the exchange of input and output. Doing so, the terms $(a{\cdot}b^\circ) \parallel c$ and $(b{\cdot}a^\circ)^\circ \parallel c$ denote the graph (ii), and series-parallel graphs with converse become a free algebra when we ask that $\cdot^\circ$ is an involution that distributes over $\parallel$ and satisfies $(a{\cdot}b)^\circ = b^\circ{\cdot}a^\circ$.

Second, the treewidth of a graph does not depend on self-loops, so that the graph (iii) actually has treewidth one (it is a tree once we remove the self-loop). There it suffices to add a constant, 1, interpreted as the graph with a single vertex (both input and output), and no edge. Doing so, the graph (iii) is denoted by $a{\cdot}(1 \parallel b){\cdot}c$. While the constant 1 is clearly a neutral element for series composition $(\cdot)$, axiomatising its interactions with parallel composition is much harder, and is actually one of the key contributions of the present work. For instance, the equation $(1 \parallel a){\cdot}(1 \parallel b) = 1 \parallel a \parallel b$ belongs to the theory as both sides denote the graph (iv).

Up-to this point, we have recovered the syntax of *allegories* [20], and the graphs associated to the terms are precisely the ones Freyd and Scedrov use to obtain that the theory of representable allegories is decidable, yet not finitely presentable [20, p. 210].

But we still miss some graphs, like (v). One can also remark that we only obtain connected graphs using the above operations. Instead, treewidth allows disconnected graphs: a graph has a given treedwidth if and only if all of its connected components do. Surprisingly, it suffices to add a second constant, $\top$, interpreted as the disconnected graph with no edges and two distinct vertices (the input and the output). This allows us to obtain disconnected graphs, but also to get a term for the connected graph (v), namely, $a{\cdot}((b{\cdot}\top) \parallel c)$. Again while it is clear that this constant is a neutral element for parallel composition ($\parallel$), capturing its interactions with the other operations is non-trivial. For instance, $\top{\cdot}a{\cdot}\top{\cdot}b{\cdot}\top$ and $\top{\cdot}b{\cdot}\top{\cdot}a{\cdot}\top$ both denote the graph (vi), and should thus be equated.

To sum up, the set Gph of graphs forms an algebra for the signature $\langle \cdot_2, \parallel_2, \cdot^\circ_1, 1_0, \top_0 \rangle$; the various operations of this algebra are depicted in Figure 2.

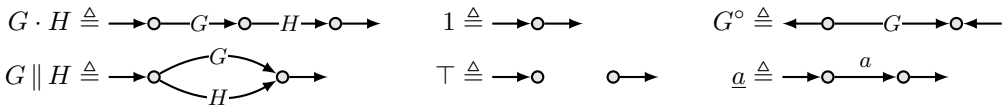
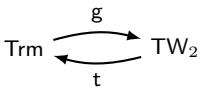


Figure 2 The 2p-algebra of graphs and the graph of a letter

Write Trm for the set of terms over the alphabet Σ and TW_2 for the set of graphs of treewidth at most two when an extra edge is added between input and output[1]. One easily proves that the latter set actually forms a subalgebra of the algebra of graphs. Therefore, the function interpreting each term as a graph actually gives a function $\mathsf{g} : \mathsf{Trm} \to \mathsf{TW}_2$.

We first prove that this function has a right-inverse: we define a function $\mathsf{t} : \mathsf{TW}_2 \to \mathsf{Trm}$ such that for all graph $G \in \mathsf{TW}_2$, $\mathsf{g}(\mathsf{t}(G))$ is isomorphic to G:



$$\mathsf{g}(\mathsf{t}(G)) \simeq G \tag{1}$$

By doing so, we get that the graphs of treewidth at most two are exactly the ones that can be expressed using the syntax.

Our key contribution then consists in giving a finite equational axiomatisation of graph isomorphism over this syntax. This answers positively the question asked by Courcelle and Engelfriet in their book, for treewidth two [13, p. 118].

Note that the choice of the syntax is important. Various finite syntaxes have already been proposed [15, 16, 13] to capture graphs of treewidth at most k, for a given k. However, some choices prevent finite presentations. For instance, while the *converse* operation we use could be eliminated by pushing it to the leaves, doing so would turn some of our axioms into infinite equational schemes. (See also Remark 27.)

As explained above, a few laws are rather natural like associativity of the two compositions, commutativity of $\parallel$, or the facts that 1 and $\top$ are neutral elements and that $\cdot^\circ$ is an involution. Those are the first eight laws in Figure 3. Surprisingly, the four subsequent axioms suffice to obtain a complete axiomatisation: for all terms u, v, u and v are provably equal using the axioms from Figure 3 if and only if $\mathsf{g}(u)$ and $\mathsf{g}(v)$ are isomorphic:

$$u \equiv v \quad \Leftrightarrow \quad \mathsf{g}(u) \simeq \mathsf{g}(v) \tag{2}$$

In other words, calling a *2p-algebra* an algebra satisfying the axioms from Figure 3, TW_2 is the free 2p-algebra.

All axioms but (A3) are independent; (A3) follows from (A9) and (A12). Correctness, i.e., the left-to-right implication in (2), is easy to establish. Indeed, it suffices to compute and compare the graphs of each equation, and to prove that valid equations are stable under graph substitution. The converse implication, completeness, is much harder. This is because there is no canonical way of extracting a term out of a graph. In particular, the function t we define to this end has to make choices based on the concrete representation of the input graph, so that isomorphic graphs do not always map to syntactically equal terms.

[1] This additional condition is natural when considering pointed graphs [11]; this is not a restriction for unpointed graphs as one can always set input and output to the same arbitrary node.

$$u \parallel (v \parallel w) \equiv (u \parallel v) \parallel w \quad \text{(A1)} \qquad u{\cdot}(v{\cdot}w) \equiv (u{\cdot}v){\cdot}w \quad \text{(A4)}$$

$$u \parallel v \equiv v \parallel u \quad \text{(A2)} \qquad u{\cdot}1 \equiv u \quad \text{(A5)}$$

$$u \parallel \top \equiv u \quad \text{(A3)}$$

$$1 \parallel 1 \equiv 1 \quad \text{(A9)}$$

$$u^{\circ\circ} \equiv u \quad \text{(A6)} \qquad 1 \parallel u{\cdot}v \equiv 1 \parallel (u \parallel v^{\circ}){\cdot}\top \quad \text{(A10)}$$

$$(u \parallel v)^{\circ} \equiv u^{\circ} \parallel v^{\circ} \quad \text{(A7)} \qquad u{\cdot}\top \equiv (1 \parallel u{\cdot}\top){\cdot}\top \quad \text{(A11)}$$

$$(u{\cdot}v)^{\circ} \equiv v^{\circ}{\cdot}u^{\circ} \quad \text{(A8)} \qquad (1 \parallel u){\cdot}v \equiv (1 \parallel u){\cdot}\top \parallel v \quad \text{(A12)}$$

Figure 3 Twelve axioms for 2p-algebras, all independent except for (A3)

We proceed in the following way to obtain completeness. First we prove that the function t maps isomorphic graphs to congruent terms:

$$G \simeq H \qquad \Rightarrow \qquad \mathsf{t}(G) \equiv \mathsf{t}(H) \tag{3}$$

Then we prove that this function is a homomorphism (up to the axioms), which allows us to deduce that for all terms u, $\mathsf{t}(\mathsf{g}(u))$ is provably equal to u:

$$\mathsf{t}(\mathsf{g}(u)) \equiv u \tag{4}$$

In a sense, by interpreting a term u into a graph and then reading it back, we obtain a term $\mathsf{t}(\mathsf{g}(u))$ which plays the role of a normal form even if it is not canonical (which would typically be the case in rewriting theory, or in normalisation by evaluation [4]).

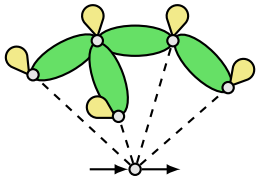
Defining a function t satisfying (1) could be done rather easily by relying on the notion of tree decomposition. However, doing so makes it extremely difficult to obtain properties (3) and (4): the notion of tree decomposition, despite its inductive nature, does not provide enough structure. Instead, we use the fact that treewidth at most two graphs are K_4-free, and we exhibit stronger graph invariants that allow us to extract terms from graphs in a much more structured way.

For instance, when the graph is connected and when its input and output are distinct, one can compute its *checkpoints*: those vertices which all paths from the input to the output must visit. Those checkpoints are linearly ordered so that the graph has the following shape

(5)

If there is at least one checkpoint then the graph should be interpreted as a series composition. Otherwise, by the absence of K_4 as a minor, one can show that the graph necessarily is a non-trivial parallel composition.

The aforementioned step is already there in the standard result that the biconnected components of a K_4-free graph are series-parallel. More challenging is the case when the input and output coincide. In this case, we consider the checkpoints of all pairs of neighbours of the input, and we show that they form a tree which is a minor of the starting graph.



This tree is a key invariant of the isomorphism class of the graph and we show that one can extract a term for each choice of a node in this tree. This is where our function $\mathtt{t}$ has to rely on the concrete representation of the graph: although all choices of a node in the tree result in provably equal terms, they do not yield syntactically equal terms. A similar situation happens with components which are disconnected from the input and the output: we handle those recursively by taking any vertex as a new choice of input and output.

2 Related work

Except for the presence of $\top$, the algebra of graphs we work with has been proposed independently by Freyd and Scedrov [20, p. 207], and by Andréka and Bredikhin [1]. They used it to characterise the equational theory of binary relations over the considered signature. Indeed the set of binary relations over a fixed set forms an algebra for the signature we consider in this paper: $\cdot$ is relational composition, $\parallel$ is set-theoretic intersection (thus it is written $\cap$ in [20, 1]), $\cdot^\circ$ is transposition, 1 is the identity relation and $\top$ is the full relation. Writing $\mathrm{Rel} \vDash u \leq v$ for the containments that hold in all such algebras of relations, and $G \blacktriangleleft H$ if there exists a graph homomorphism from H to G, we have the following equivalence.

$$\mathrm{Rel} \vDash u \leq v \qquad \Leftrightarrow \qquad \mathsf{g}(u) \blacktriangleleft \mathsf{g}(v)$$

This characterisation immediately gives decidability: existence of a graph homomorphism is an NP-complete problem. Thanks to the present observation that graphs of terms have bounded treewidth, the complexity is actually polynomial [21].

Freyd and Scedrov also use this characterisation to prove that this theory is not finitely presentable [20]: every complete equational axiomatisation must contain axioms corresponding to homomorphisms equating arbitrarily many vertices at a time, and thus must be infinite. Andréka and Bredikhin go even further and show that it is not even a variety [1].

In this work we focus on isomorphism rather than on homomorphism, and this is why we do obtain a finite equational axiomatisation. Although all algebras of relations validate our axioms, these algebras cannot be free models. For instance, their parallel composition (intersection) is always idempotent. Freyd and Scedrov remark that certain graphs cannot be the image of a term [20, p. 207], and Andréka and Bredikhin use a weak form of the K_4 exclusion property [1, Lemma 7]. They cannot obtain a characterisation result since they do not consider $\top$, which is necessary to reach all graphs of treewidth at most two.

Our work is also really close to that of Dougherty and Gutiérrez [17], who proposed an axiomatisation of graph isomorphism for a slightly different syntax: instead of the constant $\top$, they use a unary operation $\mathrm{dom}(\cdot)$, called *domain*. This operation can be defined in our setting: we have $\mathrm{dom}(u) = 1 \parallel (u \cdot \top)$; at the graphical level, it consists in relocating the output of a graph on its input. In contrast, $\top$ cannot be defined in terms of $\mathrm{dom}(\cdot)$ and the other operations. Choosing this domain operation has the advantage of keeping connected graphs, and the disadvantage of being less general: disconnected graphs cannot be expressed. More importantly, the operation $\top$ being more primitive than $\mathrm{dom}(\cdot)$, we can obtain a shorter axiomatisation: while we share with [17] the nine natural axioms from Figure 3 that do not mention $\top$, the four remaining ones in this figure have to be replaced by nine axioms when using $\mathrm{dom}(\cdot)$: three about 1 and $\parallel$, and six about $\mathrm{dom}(\cdot)$. To prove completeness, Dougherty and Gutiérrez compute normal forms for terms using rewriting techniques. Like in the present work, their normal forms are not canonical and some additional work is needed. In a second part of the paper, they characterise graphs of terms using a minor exclusion theorem which corresponds precisely to what we obtain in

the connected case (see Remark 27). There are however several typos or gaps in their paper which we were not able to fix—see [8] for more details.

Bauderon and Courcelle gave a syntax and a complete axiomatisation for arbitrary graphs [3]. While the overall statement is similar to ours, their syntax can hardly be related to the present one (it is infinitary, for instance), and the present results are not corollaries of their work. The structural invariants we exhibit here are reminiscent of the general decomposition results of Tutte [27], which Courcelle later studied in the context of MSO [12].

3 2p-algebra

We consider the signature $\langle \cdot_2, \parallel_2, \cdot^\circ_1, 1_0, \top_0 \rangle$ and we let u, v, w range over *terms* over a set Σ of variables. 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 $ab^\circ \parallel c$. A *2p-algebra* is an algebra over this signature satisfying the axioms from Figure 3. We write $u \equiv v$ when two terms u and v are congruent modulo those axioms, or equivalently, when the equation holds in all 2p-algebras.

Following notations from Kleene algebra with tests (KAT) [22], we let α, β range over *tests*, those terms that are congruent to some term of the shape $1 \parallel u$. (By axiom A9, u is a test iff $u \equiv 1 \parallel u$.) Graphs of tests are those whose input and output coincide.

We shall use the derived operation mentioned in Section 2, *domain*, as well as its dual, *codomain*: $\mathrm{dom}(u) \triangleq 1 \parallel u\top$ and $\mathrm{cod}(u) \triangleq 1 \parallel \top u$. Those are tests by definition.

As is standard for involutive monoids, the first eight axioms from Figure 3 entail $1^\circ \equiv 1$, $\top^\circ \equiv \top$, and $1u \equiv u$. We use such laws freely in the sequel. We recall the four remaining axioms below, using the above notations.

$$1 \parallel 1 \equiv 1 \qquad \text{(A9)} \qquad\qquad u\top \equiv \mathrm{dom}(u)\top \qquad \text{(A11)}$$

$$1 \parallel uv \equiv \mathrm{dom}(u \parallel v^\circ) \qquad \text{(A10)} \qquad\qquad \alpha v \equiv \alpha\top \parallel v \qquad \text{(A12)}$$

Thanks to converse being an involution, there is a notion of duality in 2p-algebras: one obtains a valid law when swapping the arguments of all products and exchanging domains with codomains in a valid law. (We have $\mathrm{cod}(u) \equiv \mathrm{dom}(u^\circ)$.)

▶ **Proposition 1.** *The following equations hold in all 2p-algebras.*

$$\alpha^\circ \equiv \alpha \qquad (6)$$

$$\alpha\beta \equiv \alpha \parallel \beta \qquad (7)$$

$$\alpha(v \parallel w) \equiv \alpha v \parallel w \qquad (8)$$

$$(v \parallel w)\alpha \equiv v\alpha \parallel w \qquad (9)$$

$$\mathrm{dom}(uv \parallel w) \equiv \mathrm{dom}(u \parallel wv^\circ) \qquad (10)$$

$$\top u^\circ \top \equiv \top u \top \qquad (11)$$

$$u\top w \equiv u\top \parallel \top w \qquad (12)$$

$$u\top v\top w \equiv u\top w \parallel \top v\top \qquad (13)$$

$$(u \parallel \top v\top)w \equiv uw \parallel \top v\top \qquad (14)$$

4 Graphs

As explained in the introduction, we consider labelled directed graphs with two designated vertices. We just call them graphs in the sequel. Note that we allow multiple edges between two vertices, as well as self-loops.

▶ **Definition 2.** 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, $l : E \to \Sigma$ is map indicating the *label* of each edge, and $\iota, o \in V$ are the designated vertices, respectively called *input* and *output*.

We write $G[x;y]$ for the graph G with input set to x and output set to y; we abbreviate $G[x;x]$ to $G[x]$.

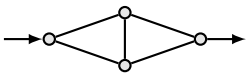
▶ **Definition 3.** An *homomorphism* from $G = \langle V, E, s, t, l, \iota, o\rangle$ to $G' = \langle V', E', s', t', l', \iota', o'\rangle$ is a pair $h = \langle f, g\rangle$ of functions $f : V \to V'$ and $g : E \to E'$ that respect the various components: $s' \circ g = f \circ s$, $t' \circ g = f \circ t$, $l' = g \circ l$, $\iota' = f(\iota)$, 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'.

▶ **Proposition 4.** *Graphs up to isomorphism form a 2p-algebra.*

▶ **Definition 5.** Let $G = \langle V, E, s, t, l, \iota, o\rangle$ be a graph. A *tree decomposition* of G is a tree T where each node t is labelled with a set $V_t \subseteq V$ of vertices, such that:
(T1) for every vertex $x \in V$, the set of nodes t such that $x \in V_t$ forms a sub-tree of T;
(T2) for every edge $e \in E$, there exists a node t such that $\{s(e), t(e)\} \subseteq V_t$;
(T3) there exists a node t such that $\{\iota, o\} \subseteq V_t$.
The *width* of a tree decomposition is the size of the largest set V_t minus one; the *treewidth* of a graph is the minimal width of a tree decomposition for this graph. We write TW_2 for the set of graphs of treewidth at most two.

The first two conditions in the definition of tree decomposition are standard; the third one is related to the presence of distinguished nodes: it requires them to lie together in some node of the tree. This condition ensures that the following graph is excluded from TW_2 whatever the orientation and labelling of its edges.



(M_3)

Indeed, such a graph cannot be represented in the syntax we consider. (Something already observed by Freyd and Scedrov [20]—the addition of $\top$ to the syntax does not help.)

▶ **Proposition 6.** *Graphs of treewidth at most two form a subalgebra of the algebra of graphs.*

The graphs we associate to each letter (Figure 2) also belong to this subalgebra, so that we obtain a homomorphism $\mathsf{g} : \mathsf{Trm} \to \mathsf{TW}_2$ associating a graph of treewidth at most two to each syntactic term. When taking quotients under term congruence and graph isomorphism, this function becomes a 2p-algebra homomorphism $\mathsf{g}' : \mathsf{Trm}_{/\equiv} \to \mathsf{TW}_{2/\simeq}$. Our key result is that g' actually is an isomorphism of 2p-algebras (Corollary 33).

5 K_4-freeness

In this section we establish preliminary technical results about unlabelled undirected graphs with at most one edge between two vertices and without self-loops; we call those *simple graphs*. We use standard notation and terminology from graph theory [16]. In particular, we denote by xy a potential edge between two vertices x and y; an xy-path is a (possibly trivial) path whose ends are x and y; $G + xy$ is the simple graph obtained from G by adding the edge xy if x and y were not already adjacent; $G \backslash x$ is the simple graph obtained from G by removing the vertex x and its incident edges.

▶ **Definition 7.** A *minor* of a simple graph G is a simple graph obtained by performing a sequence of the following operations on G: delete an edge or a vertex, contract an edge.

A cornerstone result of graph theory, Robertson and Seymour's graph minor theorem [26], states that (simple) graphs are well-quasi-ordered by the minor relation. As a consequence, the classes of graphs of bounded treewidth, which are closed under taking minors, can be characterised by finite sets of excluded minors. Two simple and standard instances are the following ones: the graphs of treewidth at most one (the forests) are precisely those excluding the cycle with three vertices (C_3); those of treewidth at most two are those excluding the complete graph with four vertices (K_4) [18]. We eventually reprove the latter one here.

 (C_3)  (K_4)

We fix a connected simple graph G in the remainder of this section.

▶ **Definition 8.** The *checkpoints* between two vertices x, y are the vertices which any xy-path must visit: $\mathsf{CP}(x, y) \triangleq \{z \mid \text{every } xy\text{-path crosses } z\}$.

For all vertices x, y, we have $\mathsf{CP}(x, x) = \{x\}$ and $\{x, y\} \subseteq \mathsf{CP}(x, y) = \mathsf{CP}(y, x)$. Two vertices x, y are *linked*, written $x \Diamond y$, when $x \neq y$ and $\mathsf{CP}(x, y) = \{x, y\}$, i.e., when there are no proper checkpoints between x and y. The *link graph* of G is the graph of linked vertices. Note that G is a subgraph of its link graph: if xy is an edge in G then $x \Diamond y$. We also have the following properties.

▶ **Lemma 9.** *Any cycle in the link graph is actually a clique.*

▶ **Lemma 10.** *If xyz is a triangle in the link graph and ι is a vertex not in G, then the graph $G + \iota x + \iota y + \iota z$ admits K_4 as a minor.*

Now fix a set U of vertices; we extend the notion of checkpoints as follows.

▶ **Definition 11.** The *checkpoints* of U, $\mathsf{CP}(U)$, is the set of vertices which are checkpoints of some pair in U: $\mathsf{CP}(U) \triangleq \bigcup_{x,y \in U} \mathsf{CP}(x, y)$. The *checkpoint graph* of U is the subgraph of the link graph induced by this set. We also denote this graph by $\mathsf{CP}(U)$.

▶ **Lemma 12.** CP *is a closure operator on the set of vertices. In particular, for all checkpoints $x, y \in \mathsf{CP}(U)$, $\mathsf{CP}(x, y) \subseteq \mathsf{CP}(U)$.*

▶ **Lemma 13.** *For every path in G between two checkpoints $x, y \in \mathsf{CP}(U)$, the sequence obtained by keeping only the elements in $\mathsf{CP}(U)$ is an xy-path in $\mathsf{CP}(U)$.*

Since G is assumed to be connected, it follows that so is $\mathsf{CP}(U)$. A key instance of a checkpoint graph is when U only contains two vertices, presumably the input and output of some graph: the checkpoint graph is a line in this case, as in (5), and it allows us to decompose the considered graph into a sequence of series compositions.

▶ **Lemma 14.** *If $U = \{x, y\}$, then $\mathsf{CP}(U)$ is a line graph whose ends are x and y.*

The following two lemmas are helpful in Proposition 20 below, to prove that the checkpoint graph is a tree under certain circumstances.

▶ **Lemma 15.** *If xy is an edge in $\mathsf{CP}(U)$, then there exists $x', y' \in U$ such that x and y belong to $\mathsf{CP}(x', y')$.*

▶ **Lemma 16.** *If xyz is a triangle in $\mathsf{CP}(U)$, then there exists $x', y', z' \in U$ such that x and y (resp. x and z, y and z) belong to $\mathsf{CP}(x', y')$ (resp. $\mathsf{CP}(x', z')$, $\mathsf{CP}(y', z')$).*

As explained above we use the checkpoint graphs to decompose graphs. The following notions of intervals and bags are the basic blocks of those decompositions.

▶ **Definition 17.** Let x, y be two vertices. The *strict interval* $]\!]x;y[\!]$ is the following set of vertices.

$$]\!]x;y[\!] \triangleq \{p \mid \text{there is an } xp\text{-path avoiding } y \text{ and a } py\text{-path avoiding } x\}$$

The *interval* $[\![x;y]\!]$ is obtained by adding x and y to that set. We abuse notation and write $[\![x;y]\!]$ for the subgraph of G induced by the set $[\![x;y]\!]$.

Note that while the intervals do not depend on the set U, we mostly use them under the assumption that xy is an edge in a checkpoint graph.

▶ **Definition 18.** The *bag* of a checkpoint $x \in \mathsf{CP}(U)$ is the set of vertices that need to cross x in order to reach the other checkpoints.

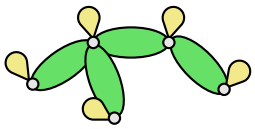
$$[\![x]\!]_U \triangleq \{p \mid \forall y \in \mathsf{CP}(U),\ \text{any } py\text{-path crosses } x\}\,.$$

As before, we also write $[\![x]\!]_U$ for the induced subgraph of G.

Note that $[\![x]\!]_U$ depends on U and differs from $[\![x;x]\!]$ (which is always the singleton $\{x\}$).

▶ **Proposition 19.** *If $\mathsf{CP}(U)$ is a tree, then the following set $\mathcal{V}$ is a partition of the vertices of G such that any edge of G appears in exactly one graph of the set $\mathcal{E}$.*

$$\mathcal{V} \triangleq \{[\![x]\!]_U \mid x \in \mathsf{CP}(U)\} \cup \{]\!]x;y[\!] \mid xy \text{ edge in } \mathsf{CP}(U)\}$$
$$\mathcal{E} \triangleq \{[\![x]\!]_U \mid x \in \mathsf{CP}(U)\} \cup \{[\![x;y]\!] \mid xy \text{ edge in } \mathsf{CP}(U)\}$$



Graphically, this means G can be decomposed as in the picture above; only the vertices of $\mathsf{CP}(U)$ are depicted, the green blocks correspond to edges in $\mathsf{CP}(U)$, the yellow blocks correspond to the graphs $[\![x]\!]_U$. The leaves of $\mathsf{CP}(U)$ are elements of U (but not always conversely). As a consequence, when $\mathsf{CP}(U)$ is a tree, it is a minor of G: contract all subgraphs of the form $[\![x]\!]_U$ into vertex x and all subgraphs of the form $[\![x;y]\!]$ into edge xy.

The following proposition is a key element in the developments to come. It makes it possible to extract a term out of a graph whose input and output coincide, by providing ways to chose an element where to relocate the output and resort to the easier case when input and output differ. (Note that G is still assumed to be connected.)

▶ **Proposition 20.** *Assume $G = H\backslash\iota$, for some K_4-free simple graph H and some vertex ι. Further assume that U is the set of neighbours of ι in H and that this set is not empty.*

(i) *$\mathsf{CP}(U)$ is a tree,*
(ii) *for every edge xy in $\mathsf{CP}(U)$, the graph $[\![x;y]\!] + xy$ is K_4-free,*
(iii) *for every vertex x in $\mathsf{CP}(U)$, the graph $H + \iota x$ is K_4-free.*

As a consequence of the above proposition, we have the following one, which makes it possible to decompose graphs with distinct input and output into a parallel composition when they cannot be a series composition.

▶ **Proposition 21.** *Let ι, o be two distinct vertices such that $G + \iota o$ is K_4-free. We have that:*

(i) *if ι and o are not adjacent in G and $\iota \Diamond o$, then the graph induced by $[\![\iota; o]\!]$ has at least two connected components.*

(ii) *for every edge xy in $\mathsf{CP}(\{\iota, o\})$, the graph $[\![x; y]\!] + xy$ is K_4-free,*

6 Extracting terms

Now we have enough preliminary material and we can look for a right inverse to the function $\mathsf{g} : \mathsf{Trm} \to \mathsf{TW}_2$. As explained in the Introduction, we use K_4-freeness to extract terms from graphs in a more structured way than using tree decompositions directly.

▶ **Definition 22.** The *skeleton* of a graph G is the simple graph S obtained from G by forgetting input, output, labelling, edge directions, edge multiplicities, and self-loops. The *strong skeleton* of G is $S + \iota o$ if $\iota \neq o$, and S otherwise.

As an example, the strong skeleton of any instance of the graph (M_3) from Section 4 is K_4. More generally, a graph belongs to TW_2 if and only if its strong skeleton has treewidth at most two in the standard sense.

▶ **Proposition 23.** *The strong skeleton of every graph in TW_2 is K_4-free.*

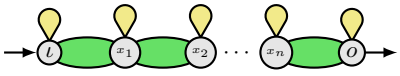
Given a graph G and two vertices x, y, we write $G[\![x; y]\!]$ for the subgraph of G induced by the set $[\![x; y]\!]$ (computed in the skeleton of G), with input and output respectively set to x and y, and with self-loops on x and y removed. The strong skeleton of $G[\![x; y]\!]$ is $[\![x; y]\!] + xy$.

Similarly, given a graph G, a set U of vertices and vertex x, we write $G[\![x]\!]_U$ for the subgraph of G induced by the set $[\![x]\!]_U$ (computed in the skeleton of G), with both input and output set to x. Doing so, the skeleton and strong skeleton of $G[\![x]\!]_U$ are both $[\![x]\!]_U$. We shall omit the subscript when it is clear from the context.

▶ **Definition 24.** The term $\mathsf{t}(G)$ of a graph G whose strong skeleton is K_4-free is defined by induction on the number of edges in G[2]. When G is connected there are two main cases depending on whether the input and output coincide *(a)* or not *(b)*. We deal with the general case *(c)* by decomposing the graph into connected components.

(a) Connected, distinct input and output

Consider the line graph (Lemma 14) obtained by taking the checkpoint graph of $U = \{\iota, o\}$ in the skeleton of G. Write it as $x_0 \dots x_{n+1}$ with $\iota = x_0$ and $o = x_{n+1}$. According to Proposition 19, G looks as follows.



We set $\mathsf{t}(G) \triangleq \mathsf{t}(G[\![x_0]\!]) \cdot \mathsf{t}(G[\![x_0; x_1]\!]) \cdot \mathsf{t}(G[\![x_1]\!]) \cdot \ldots \cdot \mathsf{t}(G[\![x_n]\!]) \cdot \mathsf{t}(G[\![x_n; x_{n+1}]\!]) \cdot \mathsf{t}(G[\![x_{n+1}]\!])$

The (strong) skeleton of each graph $G[\![x_i]\!]$ is just $[\![x_i]\!]$, which is necessarily K_4-free, as a subgraph of that of G. Proposition 21(2) moreover ensures that so are the strong skeletons of all graphs $G[\![x_i; x_{i+1}]\!]$. The above recursive calls occur on smaller graphs unless $n = 0$ and the graphs $G[\![\iota]\!]$ and $G[\![o]\!]$ are reduced to the trivial graph with one vertex and no edge (i.e., the graph 1). In such a situation,

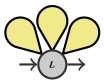
[2] More precisely, on the lexicographic product of the number of edges and the textual precedence of the three considered cases.

- either ι and o are adjacent in G. Then let G' be the graph obtained by removing from G all edges between ι and o and let $u = a_1 \parallel \ldots \parallel a_i \parallel b_1^\circ \parallel \ldots b_j^\circ$ be a term corresponding to those edges. Accordingly, we set $\mathsf{t}(G) \triangleq \mathsf{t}(G') \parallel u$.
- Or they are not, and Proposition 21(2) applies so that we can decompose G into parallel components: $G = G_1 \parallel \ldots \parallel G_m$ with $m \geq 2$. We set $\mathsf{t}(G) \triangleq \mathsf{t}(G_1) \parallel \ldots \parallel \mathsf{t}(G_m)$.

(b) Connected, input equals output

If there are self-loops on ι, let $u = a_1 \parallel \ldots \parallel a_n$ be a term corresponding to those edges, let G' be the graph obtained by removing them, and recursively set $\mathsf{t}(G) \triangleq \mathsf{t}(G') \parallel u$.

Otherwise let H be the skeleton of G. Decompose $H\backslash\iota$ into connected components $H_1\backslash\iota, \ldots, H_m\backslash\iota$ such that $H \simeq H_1 \cup \cdots \cup H_m$. The graph looks as follows.



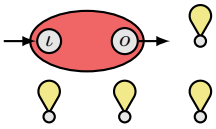
If $m = 0$, then set $\mathsf{t}(G) = 1$. If $m > 1$, set $\mathsf{t}(G) \triangleq \|_{i \leq m} \mathsf{t}(G_i)$, where G_i is the subgraph of G induced by H_i. It remains to cover the case where $m = 1$. Let U be the set of neighbours of the input and compute the checkpoint graph $\mathsf{CP}(U)$ in $H\backslash\iota$. Pick an arbitrary node $x \in \mathsf{CP}(U)$. By Proposition 20(3), the strong skeleton of $G[\iota; x]$ is K_4-free. Set $\mathsf{t}(G) \triangleq \mathrm{dom}(\mathsf{t}(G[\iota; x]))$. (Remember that $\mathrm{dom}(\cdot)$ relocates the output to the input.)

(c) General case

Decompose the graph G into connected components $G_1, \ldots, G_n$. For all $i \leq n$, pick an arbitrary vertex x_i in the component G_i. There are two cases:

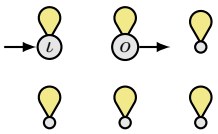
- either input and output belong to the same component, say G_j; then set

$$\mathsf{t}(G) \triangleq \mathsf{t}(G_j) \parallel \Big\|_{i \neq j} \top \cdot \mathsf{t}(G_i[x_i]) \cdot \top$$



- or they belong to two distinct components, say ι in G_j and o in G_k, in which case we set

$$\mathsf{t}(G) \triangleq \mathsf{t}(G_j[\iota]) \cdot \top \cdot \mathsf{t}(G_k[o]) \parallel \Big\|_{i \neq j,k} \top \cdot \mathsf{t}(G_i[x_i]) \cdot \top$$



In both cases, it is easy to check that the recursive calls occur on graphs whose (strong) skeletons are subgraphs of the strong skeleton of G, and thus K_4-free.

The definition of the extraction function t ends here. This function is defined on "concrete" graphs: we need to choose some vertices in cases *(b)* and *(c)*, and we can only do so by relying on the concrete identity of those vertices (e.g., choosing the smallest one, assuming they are numbers). We shall see in the following section that all those potential choices, nevertheless, always lead to congruent terms (Theorem 30). By construction, we obtain:

▶ **Theorem 25.** *For every graph $G \in \mathsf{TW}_2$, $\mathsf{g}(\mathsf{t}(G)) \simeq G$.*

▶ **Corollary 26.** *The following are equivalent for all graphs G:*

(i) *G has treewidth at most two;*
(ii) *the strong skeleton of G is K_4-free;*
(iii) *G is (isomorphic to) the graph of a term.*

▶ Remark 27. When G is connected, $\mathsf{t}(G)$ does not contain occurrences of $\top$ other than those that are implicit in our uses of $\mathrm{dom}(\cdot)$ in case *(b)*. Thus we obtain an alternative proof of Dougherty and Gutiérrez' characterisation [17, Section 4, Theorem 31] (their minor exclusion property is easily proved equivalent to ours—they do not mention treewidth).

Also note that we can easily avoid using 1 (but not $\mathrm{dom}(\cdot)$) when the graph does not contain self-loops and is not reduced to the trivial graph 1. When the graph does not contain self-loops and has distinct input and output, the construction can be modifed to produce terms without both 1 and $\mathrm{dom}(\cdot)$; the resulting construction becomes, however, less local, and we do not know how to use it to axiomatise the 1-free reduct of 2p-algebra.

7 Completeness of the axioms

We can finally prove that the axioms of 2p-algebras are complete w.r.t. graphs: they suffice to equate all terms denoting the same graph up to isomorphism. For lack of space, we present only the main steps. Proofs for this last part consist in detailed analyses of the term extraction function (t) through inductive arguments following its recursive definition, and using the laws from Proposition 1 to relate the extracted terms. All details are in [8].

We first prove that t maps isomorphic graphs to congruent terms. We need for that the following propositions.

▶ **Proposition 28.** *Let $G \in \mathsf{TW}_2$ be a graph with $\iota = o$, without self-loops on ι. Let S be its skeleton, and assume that $S\backslash\iota$ is connected. Let U be the neighbours of ι in G and consider the checkpoint graph of U in the skeleton of $S\backslash\iota$. For all checkpoints x, y, we have* $\mathrm{dom}(\mathsf{t}(G[\iota, x])) \equiv \mathrm{dom}(\mathsf{t}(G[\iota, y]))$.

▶ **Proposition 29.** *Let $G \in \mathsf{TW}_2$ be a connected graph. For all vertices x, y, we have* $\top\mathsf{t}(G[x])\top \equiv \top\mathsf{t}(G[y])\top$.

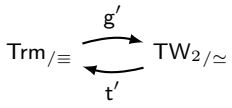
▶ **Theorem 30.** *Let $G, H \in \mathsf{TW}_2$ be two graphs. If $G \simeq H$ then* $\mathsf{t}(G) \equiv \mathsf{t}(H)$.

In other words, the extraction function t yields a function $\mathsf{t}' : \mathsf{TW}_{2/\simeq} \to \mathsf{Trm}_{/\equiv}$ between 2p-algebras. We finally prove that t' is a homomorphism, and, in fact, an isomorphism.

▶ **Proposition 31.** *The function $\mathsf{t}' : \mathsf{TW}_{2/\simeq} \to \mathsf{Trm}_{/\equiv}$ is an homomorphism of 2p-algebras.*

▶ **Theorem 32.** *For every term u, we have* $\mathsf{t}(\mathsf{g}(u)) \equiv u$.

▶ **Corollary 33.** *For all terms u and v, we have $u \equiv v$ if and only if $\mathsf{g}(u) \simeq \mathsf{g}(v)$. Graphs of treewidth at most two form the free 2p-algebra, as witnessed by the diagram on the right.*



8 Future work

What is the free idempotent 2p-algebra? (Where parallel composition is idempotent.) One could be tempted to switch to simple directed graphs, where there is at most one edge with a given label from one vertex to another. This is however not an option: the graphs of $ab \parallel ab$ and ab are not isomorphic. One could also consider equivalences on graphs that are weaker than isomorphism. The notion of (two-way) bisimilarity [25, 24] that come to mind does not work either: such an equivalence relation on graphs certainly validates idempotency of

parallel composition, but it also introduces new laws, e.g., $\top(1 \parallel aa)\top = \top(1 \parallel a)\top$, which are not even true in algebras of binary relations.

Courcelle used the algebraic theory he defined with Bauderon for arbitrary graphs [3] to propose a notion of graph recognisability [9], based on the generic framework by Mezei and Wright [23]. He proved that sets of graphs definable in MSO are recognisable. The converse does not hold in general. He later proved it for graphs of treewidth at most two [10] with a *counting* variant of MSO, conjecturing that it would be so for classes of graphs of bounded treewidth. This conjecture was proved only last year, by Bojańczyk and Pilipczuk [6].

The present work makes it possible to propose an alternative notion of recognisability for treewidth at most two, *2p-recognisability*: recognisability by a finite 2p-algebra. We conjecture that this notion coincides with recognisability. That recognisability entails 2p-recognisability is easy. The converse is harder; it amounts to proving that any finite congruence with respect to substitutions in treewidth at most two graphs can be refined into a finite congruence with respect to substitutions in arbitrary graphs. We see two ways of attaining this implication:

1. prove that 2p-recognisability entails MSO-definability, which could possibly be done along the lines of [10], by showing that our term extraction procedure is MSO-definable.
2. or use a slight generalisation of the result by Courcelle and Lagergren [14], relating recognisability to *k-recognisability* for graphs of treewidth at most k. Indeed, 2*p*-recognisability is really close to 2-recognisability. Unfortunately, Courcelle and Lagergren's result is established only for unlabelled, undirected graphs, without sources, while we need labelled directed graphs with two sources.

One can easily extend our syntax to cover graphs of treewidth at most k, with k sources, for a given k (see, e.g., [15, 2]). However, we do not know how to generate finite axiomatisations in a systematic way, for every such k. Moreover, our proof strategy heavily depends on the fact that when $k = 2$, K_4 is the only excluded minor. We would need another strategy to deal with the general case since the excluded minors are not known for $k \geq 4$.

References

1 H. Andréka and D. A. Bredikhin. The equational theory of union-free algebras of relations. *Algebra Universalis*, 33(4):516–532, 1995. doi:10.1007/BF01225472.

2 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.

3 Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20(2-3):83–127, 1987. doi:10.1007/BF01692060.

4 Ulrich Berger and Helmut Schwichtenberg. An inverse of the evaluation functional for typed lambda-calculus. In *LICS*, pages 203–211. IEEE, 1991. doi:10.1109/LICS.1991.151645.

5 H.L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

6 Mikołaj Bojańczyk and Michal Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In *LICS*, pages 407–416. ACM, 2016. doi:10.1145/2933575.2934508.

7 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.

8 Enric Cosme-Llópez and Damien Pous. K_4-free graphs as a free algebra, 2017. Full version of this extended abstract, with all proofs, available at https://hal.archives-ouvertes.fr/hal-01515752/. URL: https://hal.archives-ouvertes.fr/hal-01515752/.

9 B. Courcelle. The monadic second-order logic of graphs. I: Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

10 B. Courcelle. The monadic second-order logic of graphs V: on closing the gap between definability and recognizability. *Theoretical Computer Science*, 80(2):153–202, 1991. doi:10.1016/0304-3975(91)90387-H.

11 B. Courcelle. Recognizable sets of graphs: equivalent definitions and closure properties. *Mathematical Structures in Computer Science*, 4(1):1–32, 1994.

12 B. Courcelle. The monadic second-order logic of graphs XI: Hierarchical decompositions of connected graphs. *Theoretical Computer Science*, 224(1):35–58, 1999. doi:10.1016/S0304-3975(98)00306-5.

13 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 University Press, 2012.

14 B. Courcelle and J. Lagergren. Equivalent definitions of recognizability for sets of graphs of bounded tree-width. *Mathematical Structures in Computer Science*, 6(2):141–165, 1996. doi:10.1017/S096012950000092X.

15 Bruno Courcelle. Graph grammars, monadic second-order logic and the theory of graph minors. In *Graph Structure Theory*, volume 147 of *Contemporary Mathematics*, pages 565–590. American Mathematical Society, 1993. Proceedings of a Joint Summer Research Conference on Graph Minors held June 22 to July 5, 1991, at the University of Washington, Seattle. doi:10.1090/conm/147.

16 R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2005.

17 Daniel J. Dougherty and Claudio Gutiérrez. Normal forms for binary relations. *Theoretical Computer Science*, 360(1-3):228–246, 2006. doi:10.1016/j.tcs.2006.03.023.

18 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.

19 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.

20 P.J. Freyd and A. Scedrov. *Categories, Allegories*. North Holland. Elsevier, 1990. URL: https://books.google.fr/books?id=fCSJRegkKdoC.

21 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.

22 D. Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997. doi:10.1145/256167.256195.

23 J. Mezei and J.B. Wright. Algebraic automata and context-free sets. *Information and Control*, 11(1–2):3–29, 1967. doi:10.1016/S0019-9958(67)90353-1.

24 R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

25 David Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, pages 167–183, 1981. URL: http://dl.acm.org/citation.cfm?id=647210.720030.

26 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.

27 W. Tutte. *Graph Theory*. Addison-Wesley, Reading, MA, 1984.

Making Metric Temporal Logic Rational*

Shankara Narayanan Krishna[1], Khushraj Madnani[1], and Paritosh K. Pandya[3]

1 **IIT Bombay, Mumbai, India**
krishnas@cse.iitb.ac.in
2 **IIT Bombay, Mumbai, India**
khushraj@cse.iitb.ac.in
3 **Tata Institute of Fundamental Research, Mumbai, India**
pandya@tifr.res.in

Abstract

We study an extension of MTL in pointwise time with regular expression guarded modality $\mathsf{Rat}_I(\mathsf{re})$ where re is a rational expression over subformulae. We study the decidability and expressiveness of this extension (MTL+ URat+Rat), called RatMTL, as well as its fragment SfrMTL where only star-free rational expressions are allowed. Using the technique of temporal projections, we show that RatMTL has decidable satisfiability by giving an equisatisfiable reduction to MTL. We also identify a subclass MITL + URat of RatMTL for which our equisatisfiable reduction gives rise to formulae of MITL, yielding elementary decidability. As our second main result, we show a tight automaton-logic connection between SfrMTL and partially ordered (or very weak) 1-clock alternating timed automata.

1998 ACM Subject Classification F.4.1. Mathematical Logic

Keywords and phrases Metric Temporal Logic, Timed Automata, Regular Expression, Equisatisfiability, Expressiveness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.77

1 Introduction

Temporal logics provide constructs to specify qualitative ordering between events in time. Real time logics are quantitative extensions of temporal logics with the ability to specify real time constraints amongst events. Logics MTL and TPTL are amongst the prominent real time logics [2]. Two notions of MTL semantics have been studied in the literature : continuous and pointwise [5]. The expressiveness and decidability results vary considerably with the semantics used : while the satisfiability checking of MTL is undecidable in the continuous semantics even for finite timed words [1], it is decidable in pointwise semantics with non-primitive recursive complexity over finite timed words [15]. The satisfiability checking over infinite timed words is undecidable for both the semantics. Due to the hardness of analysis, quest for a decidable subclass and extension was started.

Related Work. Due to limited expressive power of MTL, several additional modalities have been proposed : the threshold counting modality [16] $\mathsf{C}_I^{\geq n}\phi$ states that in time interval I relative to current point, ϕ occurs at least n times. Note that we represent the set of modalities C_I is represented by C. The Pnueli modality [16] $\mathsf{Pn}_I(\phi_1, \ldots, \phi_n)$ states that there

* Please refer url <http://arxiv.org/abs/1705.01501> for full version

is a subsequence of n time points inside interval I where at i^{th} point the formula ϕ_i holds. In a recent result, Hunter [10] showed that, in continuous time semantics, MTL enriched with C modality (denoted MTL + C) is as expressive as FO[<, +1], which is as expressive as TPTL. Unfortunately, satisfiability and model checking of all these logics are undecidable. This has led us to focus on the pointwise case with only the until modality, i.e. logic MTL[U_I], which we abbreviate as MTL in rest of the paper.Also, MTL + *op* means MTL with modalities U_I as well as *op*.

In pointwise semantics, it can be shown that MTL+C is strictly more expressive than MTL and remains decidable for finite words (see [12]). In this paper, we propose a generalization of threshold counting and Pnueli modalities by a rational expression modality $\mathsf{Rat}_I\mathsf{re}(\phi_1, \ldots, \phi_k)$, which specifies that the truth of the subformulae, $\phi_1, \ldots, \phi_k$, at the set of points within interval I is in accordance with the regular expression $\mathsf{re}(\phi_1, \ldots, \phi_k)$. The resulting logic is called RatMTL and is the subject of this paper. The inability to specify rational expression constraints has been an important lacuna of LTL and its practically useful extensions such as PSL sugar [7], [6] (based on Dynamic Logic [8]) which extend LTL with both counting and rational expressions were studied. This indicates that our logic RatMTL is a natural and useful logic for specifying properties. Adding timing constraints to regular expressions was first given by Asarin, Caspi and Maler in [3] and was called as Timed Regular Expressions. They also show that these expressions exactly characterize the expressive power of Timed Automata. But this equivalence relies indispensably on the addition of renaming operation within there syntax [9] and are not closed under negations. In fact the validity checking for this extension was undecidable. Thus we propose a boolean closed decidable logic which can express regular expressions along with timing constraints. To our knowledge, impact of rational expression constraints on metric temporal modalities have not been studied before. The expressive power of logic RatMTL raises several points of interest.

As our first main result, we show that satisfiability of RatMTL is decidable by giving an equisatisfiable reduction to MTL. The reduction makes use of the technique of *oversampled temporal projections* which was previously proposed [11], [12] and used for proving the decidability of MTL + C. The reduction given here has several novel features such as an MTL encoding of the run tree of an alternating automaton which restarts the DFA of a given rational expression at each time point (section 3.1). We identify two syntactic subsets of RatMTL, the first denoted as MITL + URat with 2EXPSPACE easy satisfiability, and its further subset MITL+UM with EXPSPACE-complete satisfiability. As our second main result, we show that the star-free fragment SfrMTL of RatMTL characterizes exactly the class of partially ordered 1-clock alternating timed automata, thereby giving a tight logic automaton connection. The most non-trivial part of this proof is the construction of SfrMTL formula equivalent to a given partially ordered 1-clock alternating timed automaton $\mathcal{A}$ (Lemma 11).

2 Timed Temporal Logics

This section describes the syntax and semantics of the timed temporal logics needed in this paper : MTL and TPTL. Let Σ be a finite set of propositions. A finite timed word over Σ is a tuple $\rho = (\sigma, \tau)$. σ and τ are sequences $\sigma_1\sigma_2 \ldots \sigma_n$ and $\tau_1\tau_2 \ldots \tau_n$ respectively, with $\sigma_i \in \mathcal{P}(\Sigma) - \emptyset$, and $\tau_i \in \mathbb{R}_{\geq 0}$ for $1 \leq i \leq n$ and $\forall i \in dom(\rho)$, $\tau_i \leq \tau_{i+1}$, where $dom(\rho)$ is the set of positions $\{1, 2, \ldots, n\}$ in the timed word. For convenience, we assume $\tau_1 = 0$. The σ_i's can be thought of as labelling positions i in $dom(\rho)$. For example, given $\Sigma = \{a, b, c\}$, $\rho = (\{a, c\}, 0)(\{a\}, 0.7)(\{b\}, 1.1)$ is a timed word. ρ is strictly monotonic iff $\tau_i < \tau_{i+1}$ for all $i, i + 1 \in dom(\rho)$. Otherwise, it is weakly monotonic. The set of finite timed words

over Σ is denoted $T\Sigma^*$. Given $\rho = (\sigma, \tau)$ with $\sigma = \sigma_1 \dots \sigma_n$, σ^{single} denotes the set of words $\{w_1 w_2 \dots w_n \mid w_i \in \sigma_i\}$. For ρ as above, σ^{single} consists of $(\{a\}, 0)(\{a\}, 0.7)(\{b\}, 1.1)$ and $(\{c\}, 0)(\{a\}, 0.7)(\{b\}, 1.1)$. Let $I\nu$ be a set of open, half-open or closed time intervals. The end points of these intervals are in $\mathbb{N} \cup \{0, \infty\}$. For example, $[1, 3), [2, \infty)$. For $\tau \in \mathbb{R}_{\geq 0}$ and interval $\langle a, b\rangle$, with $<\in \{(, [\}$ and $>\in \{],)\}$, $\tau + \langle a, b\rangle$ stands for the interval $\langle \tau + a, \tau + b\rangle$.

Metric Temporal Logic (MTL). Given a finite alphabet Σ, the formulae of MTL are built from Σ using boolean connectives and time constrained version of the modality U as follows: $\varphi ::= a(\in \Sigma) \mid true \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi\, \mathsf{U}_I \varphi$, where $I \in I\nu$. For a timed word $\rho = (\sigma, \tau) \in T\Sigma^*$, a position $i \in dom(\rho)$, and an MTL formula φ, the satisfaction of φ at a position i of ρ is denoted $(\rho, i) \models \varphi$, and is defined as follows: (i) $\rho, i \models a \leftrightarrow a \in \sigma_i$, (ii) $\rho, i \models \neg\varphi \leftrightarrow \rho, i \not\models \varphi$, (iii) $\rho, i \models \varphi_1 \wedge \varphi_2 \leftrightarrow \rho, i \models \varphi_1$ and $\rho, i \models \varphi_2$, (iv) $\rho, i \models \varphi_1\, \mathsf{U}_I \varphi_2 \leftrightarrow \exists j > i$, $\rho, j \models \varphi_2, \tau_j - \tau_i \in I$, and $\rho, k \models \varphi_1\ \forall\ i < k < j$.

The language of a MTL formula φ is $L(\varphi) = \{\rho \mid \rho, 1 \models \varphi\}$. Two formulae φ and ϕ are said to be equivalent denoted as $\varphi \equiv \phi$ iff $L(\varphi) = L(\phi)$. Additional temporal connectives are defined in the standard way: we have the constrained future eventuality operator $\Diamond_I a \equiv true\, \mathsf{U}_I a$ and its dual $\Box_I a \equiv \neg\Diamond_I \neg a$. We also define the next operator as $\mathsf{O}_I \phi \equiv \bot\, \mathsf{U}_I \phi$. Non-strict versions of operators are defined as $\Diamond_I^{\mathsf{ns}} a = a \vee \Diamond_I a, \Box_I^{\mathsf{ns}} a \equiv a \wedge \Box_I a$, $a\, \mathsf{U}_I^{\mathsf{ns}} b \equiv b \vee [a \wedge (a\, \mathsf{U}_I b)]$ if $0 \in I$, and $[a \wedge (a\, \mathsf{U}_I b)]$ if $0 \notin I$. Also, $a\, \mathsf{W} b$ is a shorthand for $\Box a \vee (a\, \mathsf{U} b)$. The subclass of MTL obtained by restricting the intervals I in the until modality to non-punctual intervals is denoted MITL.

Timed Propositional Temporal Logic (TPTL). TPTL is a prominent real time extension of LTL, where timing constraints are specified with the help of freeze clocks. The set of TPTL formulas are defined inductively as $\varphi ::= a(\in \Sigma) \mid true \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi\, \mathsf{U} \varphi \mid y.\varphi \mid y \in I$. $\mathcal{C}$ is a set of clock variables progressing at the same rate, $y \in \mathcal{C}$, and I is an interval as above. For a timed word $\rho = (\sigma_1, \tau_1) \dots (\sigma_n, \tau_n)$, we define the satisfiability relation, $\rho, i, \nu \models \phi$ saying that the formula ϕ is true at position i of the timed word ρ with valuation ν of all the clock variables as follows: (1) $\rho, i, \nu \models a \leftrightarrow a \in \sigma_i$, (2) $\rho, i, \nu \models \neg\varphi \leftrightarrow \rho, i, \nu \not\models \varphi$, (3) $\rho, i, \nu \models \varphi_1 \wedge \varphi_2 \leftrightarrow \rho, i, \nu \models \varphi_1$ and $\rho, i, \nu \models \varphi_2$, (4) $\rho, i, \nu \models x.\varphi \leftrightarrow \rho, i, \nu[x \leftarrow \tau_i] \models \varphi$, (5) $\rho, i, \nu \models x \in I \leftrightarrow \tau_i - \nu(x) \in I$, (6) $\rho, i, \nu \models \varphi_1\, \mathsf{U} \varphi_2 \leftrightarrow \exists j > i$, $\rho, j, \nu \models \varphi_2$, and $\rho, k, \nu \models \varphi_1\ \forall\ i < k < j$. ρ satisfies ϕ denoted $\rho \models \phi$ iff $\rho, 1, \bar{0} \models \phi$. Here $\bar{0}$ is the valuation obtained by setting all clock variables to 0. We denote by $k-$TPTL the fragment of TPTL using at most k clock variables.

▶ **Theorem 1** ([15]). MTL *satisfiability is decidable over finite timed words and is non-primitive recursive.*

MTL with Rational Expressions (RatMTL)

We propose an extension of MTL with rational expressions, that forms the core of the paper. These modalities can assert the truth of a rational expression (over subformulae) within a particular time interval with respect to the present point. For example, $\mathsf{Rat}_{(0,1)}(\varphi_1.\varphi_2)^+$ when evaluated at a point i, asserts the existence of $2k$ points $\tau_i < \tau_{i+1} < \tau_{i+2} < \dots < \tau_{i+2k} < \tau_i + 1$, $k > 0$, such that φ_1 evaluates to true at τ_{i+2j+1}, and φ_2 evaluates to true at τ_{i+2j+2}, for all $0 \leq j < k$.

RatMTL Syntax Formulae of RatMTL are built from Σ (atomic propositions) as follows: $\varphi ::= a(\in \Sigma) \mid true \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathsf{Rat}_I \mathsf{re}(\mathsf{S}) \mid \varphi \mathsf{U} \mathsf{Rat}_{I,\mathsf{re}(\mathsf{S})} \varphi$, where $I \in I\nu$ and S is a finite set of formulae of interest generated by this grammar, and $\mathsf{re}(\mathsf{S})$ is defined as a

rational expression over S. $\mathsf{re(S)} ::= \varphi(\in \mathsf{S}) \mid \mathsf{re(S).re(S)} \mid \mathsf{re(S)} + \mathsf{re(S)} \mid [\mathsf{re(S)}]^*$. Thus, RatMTL is MTL + URat + Rat. An *atomic* rational expression re is any well-formed formula $\varphi \in \mathsf{RatMTL}$.

RatMTL Semantics For a timed word $\rho = (\sigma, \tau) \in T\Sigma^*$, a position $i \in dom(\rho)$, and a RatMTL formula φ, a finite set S of formulae, we define the satisfaction of φ at a position i as follows. For positions $i < j \in dom(\rho)$, let $\mathsf{Seg}(\rho, \mathsf{S}, i, j)$ denote the untimed word over $\mathcal{P}(\mathsf{S})$ obtained by marking the positions $k \in \{i+1, \ldots, j-1\}$ of ρ with $\psi \in \mathsf{S}$ iff $\rho, k \models \psi$. For a position $i \in dom(\rho)$ and an interval I, let $\mathsf{TSeg}(\rho, S, I, i)$ denote the untimed word over $\mathcal{P}(\mathsf{S})$ obtained by marking all the positions k such that $\tau_k - \tau_i \in I$ of ρ with $\psi \in \mathsf{S}$ iff $\rho, k \models \psi$.

1. $\rho, i \models \varphi_1 \mathsf{URat}_{I,\mathsf{re(S)}} \varphi_2 \leftrightarrow \exists j > i,\ \rho, j \models \varphi_2, \tau_j - \tau_i \in I,\ \rho, k \models \varphi_1\ \forall i < k < j$ and, $[\mathsf{Seg}(\rho, \mathsf{S}, i, j)]^{\mathsf{single}} \cap L(\mathsf{re}(S)) \neq \emptyset$, where $L(\mathsf{re(S)})$ is the language of the rational expression re formed over the set S. The subclass of RatMTL using only the URat modality is denoted RatMTL[URat] or MTL + URat and if only non-punctual intervals are used, then it is denoted RatMITL[URat] or MITL + URat.
2. $\rho, i \models \mathsf{Rat}_I \mathsf{re} \leftrightarrow [\mathsf{TSeg}(\rho, S, I, i)]^{\mathsf{single}} \cap L(\mathsf{re}(S)) \neq \emptyset$.

The language accepted by a RatMTL formula φ is given by $L(\varphi) = \{\rho \mid \rho, 0 \models \varphi\}$.

▶ **Example 2.** Consider the formula $\varphi = a\mathsf{URat}_{(0,1),ab^*} b$. Then $\mathsf{re} = ab^*$, and the subformulae of interest are a, b. For $\rho = (\{a\}, 0)(\{a, b\}, 0.3)(\{a, b\}, 0.99)$, $\rho, 1 \models \varphi$, since $a \in \sigma_2, b \in \sigma_3$, $\tau_3 - \tau_1 \in (0, 1)$ and $a \in [\mathsf{Seg}(\rho, \{a, b\}, 1, 3)]^{\mathsf{single}} \cap L(ab^*)$. On the other hand, for the word $\rho = (\{a\}, 0)(\{a\}, 0.3)(\{a\}, 0.5)(\{a\}, 0.9)(\{b\}, 0.99)$, we know that $\rho, 1 \not\models \varphi$, since even though $b \in \sigma_5, a \in \sigma_i$ for $i < 5$, $[\mathsf{Seg}(\rho, \{a, b\}, 1, 5)]^{\mathsf{single}} = aaa$ and $aaa \notin L(ab^*)$.

▶ **Example 3.** Consider the formula $\varphi = \mathsf{Rat}_{(0,1)}[\mathsf{Rat}_{(0,1)} a]^*$.
For $\rho = (\{a, b\}, 0)(\{a, b\}, 0.7)(\{b\}, 0.98)(\{a, b\}, 1.4)$, we have $\rho, 1 \not\models \mathsf{Rat}_{(0,1)}[\mathsf{Rat}_{(0,1)} a]^*$, since point 2 is not marked $\mathsf{Rat}_{(0,1)} a$, even though point 3 is.

Generalizing Counting, Pnueli & Mod Counting Modalities. The following reductions show that RatMTL subsumes most of the extensions of MTL studied in the literature.

(1) Threshold Counting constraints [16], [13], [12] specify the number of times a property holds within some time region is at least (or at most) n. These can be expressed in RatMTL: (i) $\mathsf{C}_I^{\geq n} \varphi \equiv \mathsf{Rat}_I(\mathsf{re}_{th})$, (ii) $\phi_1 \mathsf{UT}_{I,\varphi \geq n} \phi_2 \equiv \phi_1 \mathsf{URat}_{I,\mathsf{re}_{th}} \phi_2$, where $\mathsf{re}_{th} = true^* \underbrace{\varphi.true^*. \ldots .\varphi.true^*}_{n \text{ times}}$.

(2) Pnueli Modalities[1] [16], which enhance the expressiveness of MITL in continuous semantics preserving the complexity, can be written in RatMTL: $\mathsf{Pn}_I(\phi_1, \phi_2, \ldots, \phi_k)$ can be written as $\mathsf{Rat}_I(true^*.\phi_1.true^*\phi_2. \ldots .true^*.\phi_k.true^*)$.

(3) Modulo Counting constraints [4], [14] specify the number of times a property holds modulo $n \in \mathbb{N}$, in some region. We extend these to the timed setting by proposing two modalities $\mathsf{MC}_I^{k\%n}$ and $\mathsf{UM}_{I,\varphi=k\%n}$. $\mathsf{MC}_I^{k\%n} \varphi$ checks if the number of times φ is true in interval I is $M(n) + k$, where $M(n)$ denotes a non-negative integer multiple of n, and $0 \leq k \leq n-1$, while $\varphi_1 \mathsf{UM}_{I,\#\psi=k\%n} \varphi_2$ when asserted at a point i, checks

[1] The version of the modality only specified sequences for the next unit interval. We talk about a more general version of this operator which is appended by timing interval.

the existence of $j > i$ such that $\tau_j - \tau_i \in I$, φ_2 is true at j, φ_1 holds between i, j, and the number of times ψ is true between i, j is $M(n) + k$, $0 \leq k \leq n-1$. As an example, $\psi = true\mathsf{UM}_{(0,1),\#b=1\%2}(a \vee b)$, when asserted at a point i, checks the existence of a point $j > i$ such that a or $b \in \sigma_j$, $\tau_j - \tau_i \in (0,1)$, and the number of points between i, j where b is true is odd. Both these modalities can be rewritten equivalently in RatMTL as follows: $\mathsf{MC}_I^{k\%n}\varphi \equiv \mathsf{Rat}_I(\mathsf{re}_{mod})$ and $\phi_1\mathsf{UM}_{I,\varphi=k\%n}\phi_2 \equiv \phi_1\mathsf{URat}_{I,\mathsf{re}_{mod}}\phi_2$ where $\mathsf{re}_{mod} = ([\underbrace{(\neg\varphi)^*.\varphi.\ldots.(\neg\varphi)^*.\varphi}_{n \text{ times}}]^*.[\underbrace{(\neg\varphi)^*.\varphi.\ldots.(\neg\varphi)^*.\varphi}_{k \text{ times}}]$. The extension of MTL (MITL) with only UM is denoted MTL + UM (MITL + UM) while MTL + MC (MITL + MC) denotes the extension using MC.

3 Satisfiability of RatMTL and Complexity

The main results of this section are as follows.

▶ **Theorem 4.** *(1) Satisfiability of* RatMTL *is decidable over finite timed words. (2) Satisfiability of* MITL + UM *is* EXPSPACE*-complete. (3) Satisfiability of* MITL + URat *is within* 2EXPSPACE*. (4) Satisfiability of* MITL + MC *is* $\mathbf{F}_{\omega^\omega}$*-hard.*

Details of 4.2, 4.3, 4.4 are in appendices E.2,E.3 and E.4 of the full version, respectively.

▶ **Theorem 5.** MTL + URat $\subseteq$ MTL + Rat, MTL + UM $\subseteq$ MTL + MC.

Theorem 5 shows that the Rat modality can capture URat (and likewise, MC captures UM). Thus, RatMTL $\equiv$ MTL + Rat. Observe that any re can be decomposed into finitely many factors, i.e. $\mathsf{re} = \sum_{i=1}^{n} R_1^i.R_2^i$. Given $true\mathsf{URat}_{[l,u),\mathsf{re}}\phi_2$, we assert R_1^i within interval $(0,l]$ and R_2^i in the prefix of the latter part within $[l,u)$, followed by ϕ_2. $true\mathsf{URat}_{[l,u),\mathsf{re}}\phi_2 \equiv \bigvee_{i\in\{1,2\ldots,n\}} \mathsf{Rat}_{(0,l)}R_1^i \wedge \mathsf{Rat}_{[l,u)}R_2^i.\phi_2.\Sigma^*$. The proofs are in appendix G of the full version.

3.1 Proof of Theorem 4.1

Equisatisfiability. We will use the technique of equisatisfiability modulo oversampling [11] in the proof of Theorem 4. Using this technique, formulae φ in one logic (say RatMTL) can be transformed into formulae ψ over a simpler logic (say MTL) such that whenever $\rho \models \varphi$ for a timed word ρ over alphabet Σ, one can construct a timed word ρ' over an extended set of positions and an extended alphabet Σ' such that $\rho' \models \psi$ and vice-versa [11], [12]. In *oversampling*, (i) $dom(\rho')$ is extended by adding some extra positions between the first and last point of ρ, (ii) the labeling of a position $i \in dom(\rho)$ is over the extended alphabet $\Sigma' \supset \Sigma$ and can be a superset of the previous labeling over Σ, while the new positions are labeled using only the new symbols $\Sigma' - \Sigma$. We can recover ρ from ρ' by erasing the new points and the new symbols. A restricted use of oversampling, when one only extends the alphabet and not the set of positions of a timed word ρ is called *simple extension*. In this case, if ρ' is a simple extension of ρ, then $dom(\rho) = dom(\rho')$, and by erasing the new symbols from ρ', we obtain ρ. See Figure 1 for an illustration. The formula ψ over the larger alphabet $\Sigma' \supset \Sigma$ such that $\rho' \models \psi$ iff $\rho \models \varphi$ is said to be equisatisfiable modulo temporal projections to φ. In particular, ψ is equisatisfiable to φ modulo simple extensions or modulo oversampling, depending on how the word ρ' is constructed from the word ρ.

The oversampling technique is used in the proofs of parts 4.1, 4.3 and 4.4.

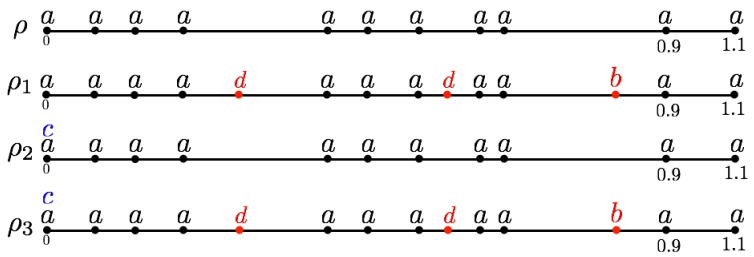


Figure 1 ρ is over $\Sigma = \{a\}$ and satisfies $\varphi = \Box_{(0,1)}a$. ρ_1 is an oversampling of ρ over an extended alphabet $\Sigma_1 = \Sigma \cup \{b, d\}$ and satisfies $\psi_1 = \Box(b \leftrightarrow \neg a) \wedge (\neg b \,\mathsf{U}_{(0,1)} b)$. The red points in ρ_1 are the oversampling points. ρ_2 is a simple extension of ρ over an extended alphabet $\Sigma_2 = \Sigma \cup \{c\}$ and satisfies $\psi_2 = \Box(c \leftrightarrow \Box_{(0,1)}a) \wedge c$. It can be seen that ψ_1 is equivalent to φ modulo oversampling, and ψ_2 is equivalent to φ modulo simple extensions using the (respectively oversampling, simple) extensions ρ_1, ρ_2 of ρ. However, ρ_3 above, obtained by merging ρ_1, ρ_2, eventhough an oversampling of ρ, is not a good model for the formula $\psi_1 \wedge \psi_2$ over $\Sigma_1 \cup \Sigma_2$. However, we can relativize ψ_1 and ψ_2 with respect to Σ as $\Box(act_1 \rightarrow (b \leftrightarrow \neg a)) \wedge [(act_1 \rightarrow \neg b) \,\mathsf{U}_{(0,1)} (b \wedge act_1)]$, and $\Box(act_2 \rightarrow (c \leftrightarrow \Box_{[0,1)}(act_2 \rightarrow a))) \wedge (act_2 \wedge c)$ where $act_1 = \bigvee \Sigma_1, act_2 = \bigvee \Sigma_2$. The relativized formula $\kappa = Rel(\psi_1, \Sigma) \wedge Rel(\psi_2, \Sigma)$ is then equisatisfiable to φ modulo oversampling, and ρ_3 is indeed an oversampling of ρ satisfying κ. This shows that while combining formulae ψ_1, ψ_2 which are equivalent to formulae φ_1, φ_2 modulo oversampling, we need to relativize ψ_1, ψ_2 to obtain a conjunction which will be equisatisfiable to $\varphi_1 \wedge \varphi_2$ modulo oversampling. See [11] for details.

Equisatisfiable Reduction : RatMTL to MTL

Let φ be a RatMTL formula. To obtain equisatisfiable MTL formula ψ, we do the following.

1. We "flatten" the rational(Rat & URat) modalities to simplify the formulae, eliminating nested rational modalities by allotting witness variable for each rational subformulae . Thus the resulting formulae will be of the form $\mathsf{prop} \wedge \Box^{\mathsf{ns}}[w_1 \leftrightarrow \mathsf{Rat}_I, \mathsf{URat}] \cdots \wedge \Box^{\mathsf{ns}}[w_k \leftrightarrow \mathsf{Rat}_I, \mathsf{URat}]$ where prop refers to some boolean formulae over atoms and $\mathsf{Rat}_I, \mathsf{URat}$ denotes formulae of the form $\mathsf{Rat}_I \mathsf{re{-}atom}$, $\mathsf{propURat}_{I,\mathsf{re-atom}}\mathsf{prop}$, respectively. Each conjunct of the form $\Box^{\mathsf{ns}}[w_1 \leftrightarrow \mathsf{Rat}_I, \mathsf{URat}]$ is called as *temporal definition*.
2. The elimination of rational modalities is achieved by obtaining equisatisfiable MTL formulae ψ_i over X_i, possibly a larger set of propositions than $\Sigma \cup W_i$ corresponding to each temporal definition T_i of φ_{flat}. Relativizing these MTL formulae and conjuncting them, we obtain an MTL formula $\bigwedge_i Rel(\psi_i, \Sigma)$ that is equisatisfiable to φ (see Figure 1 for relativization).

The above steps are routine [11], [12]. What remains is to handle the temporal definitions.

Embedding the Runs of the DFA

For any given ρ over $\Sigma \cup W$, where W is the set of witness propositions used in the temporal definitions T of the forms $\Box^{\mathsf{ns}}[w \leftrightarrow \mathsf{Rat}_I \mathsf{re{-}atom}]$ or $\Box^{\mathsf{ns}}[w \leftrightarrow x\mathsf{URat}_{I',\mathsf{re-atom}}y]$, the rational expression re−atom has a corresponding minimal DFA recognizing it. We define an LTL formula $\mathsf{GOODRUN}(\phi_e)$ which takes a formula ϕ_e as a parameter with the following behaviour. $\rho, i \models \mathsf{GOODRUN}(\phi_e)$ iff for all $k > i$, $(\rho, k \models \phi_e) \rightarrow (\rho[i, k] \in L(\mathsf{re{-}atom}))$. To achieve this, we use two new sets of symbols Threads and Merge for this information. This results in the extended alphabet $\Sigma \cup W \cup \mathsf{Threads} \cup \mathsf{Merge}$ for the simple extension ρ' of ρ. The behaviour of Threads and Merge are explained below.

Consider $\mathsf{re{-}atom} = \mathsf{re}(\mathsf{S})$. Let $\mathcal{A}_{\mathsf{re-atom}} = (Q, 2^{\mathsf{S}}, \delta, q_1, Q_F)$ be the minimal DFA for re−atom and let $Q = \{q_1, q_2, \ldots, q_m\}$. Let $\mathsf{In} = \{1, 2, \ldots, m\}$ be the indices of the states.

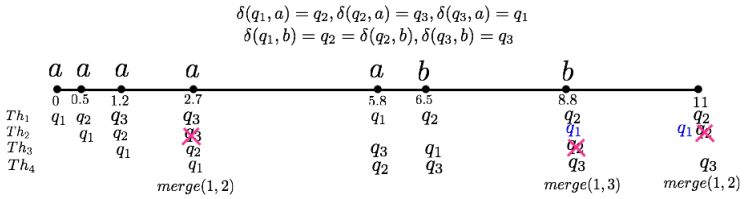


Figure 2 Depiction of threads and merging. At time point 2.7, thread 2 is merged with 1, since they both had the same state information. This thread remains inactive till time point 8.8, where it becomes active, by starting a new run in state q_1. At time point 8.8, thread 3 merges with thread 1, while at time point 11, thread 2 merges with 1, but is reactivated in state q_1.

Conceptually, we consider multiple runs of $\mathcal{A}_{\mathsf{re-atom}}$ with a new run (new thread) started at each point in ρ. Threads records the state of each previously started run. At each step, each thread is updated from it previous value according to the transition function δ of $\mathcal{A}_{\mathsf{re-atom}}$ and also augmented with a new run in initial state. Potentially, the number of threads would grow unboundedly in size but notice that once two runs are the same state at position i they remain identical in future. Hence they can be merged into single thread (see Figure2). As a result, m threads suffice. We record whether threads are merged in the current state using variables Merge. An LTL formula records the evolution of Threads and Merge over any behaviour ρ. We can define formula $\mathsf{GOODRUN}(\phi_e)$ in LTL over Threads and Merge.

1. At each position, let $\mathsf{Th}_i(q_x)$ be a proposition that denotes that the ith thread is active and is in state q_x, while $\mathsf{Th}_i(\bot)$ be a proposition that denotes that the ith thread is not active. The set Threads consists of propositions $\mathsf{Th}_i(q_x), \mathsf{Th}_i(\bot)$ for $1 \leq i, x \leq m$.
2. If at a position e, we have $\mathsf{Th}_i(q_x)$ and $\mathsf{Th}_j(q_y)$ for $i < j$, and if $\delta(q_x, \sigma_e) = \delta(q_y, \sigma_e)$, then we can merge the threads i, j at position $e+1$. Let $\mathsf{merge}(i,j)$ be a proposition that signifies that threads i, j have been merged. In this case, $\mathsf{merge}(i,j)$ is true at position $e+1$. Let Merge be the set of all propositions $\mathsf{merge}(i,j)$ for $1 \leq i < j \leq m$.

We now describe the conditions to be checked in ρ'.

- **Initial condition**(φ_{init})- At the first point of the word, we start the first thread and initialize all other threads as $\bot$: $\varphi_{init} = ((\mathsf{Th}_1(q_1)) \wedge \bigwedge_{1<i\leq m} \mathsf{Th}_i(\bot))$.
- **Initiating runs at all points**(φ_{start})- To check the rational expression within an arbitrary interval, we need to start a new run from every point. $\varphi_{start} = \Box^{\mathsf{ns}}(\bigvee_{i\leq m} \mathsf{Th}_i(q_1))$
- **Disallowing Redundancy**(φ_{no-red})- At any point of the word, if $i < j$ and $\mathsf{Th}_i(q_x)$ and $\mathsf{Th}_j(q_x)$ are both true, $q_x \neq q_y$. $\varphi_{no-red} = \bigwedge_{x\in\mathsf{In}} \Box^{\mathsf{ns}}[\neg \bigvee_{1\leq i<j\leq m} (\mathsf{Th}_i(q_x) \wedge \mathsf{Th}_j(q_x))]$
- **Merging Runs**$(\varphi_{\mathsf{merge}})$- If two different threads $\mathsf{Th}_i, \mathsf{Th}_j (i < j)$ reach the same state q_x on reading the input at the present point, then we merge thread Th_j with Th_i. We remember the merge with the proposition $\mathsf{merge}(i,j)$. We define a macro $\mathsf{Nxt}(\mathsf{Th}_i(q_x))$ which is true at a point e if and only if $\mathsf{Th}_i(q_y)$ is true at e and $\delta(q_y, \sigma_e) = q_x$, where $\sigma_e \subseteq AP$ is the maximal set of propositions true at e: $\bigvee_{\{(q_y,prop)\in(Q,2^{AP})|\delta(q_y,prop)=q_x\}} [prop \wedge \mathsf{Th}_i(q_y)]$.
  Let $\psi(i,j,k,q_x)$ be a formula that says that at the next position, $\mathsf{Th}_i(q_x)$ and $\mathsf{Th}_k(q_x)$ are true for $k > i$, but for all $j < i$, $\mathsf{Th}_j(q_x)$ is not. $\psi(i,j,k,q_x)$ is given by $\mathsf{Nxt}(\mathsf{Th}_i(q_x)) \wedge \bigwedge_{j<i} \neg\mathsf{Nxt}(\mathsf{Th}_j(q_x)) \wedge \mathsf{Nxt}(\mathsf{Th}_k(q_x))$. In this case, we merge threads $\mathsf{Th}_i, \mathsf{Th}_k$, and either restart Th_k in the initial state, or deactivate the kth thread at the next position. This is given by the formula $\mathsf{NextMerge(i,k)} = \mathsf{O}[\mathsf{merge}(i,k) \wedge (\mathsf{Th}_k(\bot) \vee \mathsf{Th}_k(q_1)) \wedge \mathsf{Th}_i(q_x)]$.
  $\varphi_{\mathsf{merge}} = \bigwedge_{x,i,k\in\mathsf{In}\wedge k>i} \Box^{\mathsf{ns}}[\psi(i,j,k,q_x) \rightarrow \mathsf{NextMerge(i,k)}]$.

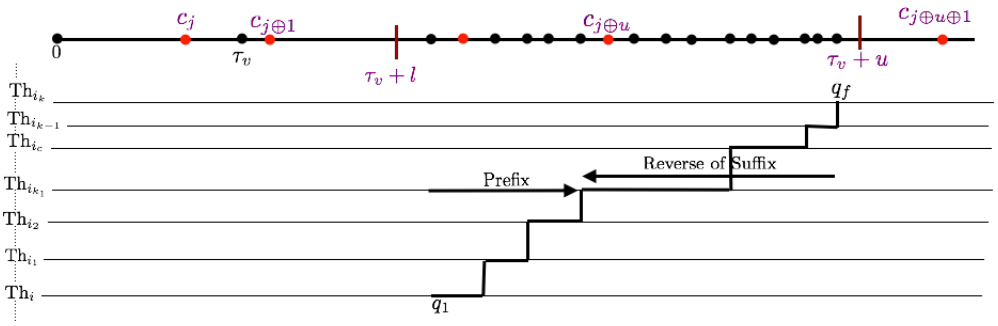


Figure 3 The linking thread at $c_{j\oplus u}$. The points in red are the oversampling integer points, and so are $\tau_v + l$ and $\tau_v + u$.

- **Propagating runs**$(\varphi_{pro}, \varphi_{NO-pro})$- If $\mathsf{Nxt}(\mathsf{Th}_i(q_x))$ is true at a point, and if for all $j < i$, $\neg\mathsf{Nxt}(\mathsf{Th}_j(q_x))$ is true, then at the next point, we have $\mathsf{Th}_i(q_x)$. Let $\mathsf{NextTh}(i, j, q_x)$ denote the formula $\mathsf{Nxt}(\mathsf{Th}_i(q_x)) \wedge \neg\mathsf{Nxt}(\mathsf{Th}_j(q_x))$. The formula φ_{pro} is given by $\bigwedge_{i,j\in\mathsf{In}\wedge i<j} \Box^{\mathsf{ns}}[\mathsf{NextTh}(i, j, q_x) \rightarrow \mathsf{O}[\mathsf{Th}_i(q_x) \wedge \neg\mathsf{merge}(i, j)]]$. If $\mathsf{Th}_i(\bot)$ is true at the current point, then at the next point, either $\mathsf{Th}_i(\bot)$ or $\mathsf{Th}_i(q_1)$. The latter condition corresponds to starting a new run on thread Th_i. $\varphi_{NO-pro} = \bigwedge_{i\in\mathsf{In}} \Box^{\mathsf{ns}}\{\mathsf{Th}_i(\bot) \rightarrow \mathsf{O}(\mathsf{Th}_i(\bot) \vee \mathsf{Th}_i(q_1))\}$

Let Run be the formula obtained by conjuncting all formulae explained above. Once we construct the simple extension ρ', checking whether the rational expression $\mathsf{re{-}atom}$ holds in some interval I in the timed word ρ, is equivalent to checking that if u is the first action point within I, and if $\mathsf{Th}_i(q_1)$ holds at u, then after a series of merges of the form $\mathsf{merge}(i_1, i)$,$\mathsf{merge}(i_2, i_1)$, $\ldots$ $\mathsf{merge}(j, i_n)$, at the last point v in the interval I, $\mathsf{Th}_j(q_f)$ is true, for some final state q_f. This is encoded as $\mathsf{GOODRUN}(q_f)$. It can be seen that the number of possible sequences of merges are bounded. Figure 2 illustrates the threads and merging. To write an MTL formula that checks the truth of $\mathsf{Rat}_{[l,u)}\mathsf{re{-}atom}$ at a point v, we need to oversample ρ' as shown below.

▶ **Lemma 6.** *Let $T = \Box^{\mathsf{ns}}[w \leftrightarrow \mathsf{Rat}_I\mathsf{re{-}atom}]$ be a temporal definition built from $\Sigma \cup W$. Then we synthesize a formula $\psi \in \mathsf{MTL}$ over $\Sigma \cup W \cup X$ such that T is equivalent to ψ modulo oversampling.*

Proof. Lets first consider the case when the interval I is bounded of the form $[l, u)$. Consider a point in ρ' with time stamp τ_v. To assert w at τ_v, we look at the first action point after time point $\tau_v + l$, and check that $\mathsf{GOODRUN}(last(q_f))$ holds, where $last(q_f)$ identifies the last action point just before $\tau_v + u$. The first difficulty is the possible absence of time points $\tau_v + l$ and $\tau_v + u$. To overcome this difficulty, we oversample ρ' by introducing points at times $t + l, t + u$, whenever t is a time point in ρ'. These new points are labelled with a new proposition ovs. Sadly, $last(q_f)$ cannot be written in MTL.

To address this, we introduce new time points at every integer point of ρ'. The starting point 0 is labelled c_0. Consecutive integer time points are marked $c_i, c_{i\oplus 1}$, where $\oplus$ is addition modulo the maximum constant used in the time interval in the RatMTL formula. This helps in measuring the time elapse since the first action point after $\tau_v + l$, till the last action point before $\tau_v + u$ as follows: if $\tau_v + l$ lies between points marked $c_j, c_{j\oplus 1}$, then the last integer point before $\tau_v + u$ is **uniquely** marked $c_{j\oplus u}$.

- Anchoring at τ_v, we assert the following at distance l: no action points are seen until the first action point where $\mathsf{Th}_i(q_1)$ is true for some thread Th_i. Consider the next point

where $c_{j\oplus u}$ is seen. Let $\mathsf{Th}_{i_{k_1}}$ be the thread to which Th_i has merged at the last action point just before $c_{j\oplus u}$. Let us call $\mathsf{Th}_{i_{k_1}}$ the "last merged thread" before $c_{j\oplus u}$. The sequence of merges from Th_i till $\mathsf{Th}_{i_{k_1}}$ asserts a prefix of the run that we are looking for between $\tau_v + l$ and $\tau_v + u$. To complete the run we mention the sequence of merges from $\mathsf{Th}_{i_{k_1}}$ which culminates in some $\mathsf{Th}_{i_k}(q_f)$ at the last action point before $\tau_v + u$.

- Anchoring at τ_v, we assert the following at distance u: we see no action points since $\mathsf{Th}_{i_k}(q_f)$ at the action point before $\tau_v + u$ for some thread Th_{i_k}, and there is a path linking thread $\mathsf{Th}_{i_{k_1}}$ to Th_{i_k} since the point $c_{j\oplus u}$. We assert that the "last merged thread", $\mathsf{Th}_{i_{k_1}}$ is active at $c_{j\oplus u}$: this is the linking thread which is last merged into before $c_{j\oplus u}$, and which is the first thread which merges into another thread after $c_{j\oplus u}$.

These two formulae thus "stitch" the actual run observed between points $\tau_v + l$ and $\tau_v + u$. The formal technical details can be seen in Appendix D in the full version. If I was an unbounded interval of the form $[l, \infty)$, then we will go all the way till the end of the word, and assert $\mathsf{Th}_{i_k}(q_f)$ at the last action point of the word. Thus, for unbounded intervals, we do not need any oversampling at integer points. ◀

In a similar manner, we can eliminate the URat modality, the proof of which can be found in Appendix E in the full version. If we choose to work on logic MITL + URat, we obtain a 2EXPSPACE upper bound for satisfiability checking, since elimination of URat results in an equisatisfiable MITL formula. This is an interesting consequence of the oversampling technique; without oversampling, we can eliminate URat obtaining 1-TPTL (Appendix C, full version). However, 1-TPTL does not enjoy the benefits of non-punctuality, and is non-primitive recursive (Appendix F, full version).

4 Automaton-Metric Temporal Logic-Freeze Logic Equivalences

The focus of this section is to obtain equivalences between automata, temporal and freeze logics. First of all, we identify a fragment of RatMTL denoted SfrMTL, where the rational expressions in the formulae are all star-free. We then show the equivalence between po-1-clock ATA, 1−TPTL, and SfrMTL (po-1-clock ATA $\subseteq$ SfrMTL $\subseteq$ 1−TPTL $\equiv$ po-1-clock ATA). The main result of this section gives a tight automaton-logic connection in Theorem 7, and is proved using Lemmas 9, 10 and 11.

▶ **Theorem 7.** *1−TPTL, SfrMTL and po-1-clock ATA are all equivalent.*

We first show that partially ordered 1-clock alternating timed automata (po-1-clock ATA) capture exactly the same class of languages as 1−TPTL. We also show that 1−TPTL is equivalent to the subclass SfrMTL of RatMTL where the rational expressions re involved in the formulae are such that $L(\mathsf{re})$ is star-free.

A 1-clock ATA [15] is a tuple $\mathcal{A} = (\Sigma, S, s_0, F, \delta)$, where Σ is a finite alphabet, S is a finite set of locations, $s_0 \in S$ is the initial location and $F \subseteq S$ is the set of final locations. Let x denote the clock variable in the 1-clock ATA, and $x \bowtie c$ denote a clock constraint where $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, >, \geq\}$. Let X denote a finite set of clock constraints of the form $x \bowtie c$. The transition function is defined as $\delta : S \times \Sigma \to \Phi(S \cup \Sigma \cup X)$ where $\Phi(S \cup \Sigma \cup X)$ is a set of formulae defined by the grammar $\varphi ::= \top | \bot | \varphi_1 \wedge \varphi_2 | \varphi_1 \vee \varphi_2 | s | x \bowtie c | x.\varphi$ where $s \in S$, and $x.\varphi$ is a binding construct corresponding to resetting the clock x to 0.

The notation $\Phi(S \cup \Sigma \cup X)$ thus allows boolean combinations as defined above of locations, symbols of Σ, clock constraints and $\top, \bot$, with or without the binding construct $(x.)$. A configuration of a 1-clock ATA is a set consisting of locations along with their clock valuation. Given a configuration C, we denote by $\delta(C, a)$ the configuration D obtained by applying

$\delta(s,a)$ to each location s such that $(s,\nu) \in C$. A run of the 1-clock ATA starts from the initial configuration $\{(s_0,0)\}$, and proceeds with alternating time elapse transitions and discrete transitions obtained on reading a symbol from Σ. A configuration is accepting iff it is either empty, or is of the form $\{(s,\nu) \mid s \in F\}$. The language accepted by a 1-clock ATA $\mathcal{A}$, denoted $L(\mathcal{A})$ is the set of all timed words ρ such that starting from $\{(s_0,0)\}$, reading ρ leads to an accepting configuration. A po-1-clock ATA is one in which (i) there is a partial order denoted $\prec$ on the locations, such that whenever s_j appears in $\Phi(s_i)$, $s_j \prec s_i$, or $s_j = s_i$. Let $\downarrow s_i = \{s_j \mid s_j \prec s_i\}$, (ii) $x.s$ does not appear in $\delta(s,a)$ for all $s \in S, a \in \Sigma$.

▶ **Example 8.** Consider the po-1-clock ATA $\mathcal{A} = (\{a,b\}, \{s_0, s_a, s_\ell\}, s_0, \{s_0, s_\ell\}, \delta)$ with transitions $\delta(s_0,b) = s_0, \delta(s_0,a) = (s_0 \wedge x.s_a) \vee s_\ell$, $\delta(s_a,a) = (s_a \wedge x < 1) \vee (x > 1) = \delta(s_a,b)$, and $\delta(s_\ell,b) = s_\ell, \delta(s_\ell,a) = \bot$. The automaton accepts all strings where every non-last a has no symbols at distance 1 from it, and has some symbol at distance > 1 from it.

▶ **Lemma 9.** po-*1-clock ATA and* $1-$TPTL *are equivalent in expressive power.*

The translation from $1-$TPTL to po-1-clock ATA is easy, as in the translation from MTL to po-1-clock ATA. For the reverse direction, we start from the lowest location (say s) in the partial order, and replace the transitions of s by a 1-TPTL formula that models timed words which are accepted, when started in s. The accepting behaviours of each location s, denoted $\mathsf{Beh}(s)$ is computed bottom up. The 1-TPTL formula that we are looking for is $\mathsf{Beh}(s_0)$ where s_0 is the initial location. In example 8, $\mathsf{Beh}(s_\ell) = \Box^{\mathsf{ns}} b$, $\mathsf{Beh}(s_a) = (x < 1)\ \mathsf{U}^{\mathsf{ns}}(x > 1)$, $\mathsf{Beh}(s_0) = [(a \wedge x.\mathsf{OBeh}(s_a)) \vee b]\ \mathsf{W}(a \wedge \mathsf{OBeh}(s_\ell)) = ((a \wedge (x.\mathsf{O}[(x < 1)\ \mathsf{U}^{\mathsf{ns}} x > 1])) \vee b)\ \mathsf{W}(a \wedge \mathsf{O}\Box^{\mathsf{ns}} b)$. Step by step details for Lemma 9 can be seen in Appendix H of the full version.

▶ **Lemma 10.** $\mathsf{SfrMTL} \subseteq 1 - \mathsf{TPTL}$.

The proof of Lemma 10 can be found in Appendix I of the full version. The intuition is to freeze a clock x at the current point, and write an LTL formula equivalent to the star-free expression over an interval I which can be constrained checking $x \in I$ in the LTL formula.

▶ **Lemma 11.** *(*po-*1-clock ATA to* SfrMTL*) Given a* po-*1-clock ATA* $\mathcal{A}$*, we can construct a* SfrMTL *formula* φ *such that* $L(\mathcal{A}) = L(\varphi)$.

Proof. (Sketch) We give a proof sketch here, a detailed proof can be found in Appendix J of the full version. Let $\mathcal{A}$ be a po-1-clock ATA with locations $S = \{s_0, s_1, \ldots, s_n\}$. Let K be the maximal constant used in the guards $x \sim c$ occurring in the transitions. Let $R_{2i} = [i,i], R_{2i+1} = (i, i+1), 0 \le i < K$ and $R_K^+ = (K, \infty)$ be the regions $\mathcal{R}$ of x. Let $R_h \prec R_k$ denote that region R_h precedes region R_k. For each location s,$\mathsf{Beh}(s)$ as computed in Lemma 9 is a 1-TPTL formula that gives the timed behaviour starting at s, using constraints $x \sim c$ since the point where x was frozen. In example 8, $\mathsf{Beh}(s_a) = (x < 1)\ \mathsf{U}^{\mathsf{ns}}(x > 1)$, allows symbols a, b as long as $x < 1$ keeping the control in s_a, has no behaviour at $x = 1$, and allows control to leave s_a when $x > 1$. For any s, we "distribute" $\mathsf{Beh}(s)$ across regions by untiming it. In example 8, $\mathsf{Beh}(s_a)$ is $\Box^{\mathsf{ns}}(a \vee b)$ for regions R_0, R_1, it is $\bot$ for R_2 and is $(a \vee b)$ for R_1^+. Given any $\mathsf{Beh}(s)$, and a pair of regions $R_j \preceq R_k$, such that s has a non-empty behaviour in region R_j, and control leaves s in R_k, the untimed behaviour of s between regions $R_j, \ldots, R_k$ is written as LTL formulae $\varphi_j, \ldots, \varphi_k$. This results in a "behaviour description" (or BD for short) denoted $\mathsf{BD}(s, R_j, R_k) = \{\mathsf{BD}_1, \mathsf{BD}_2, \ldots, \mathsf{BD}_w\}$[2] where each BD_i is a $2K+1$

[2] Note that if s is one of the lowest locations in the partial order, this is a singleton set. We will denote the elements of $\mathsf{BD}(s, R_j, R_k)$ as $BD_{no.}$.

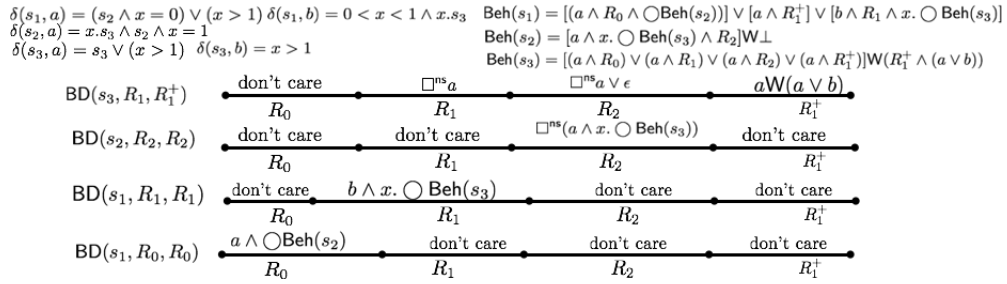


Figure 4 A po-1-clock ATA with initial location s_1 and s_2, s_3 are accepting.

tuples with $\mathsf{BD}_i[R_l] = \varphi_l$ for $j \leq l \leq k$, and $\mathsf{BD}[R] = \top$ denoting "dont care" for the other regions. Let $\mathsf{BDSet}(s)$ denote the union of all BDs for a location s. For the initial location s_0, consider all $\mathsf{BD}_i \in \mathsf{BD}(s_0, R_j, R_k)$ that have a behaviour starting in R_j, and ends in an accepting configuration in R_k. Each LTL formula $\mathsf{BD}_i[R_i]$ is replaced with a star-free rational expression denoted $\mathsf{re}(\mathsf{BD}(s_0, R_j, R_k)[R_i])$. Then $\mathsf{BD}(s_0, R_j, R_k)$ is transformed into a SfrMTL formula $\varphi(s_0, R_j, R_k) = \bigvee_{BD_i \in \mathsf{BD}(s_0,R_j,R_k)} \bigwedge_{j \leq g \leq k} \mathsf{Rat}_{R_g}\mathsf{re}(\mathsf{BD}_i[R_g])$. The language accepted by the po-1-clock ATA $\mathcal{A}$ is then given by $\bigvee_{0 \leq j \leq k \leq 2K} \varphi(s_0, R_j, R_k)$.

Computing $\mathsf{BD}(s, R_i, R_j)$ for a location s and pair of regions $R_i \preceq R_j$. We first compute $\mathsf{BD}(s, R_i, R_j)$ for locations s which are lowest in the partial order, followed by computing $\mathsf{BD}(s', R_i, R_j)$ for locations s' which are higher in the order. For any location s, $\mathsf{Beh}(s)$ has the form φ or $\varphi_1 \mathsf{W}\varphi_2$ or $\varphi_1 \mathsf{U}^{ns}\varphi_2$, where $\varphi, \varphi_1, \varphi_2$ are disjunctions of conjunctions over $\Phi(S \cup \Sigma \cup X)$, where S is the set of locations with or without the binding construct $x.$, and X is a set of clock constraints of the form $x \sim c$. Each conjunct has the form $\psi \wedge x \in R$ where $\psi \in \Phi(\Sigma \cup S)$ and $R \in \mathcal{R}$. Let $\varphi_1 = \bigvee(P_i \wedge C_i), \varphi_2 = \bigvee(Q_j \wedge E_j)$ where $P_i, Q_j \in \Phi(\Sigma \cup S)$ and $C_i, E_j \in \mathcal{R}$. Let $\mathcal{C}$ and $\mathcal{E}$ be shorthands for any C_k, E_l.

If $\mathsf{Beh}(s)$ is an expression without U, W (the case of φ above), then $\mathsf{BD}(s, R_i, R_i)$ is defined for a region R_i if $\varphi = \bigvee(Q_j \wedge E_j)$ and there is some E_l with $x \in R_i$. It is a $2K + 1$ tuple with $\mathsf{BD}(s, R_i, R_i)[R_i] = Q_l$[3]we know that , and the rest of the entries are $\top$ (for dont care). If $\mathsf{Beh}(s)$ has the form $\varphi_1 \mathsf{W}\varphi_2$ or $\varphi_1 \mathsf{U}^{ns}\varphi_2$, then for $R_i \preceq R_j$, and a location s, $\mathsf{BD}(s, R_i, R_j) = \{BD_1\}$ where BD_1 is a $2K + 1$ tuple with (i) formula $\top$ in regions $R_0, \ldots, R_{i-1}, R_{j+1}, \ldots, R_K^+$, (ii) If $C_k = E_l = (x \in R_j)$ for some C_k, E_l, then the LTL formula in region R_j is $P_k \mathsf{U}Q_l$ if s is not accepting, and is $P_k \mathsf{W}Q_l$ if s is accepting, (iii) If no C_k is equal to any E_l, and if $E_l = (x \in R_j)$ for some l, then the formula in region R_j is Q_l. If $C_m = (x \in R_i)$ for some m, then the formula for region R_i is $\Box^{ns}P_m$. If there is some $C_h = (x \in R_w)$ for $i < w < j$, then the formula in region R_w is $\Box^{ns}P_h \vee \epsilon$, where ϵ signifies that there may be no points in regions R_w. If there are no C_m's such that $C_m = (x \in R_w)$ for $R_i \prec R_w \prec R_j$, then the formula in region R_w is ϵ. ϵ is used as a *special symbol* in LTL whenever there is no behaviour in a region.

$\mathsf{BD}(s, R_i, R_j)$ for location s lowest in po. Let s be a location that is lowest in the partial order. In general, if s is the lowest in the partial order, then $\mathsf{Beh}(s)$ has the form $\varphi_1 \mathsf{W}\varphi_2$ or $\varphi_1 \mathsf{U}^{ns}\varphi_2$ or φ where $\varphi, \varphi_1, \varphi_2$ are disjunctions of conjunctions over $\Phi(\Sigma \cup X)$. Each conjunct has the form $\psi \wedge x \in R$ where $\psi \in \Phi(\Sigma)$ and $R \in \mathcal{R}$. See Figure 4, with regions

[3] We abuse the notation by indexing the $\mathsf{BD}(s, R_i, R_i)[R_i]$ instead of BD when it is a singleton set.

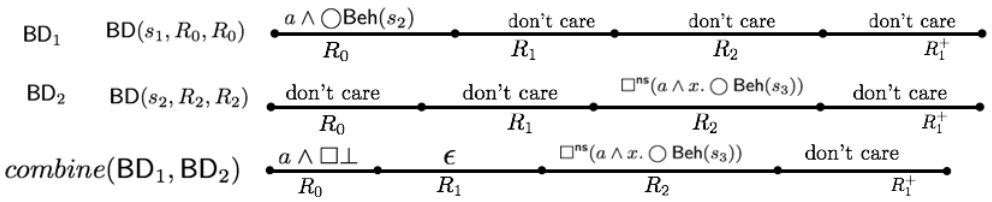


Figure 5 Combining BDs

R_0, R_1, R_2, R_1^+, and some example BDs. In Figure 4, using the BDs of the lowest location s_3, we write the SfrMTL formula for $\mathsf{Beh}(s_3)$: $\psi(s_3) = \varphi_{R_0}(s_3) \wedge \varphi_{R_1}(s_3) \wedge \varphi_{R_2}(s_3) \wedge \varphi_{R_1^+}(s_3)$, where each φ_R describes the behaviour of s_3 starting from region R. For a fixed region R_i, $\varphi_{R_i}(s_3)$ is $\bigwedge_{R_g \prec R_i} \mathsf{Rat}_{R_g}\epsilon \wedge \mathsf{Rat}_{R_i}\Sigma^+ \rightarrow \{\bigvee_{R_i \prec R_j} \varphi(s_3, R_i, R_j)\}$, where $\varphi(s_3, R_i, R_j)$ is described above. $\mathsf{Rat}_{R_g}\epsilon$ means that there is no behaviour in R_g. $\varphi_{R_0}(s_3)$ is given by $\mathsf{Rat}_{R_0}\Sigma^+ \rightarrow \{(\mathsf{Rat}_{R_0}a^* \wedge \mathsf{Rat}_{R_1}[a^* + \epsilon] \wedge \mathsf{Rat}_{R_2}[a^* + \epsilon] \wedge \mathsf{Rat}_{R_1^+}[a^* + a^*b])\}$.

$\mathsf{BD}(s, R_i, R_j)$ for a location s which is higher up . If s is not the lowest in the partial order, then $\mathsf{Beh}(s)$ can have locations $s' \in\downarrow s$. s' occurs as $\mathsf{O}(s')$ or $x.\mathsf{O}(s')$ in $\mathsf{Beh}(s)$. For $x.\mathsf{OBeh}(s_3)$ in $\mathsf{BD}(s, R_i, R_j)$, since the clock is frozen, we plug-in the SfrMTL formula $\psi(s_3)$ computed above for $x.\mathsf{OBeh}(s_3)$ in $\mathsf{BD}(s_1, R_i, R_j)$. For instance, in figure 4, $x.\mathsf{OBeh}(s_3)$ appears in $\mathsf{BD}(s_2, R_2, R_2)[R_2]$. We simply plug in the SfrMTL formula $\psi(s_3)$ in its place. Likewise, for locations s, t, if $\mathsf{OBeh}(t)$ occurs in $\mathsf{BD}(s, R_i, R_j)[R_k]$, we look up $\mathsf{BD}(t, R_k, R_l) \in \mathsf{BDSet}(t)$ for all $R_k \preceq R_l$ and *combine* $\mathsf{BD}(s, R_i, R_j), \mathsf{BD}(t, R_k, R_l)$ in a manner described below. This is done to detect if the "next point" for t has a behaviour in R_k or later.

(a) If the next point for t is in R_k itself, then we *combine* all $\mathsf{BD}_1 \in \mathsf{BD}(s, R_i, R_j)$ with every $\mathsf{BD}_2 \in \bigcup_{R_k \preceq R_l} \mathsf{BD}(t, R_k, R_l) \subseteq \mathsf{BDSet}(t)$ as follows[4]. $\mathsf{combine}(\mathsf{BD}_1, \mathsf{BD}_2)$ results in BD_3 such that $\mathsf{BD}_3[R]$=$\mathsf{BD}_1[R]$ for $R \prec R_k$, $\mathsf{BD}_3[R]$=$\mathsf{BD}_1[R] \wedge \mathsf{BD}_2[R]$ for $R_k \prec R$, where $\wedge$ denotes component wise conjunction. $\mathsf{BD}_3[R_k]$ is obtained by replacing $\mathsf{OBeh}(s_2)$ in $\mathsf{BD}_1[R_k]$ with $\mathsf{BD}_2[R_k]$. Doing so enables the next point in R_k, emulating the behaviour of t in R_k.

(b) Assume the next point for t lies in R_b, $R_k \prec R_b$. The difference with case (a) is that we combine $\mathsf{BD}_1 \in \mathsf{BD}(s, R_i, R_j)$ with $\mathsf{BD}_2 \in \bigcup_{R_k \preceq R_l} \mathsf{BD}(t, R_k, R_l) \subseteq \mathsf{BDSet}(t)$. Then $\mathsf{combine}(\mathsf{BD}_1, \mathsf{BD}_2)$ results in a BD, say BD_3 such that $\mathsf{BD}_3[R] = \mathsf{BD}_1[R]$ for $R \prec R_k$, $\mathsf{BD}_3[R] = \mathsf{BD}_1[R] \wedge \mathsf{BD}_2[R]$ for all $R_b \preceq R$, and $\mathsf{BD}_3[R] = \epsilon$ for $R_k \prec R \prec R_b$. The $\mathsf{OBeh}(t)$ in $\mathsf{BD}_1[R_k]$ is replaced with $\Box\bot$ to signify that the next point is not enabled for t. See Figure 5 where $R_b = R_2$. The conjunction with $\Box\bot$ in R_0 signifies that the next point for s_2 is not in R_0; the ϵ in R_1 signifies that there are no points in R_1 for s_2. Conjuncting $\Box\bot$ in a region signifies that the next point does not lie in this region.

We look at the "accepting" BDs in $\mathsf{BDSet}(s_0)$, viz., all $\mathsf{BD}(s_0, R_j, R_k)$, such that acceptance happens in R_k, and s_0 has a behaviour starting in R_j. The LTL formulae $\mathsf{BD}_i[R]$ [where $\mathsf{BD}_i \in \mathsf{BDSet}(s_0)$] is replaced with star-free expression $\mathsf{re}(\mathsf{BD}_i[R])$. $\mathsf{BDSet}(s_0)$ gives an SfrMTL formula φ= $\bigvee_{\mathsf{BD}_i \in \mathsf{BDSet}(s_0)} \bigwedge_{R_j \preceq R \preceq R_k} \mathsf{Rat}_R\mathsf{re}(\mathsf{BD}_i[R])$ whose language is $L(\varphi)$=$L(\mathcal{A})$. ◀

[4] Take cross product of two sets and then applying combine operation

5 Discussion

We propose RatMTL which significantly increases the expressive power of MTL and yet retains decidability over pointwise finite words. The Rat operator added to MTL syntactically subsumes several other modalities in literature including threshold counting, modulo counting and the Pnueli modality. The reduction of RatMTL to equisatisfiable MTL has elementary complexity and allows us to identify two fragments of RatMTL with 2EXPSPACE and EXPSPACE satisfiability. In [11], oversampled temporal projections were used to reduce MTL with punctual future and non-punctual past to MTL. Our reduction can be combined with the one in [11] to obtain decidability of RatMTL and elementary decidability of MITL + URat + non-punctual past. These are amongst the most expressive decidable extensions of MTL known so far. The exact complexity class for satisfiability of MITL + URat is an interesting open question. We also show an exact logic-automaton correspondence between the fragment SfrMTL and po-1-clock ATA. It is not difficult to see that full RatMTL can be reduced to equivalent 1 clock ATA. This provides an alternative proof of decidability of RatMTL but the proof will not extend to decidability of RatMTL+ non-punctual past, nor prove elementary decidability of MITL + URat+non-punctual past. Hence, we believe that our proof technique has some advantages. An interesting related formalism of timed regular expressions was defined by Asarin, Maler, Caspi, and shown to be expressively equivalent to timed automata. Our RatMTL has orthogonal expressive power, and it is boolean closed (thus the decidability of universality checking comes for free). The exact expressive power of RatMTL which is between 1-clock ATA and po-1-clock ATA is open.

References

1. R. Alur, T. Feder, and T. Henzinger. The benefits of relaxing punctuality. *J.ACM*, 43(1):116–146, 1996.
2. Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993. doi:10.1006/inco.1993.1025.
3. Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002. doi:10.1145/506147.506151.
4. Augustin Baziramwabo, Pierre McKenzie, and Denis Thérien. Modular temporal logic. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 344–351, 1999. doi:10.1109/LICS.1999.782629.
5. Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, pages 432–443, 2005. doi:10.1007/11590156_35.
6. Cindy Eisner and Dana Fisman. *A Practical Introduction to PSL*. Springer, 2006.
7. IEEE P1850-Standard for PSL-Property Specification Language, 2005.
8. Jesper G. Henriksen and P. S. Thiagarajan. Dynamic linear time temporal logic. *Ann. Pure Appl. Logic*, 96(1-3):187–207, 1999. doi:10.1016/S0168-0072(98)00039-6.
9. Philippe Herrmann. Renaming is necessary in timed regular expressions. In *Foundations of Software Technology and Theoretical Computer Science, 19th Conference, Chennai, India, December 13-15, 1999, Proceedings*, pages 47–59, 1999. doi:10.1007/3-540-46691-6_4.
10. P. Hunter. When is metric temporal logic expressively complete? In *CSL*, pages 380–394, 2013.
11. S. N. Krishna K. Madnani and P. K. Pandya. Partially punctual metric temporal logic is decidable. In *TIME*, pages 174–183, 2014.

12 Shankara Narayanan Krishna, Khushraj Madnani, and Paritosh K. Pandya. Metric temporal logic with counting. In *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 335–352, 2016.

13 F. Laroussinie, A. Meyer, and E. Petonnet. Counting ltl. In *TIME*, pages 51–58, 2010.

14 K. Lodaya and A. V. Sreejith. Ltl can be more succinct. In *ATVA*, pages 245–258, 2010.

15 J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *LICS*, pages 188–197, 2005.

16 A. Rabinovich. Complexity of metric temporal logic with counting and pnueli modalities. In *FORMATS*, pages 93–108, 2008.

Complexity of Restricted Variants of Skolem and Related Problems

S. Akshay[1], Nikhil Balaji[*2], and Nikhil Vyas[3]

1 **Department of Computer Science and Engineering, IIT Bombay, India**
akshayss@cse.iitb.ac.in
2 **Aalen University, Aalen, Germany**
nikhilrbalaji@gmail.com
3 **Department of Computer Science and Engineering, IIT Bombay, India**
nikhilvyas@cse.iitb.ac.in

Abstract

Given a linear recurrence sequence (LRS), the Skolem problem, asks whether it ever becomes zero. The decidability of this problem has been open for several decades. Currently decidability is known only for LRS of order upto 4. For arbitrary orders (i.e., number of terms the n^{th} depends on), the only known complexity result is NP-hardness by a result of Blondel and Portier from 2002.

In this paper, we give a different proof of this hardness result, which is arguably simpler and pinpoints the source of hardness. To demonstrate this, we identify a subclass of LRS for which the Skolem problem is in fact NP-complete. We show the generic nature of our lower-bound technique by adapting it to show stronger lower bounds of a related problem that encompasses many known decision problems on linear recurrent sequences.

1998 ACM Subject Classification F.1.1 Models of computation, F.2 Analysis of Algorithms and Problem Complexity, F.2.1. Numerical Algorithms and Problems

Keywords and phrases Linear recurrence sequences, Skolem problem, NP-completeness, Program termination

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.78

1 Introduction

While linear dynamical systems have been studied for a long time, several interesting and computationally relevant decision problems remain unsolved. The *Skolem problem* is a long-standing open problem in mathematics which asks whether zero ever occurs in the infinite sequence generated by a given linear recurrence sequence (LRS) with specific initial conditions. The *positivity problem* asks if the values of an LRS are always positive. Both these problems have received consider attention from mathematicians and computer scientists over the years. The positivity problem is related to program termination for initialized linear loop programs [23, 8, 16], while the Skolem and its variants have been considered in probabilistic verification [1, 2, 3] among other applications. However, despite decades of active research, the problems in their full generality have remained open.

While a result of Blondel and Portier [7] showed NP-hardness for the Skolem problem, the only known positive results are for very restricted class of recurrences, with restrictions

* Supported by DFG grant TH 472/4. Part of this work was done when the author was a Postdoctoral fellow at IIT Bombay.

on either the order of the recurrence (i.e., number of terms defining the sequence) or on the roots of the characteristic polynomial associated with the recurrence. On one hand, the Skolem problem is known to be decidable for order 4 [17, 25] and positivity for degree 6 [19]. On the other hand, by restricting LRS to only considering those with distinct roots (these are called simple LRS), Ouaknine and Worrell show decidability of positivity for order 9 LRS [18] and decidability of a related variant, *ultimate positivity* (is there a point after which the LRS is always positive) [20]. Further, in [18] they show that the decidability of the the positivity problem for order 6 LRS would entail solutions to longstanding open problems on Diophantine approximations. While it is well-known and probably a folklore result ([25]) that if the roots of characteristic polynomial are real (in fact, more generally, if the LRS has a single dominant root) then Skolem and positivity are decidable, the exact complexity of these problems have not been mapped out. Indeed, it is customary for papers in the area to study LRS that are *non-degenerate* (which is when there are no two characteristic roots such that their ratio is a root of unity), since it is known that the general case reduces to this case as far as decidability is concerned.

In this paper, we focus on linear recurrence sequences, whose characteristic polynomial has roots of some special form. Our contributions are the following: we give a new NP-hardness proof for the Skolem problem by a reduction from the classic Subset Sum problem. This gives an alternate proof of NP-hardness of Skolem (as well as coNP-hardness for positivity), which matches the best lower bound known for the Skolem problem, due to Blondel and Portier [7]. A closer inspection of our proof shows that the LRS that is output by our reduction is a special subclass of LRS whose characteristic polynomial has roots that are complex roots of unity (i.e., complex numbers α such that $\alpha^n = 1$ for some integer n). We investigate this natural class of LRS and match our lower bound by showing that the Skolem problem for this class can be solved in NP. Thus, we obtain a natural subclass of LRS of arbitrary order with an NP-complete Skolem problem, which to the best of our knowledge has not been observed before. Finally, we show that both the lower bound and upper bound techniques can be lifted to other problems.

We now explain the significance of all these three results and place them in the context of existing results. We start with the hardness result, where as mentioned earlier, Blondel and Portier [7] already proved NP-Hardness of Skolem. However, our proof is of independent interest for the following reasons:

- Our proof is a direct reduction from the classical subset sum problem and is arguably simpler/shorter than the proof in [7], which goes via automata theory, by showing a reduction from universality of unary NFA.
- The proof in [7] shows NP-hardness by considering LRS whose characteristic polynomials have 0/1-coefficients. While this is indeed a simple subclass of LRS, the characteristic polynomial of such LRS could still have complex roots with phase/angle that is an irrational multiple of π. Current techniques seem inadequate to solve the Skolem and positivity problem for LRS of this kind and hence do not give effective upper bounds. In contrast, for the subclass of LRS arising from our hardness proof, the Skolem problem admits an NP algorithm. [1]
- Our NP-hardness proof can be extended to show hardness for other problems as shown in Section 4.

Next, we turn to the NP upper bound. We first note that we are able to achieve this result for LRS of arbitrary orders. All upper bounds currently known for restricted variants of the

[1] In fact, a closer inspection of Blondel and Portier's proof, reveals that their hard instances actually fall into a stricter subclass, which can be shown to be NP-complete using our techniques.

Skolem (and related problems) problem, restrict the order to a fixed constant [18] or assume that the recurrence with arbitrary order is simple [20]. Our upper bound techniques rely on basic linear algebra and complexity theory and do not need the development or application of advanced techniques from algebraic number theory and Diophantine approximation as in the other results in [18, 20]. This indeed makes our proofs more elementary, but allows for easy generalization to other problems as we show next.

Our third and final contribution is to show that our lower bound proof and the upper bound techniques can be extended. To illustrate this, we consider a related variant of the Skolem and positivity problem which we call the *polytope containment problem*. We will define this problem in the matrix form rather than in terms of LRS, while noting that we can use Cayley-Hamilton theorem and basic linear algebra to see their equivalence (see [25] for details). Recall that a *(convex) polytope* is an intersection of finitely many half-spaces and it is said to be *bounded* if the region enclosed in it is bounded. Given a bounded polytope V_1 and a (possibly unbounded) polytope V_2 over $\mathbb{Z}^d$ and a $d * d$ matrix M with integer entries, the *Integer Polytope Containment Problem* asks if for all $v \in V_1$ does there exist a positive integer n such that $vM^n \in V_2$.

There are two main motivations to look at this problem. First, it generalizes the Skolem problem, higher-order orbit problem [10] and polyhedron hitting problem [9] over integers. For the former, we set V_1 to be the initial vector and V_2 the space orthogonal to the target vector (defined as the intersection of halfspaces $\{x \mid x \cdot w \leq 0 \wedge x \cdot w \geq 0\}$). The higher-order orbit problem is obtained by fixing V_1 as the initial vector and no restrictions on V_2.

The second main motivation is that the negation of this problem is closely related to program termination of linear loops. Program termination is a classical undecidable problem, but the special case of the problem over linear loops has received considerable attention ([21] surveys these results as well as their link to linear recurrence sequences). There are two main variants of this problem. First, the initialized termination problem asks: starting from a initial vector v, is it the case that for all $n \in \mathbb{N}$, $vM^n > 0$? Next the uninitialized termination problem asks: does there exist an initial vector v such that for all $n \in \mathbb{N}$, we have $vM^n > 0$? In [23, 8], Tiwari and Braverman showed that the uninitialized problem is decidable in polynomial time for reals and integers respectively. The initialized problem, often called the positivity problem, however is still open in its fully generality though some results in restricted cases have been proved recently [18, 20, 19].

We observe that the negation of the above defined polytope containment problem is: given a bounded polytope V_1 and a polytope V_2, does there exist $v \in V_1$ such that for all $n \in \mathbb{N}$, $vM^n \notin V_2$. By fixing V_2 to be the halfspace $\{x \in \mathbb{Z}^d \mid c^T x \leq 0\}$, we obtain (i) the initialized termination problem over integers by fixing V_1 to be the singleton initial vector and (ii) the uninitialized termination problem over integers by fixing V_1 to be the entire space $\mathbb{Z}^d$. Thus, this problem generalizes both initialized and uninitialized linear program termination problems. For e.g., the following is an instance of this problem.

Given M, V_1 (a bounded polytope), c, does the following loop terminate for all $x \in V_1$

```
while c^T x > 0 do
    x ← Mx
```

Showing decidability of this problem in general would imply the decidability of the many of these longstanding open problems, including Skolem and Positivity. Nevertheless, one may ask whether looking at this general problem allows us to prove better lower bounds or/and upper bounds in restricted cases. We remark here that if we generalize V_1 to allow

unbounded polytopes, it turns out that we can encode Petri net reachability and hence this problem is EXPSPACE-hard [6]. However, this hardness result crucially depends on the unboundedness of V_1 and does not seem to work for a bounded initial space over integers.

As before, we restrict ourselves to the subclass whose characteristic roots are all complex roots of unity (or zero). We are able to then show that for this restricted class, the problem is Π_2^P-complete, building upon our lower-bound and upper-bound techniques.

2 Preliminaries

For a complex number $z = x + iy$, the absolute value and phase of the complex number are respectively denoted by $|z| = \sqrt{x^2 + y^2}$ and $arg(z) = \tan^{-1}(\frac{y}{x})$. We denote by e_i the k-dimensional standard basis vector that has 1 at the i-th position and 0 elsewhere.

2.1 Linear Recurrence Sequences

We recall some definitions and basic properties of linear recurrence sequences that will be useful in the rest of the paper. For a detailed treatment, we refer the reader to the excellent text of Everest et al. [12].

▶ **Definition 1** (Linear Recurrence Sequence)**.** A sequence $\langle u \rangle = \langle u_n \rangle_{n=0}^{\infty}$ is called a *linear recurrence sequence* (LRS) of *order* k if k is the smallest positive integer such that the n^{th} term of the sequence can be expressed as $u_n = a_{k-1}u_{n-1} + \ldots + a_1 u_{n-k-1} + a_0 u_{n-k}$, for every $n \geq k$, where $a_j \in \mathbb{Z}$ for $j \in \{0, 1, \ldots, k-1\}$ and $a_0 \neq 0$. Such a sequence is uniquely determined by the *initial conditions* $u_0, u_1, \ldots, u_{k-1}$.

An LRS $\langle u \rangle = \{u_n\}_{n=0}^{\infty}$ is said to be *periodic* with period p if $u_n = u_{n+p}$ for every $n \geq 0$. For a linear recurrence sequence $\langle u \rangle$ of order k, we denote by $||u||$, the size of the bit representation of the coefficients of $\langle u \rangle$, namely $a_0, a_1, \ldots, a_{k-1}$ and the initial conditions $u_0, u_1, \ldots, u_{k-1}$.

To every such recurrence sequence $\langle u \rangle$ above, one can associate a univariate polynomial $\chi_u(x) = x^k - a_{k-1}x^{k-1} - a_{k-2}x^{k-2} - \ldots - a_1 x - a_0$ of degree at most k. $\chi_u(x)$ is called the *characteristic polynomial* of the recurrence $\langle u \rangle$. The roots of the characteristic polynomial are called the *characteristic roots* and they yield useful information about the asymptotic behavior of the recurrence. More formally, let $\{\lambda_1, \lambda_2, \ldots, \lambda_d\}$ be the roots of $\chi_u(x)$ with multiplicity $\rho_1, \rho_2, \ldots, \rho_d$ respectively. Then the n^{th} term of such an LRS $\langle u \rangle$, denoted u_n can be expressed as

$$u_n = \sum_{j=1}^{d} q_j(n)\lambda_j^n \tag{1}$$

where $q_j(x) \in \mathbb{C}[x]$ are univariate polynomials with complex coefficients of degree at most $\rho_j - 1$ such that $\sum_{j=1}^{d} \rho_j = k$. We say an LRS is *simple* when for every j, $\rho_j = 1$. Equivalently, for a simple LRS for every j, $q_j \in \mathbb{C}$ is a constant.

Given an LRS $\langle u \rangle$ of order k with characteristic polynomial $\chi_u(x) = x^k - a_{k-1}x^{k-1} - a_{k-2}x^{k-2} - \ldots - a_1 x - a_0$, one can associate a $k \times k$ matrix M_u called the *companion matrix* of the recurrence as shown in the following figure.

$$M_u^T = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ a_0 & a_1 & a_2 & \cdots & a_{k-1} \end{bmatrix}$$

Given a vector v of dimension $k \times 1$ containing the k initial conditions of the recurrence, one can easily observe that $u_1 = e_1 M_u^T v$ and $e_1 (M_u^T)^n v$ gives u_n. Further, the eigenvalues of this matrix are exactly the roots of the characteristic polynomial of the LRS. In what follows, we sometimes abuse notation and call them as eigenvalues of the LRS. It is often useful to rewrite each λ_j in polar coordinates as $r_j e^{i\theta_j}$. In this representation, the n^{th}-term of the sequence is given by

$$u_n = \sum_{j=1}^{d} q_j(n) r_j^n e^{in\theta_j} \tag{2}$$

The following folklore lemma says that the sum and product of two LRS is an LRS (see for example, Theorem 4.1[12]). It is also known that the resulting LRS is constructible in P.

▶ **Lemma 2.** *Let $\langle u_1 \rangle, \ldots, \langle u_\ell \rangle$ be LRS of order $k_1, \ldots k_\ell$ respectively and let $\chi_{u_1}(x), \ldots, \chi_{u_\ell}(x)$ be their respective characteristic polynomials. Then the following properties hold:*

1. *$\langle u \rangle = \sum_{i=1}^{\ell} \langle u_i \rangle$ is also an LRS of order at most $\sum_{i=1}^{\ell} k_i$. Moreover, $\chi_u(x)$ is a factor of $\prod_{i=1}^{\ell} \chi_{u_i}(x)$.*
2. *$\langle u \rangle = \langle u_1 \rangle \langle u_2 \rangle$ is also an LRS of order (and $\chi_u(x)$ is of degree) at most $k_1 k_2$.*

It is an easy observation via Lemma 2, that the complement of the Skolem problem reduces to the Positivity problem (since $u_n \neq 0$ if and only if $u_n^2 > 0$ for all n).

2.2 Algebraic Numbers

We will extensively use algebraic numbers and their properties throughout the paper. We refer the reader to the excellent text by Cohen [11] for a extensive treatment of the computational aspects of algebraic number theory. Here we collect below some useful definitions and facts that are used throughout the rest of the paper.

A complex number α is called *algebraic* if there is a unique univariate polynomial $p_\alpha(x)$ with integer coefficients of minimum degree that vanishes at α. p_α is said to be the *defining polynomial* or the *minimal polynomial* of the algebraic number α. The *degree* and *height* of α are then the degree and height of p_α (Height of a polynomial is the maximum value of its coefficients). The roots of p_α are called the *Galois conjugates* of α.

▶ **Definition 3** (Roots of unity)**.** A complex number r is an n-th root of unity if $r^n = 1$ and a *primitive* n-th root of unity if in addition, n is the smallest $k \in \{1, \ldots, n\}$ for which $r^k = 1$.

▶ **Definition 4** (Cyclotomic polynomial)**.** The n-th Cyclotomic polynomial, denoted $\Phi_n(x)$ is the unique monic irreducible (over $\mathbb{Q}$) polynomial with integer coefficients that is a divisor of $x^n - 1$ and is not a divisor of $x^k - 1$ for any $k < n$. Its roots are all the n-th primitive roots of unity. Formally, $\Phi_n(x) = \prod_{\substack{1 \leq k \leq n \\ gcd(k,n)=1}} \left(x - e^{\frac{i2\pi k}{n}}\right)$.

An important relation involving Cyclotomic polynomials (See for example [5]) and primitive n-th roots of unity is that

$$x^n - 1 = \prod_{1 \le k \le n} \left(x - e^{\frac{i2\pi k}{n}}\right) = \prod_{d|n} \prod_{\substack{1 \le k \le n \\ gcd(k,n)=1}} \left(x - e^{\frac{i2\pi k}{n}}\right) = \prod_{d|n} \Phi_{n/d}(x) = \prod_{d|n} \Phi_d(x)$$

The degree of $\Phi_n(x)$ (which is also precisely the number of n-th roots of unity) is $\phi(n)$ where ϕ is Euler's totient function, $\phi(n) = |\{k \mid k \le n, \gcd(k,n) = 1\}$. An expression for $\phi(n)$ is given by $\phi(n) = n\prod_{p|n}(1 - \frac{1}{p})$. For $n \ge 2$, $\phi(n) \ge \sqrt{\frac{n}{2}}$.

2.3 Problem statements

We now formally define three problems of interest on LRS. We will define a fourth all-encompassing problem in Section 4.

▶ **Definition 5** (Skolem, Positivity, Ultimate Positivity)**.** Given a LRS $\langle u \rangle$,

- *Skolem Problem*: Decide if there exists an $n \in \mathbb{N}$ such that $u_n = 0$.
- *Positivity problem*: Decide if $u_n > 0$ for all $n \in \mathbb{N}$.
- *Ultimate Positivity problem*: Decide if there exists $n_0 \in \mathbb{N}$ s.t., $u_n > 0$ for all $n > n_0$.

As mentioned in the introduction, we consider restriction of these problems to linear recurrent sequences with special properties, namely, the roots of their characteristic polynomial (also called characteristic roots henceforth) are complex roots of unity. We denote by Skolem_ω Pos_ω UPos_ω the Skolem, Positivity and Ultimate Positivity questions respectively for linear recurrence sequences whose characteristic roots are roots of unity.

3 LRS with roots of unity – an NP-complete subclass

In this section, we consider the special subclass of linear recurrence sequences, whose characteristic roots are exactly roots of unity, and show that the Skolem problem restricted to this subclass is NP-complete.

First, we show that Skolem_ω is NP-hard, UPos_ω and Pos_ω are coNP-hard for this class. This immediately shows NP-hardness (respectively coNP-hardness) for the Skolem problem (respectively Positivity and Ultimate Positivity) for LRS of unbounded order.

▶ **Theorem 6.** Skolem_ω *is* NP*-hard and* Pos_ω *and* UPos_ω *are* coNP*-hard.*

Proof. We show a reduction from Subset Sum to Skolem_ω. Denote by $\mathsf{SUBSETSUM}(A, T)$ the following instance of the Subset Sum problem: We have a set $A = \{a_1, a_2, \ldots, a_m\}$ where $a_i \in \mathbb{N}$ and a target $T \in \mathbb{N}$. Now, $\mathsf{SUBSETSUM}(A, T) = 1$ if there exists a subset S of A whose sum is equal to T. The *Subset Sum problem*, i.e., given A, T deciding whether $\mathsf{SUBSETSUM}(A, T) = 1$, is a classic NP-complete problem [15]. We will now construct a linear recurrence sequence $\langle u_{A,T} \rangle$ over integers which has a zero i.e., there exists r such that $u_r = 0$, iff there is a set $S \subseteq T$ such that $\sum_{s \in S} a_s = T$.

We construct the LRS as follows: For every $i \in \{1, \ldots, m\}$, let p_i be the i-th-prime. Then, for each i, we have a LRS $\langle u^i \rangle$, whose n-th term is defined as

$$u^i_n = \begin{cases} 0 & \text{if } 1 \le n < p_i \\ a_i & \text{if } n = p_i \\ u^i_{n-p_i} & \text{otherwise} \end{cases}$$

This is a periodic LRS of period p_i, where the first p_i terms are $(0, 0, \ldots, a_i)$. Note that the order (and hence the degree of the characteristic polynomial) of this LRS is p_i.

We are now ready to define LRS $\langle u_{A,T} \rangle$: the n^{th} term of $\langle u_{A,T} \rangle$ is given by $\langle u_{A,T} \rangle_n = \sum_{i=1}^{m} u_n^i - T$. Since the sum of LRS is also an LRS (by Lemma 2), $u_{A,T}$ is also an LRS. Now, by the prime number theorem (see [13] for instance), it follows that the number of primes less than $n \in \mathbb{N}$ asymptotically grows as $\frac{n}{\log(n)}$, which implies that the n^{th} prime number is of magnitude $\mathcal{O}(n \log n)$. Thus, from this and Lemma 2, it follows that the order (and hence also the degree of the characteristic polynomial) of $u_{A,T}$ is at most $(\sum_{i=1}^{m} p_i) + 1 = \mathcal{O}(m^2 \log m)$ and $u_{A,T}$ can be constructed from a given instance of $\mathsf{SUBSETSUM}(A, T)$ in polynomial time. We have the following

▶ **Claim 7.** *$\langle u_{A,T} \rangle$ is zero if and only if there exists a subset $S \subseteq [m]$ such that $\sum_{s \in S} a_s = T$.*

Proof. Suppose there exists r such that $u_r = 0$. Then $\sum_{i=1}^{m} u_r^i = S$. As u_n^i can either be a_i or 0, this implies that there exists a subset $S \subseteq [m]$ such that $\sum_{s \in S} a_s = T$. This proves the forward direction of the claim. For the other direction, let us suppose there exists a subset $S \subseteq [m]$ such that $\sum_{s \in S} a_s = T$. Consider $N = \prod_{s \in S} p_s$. Then it is easy to see that u_N is precisely $\sum_{s \in S} a_s - T = 0$. ◀

This completes the NP-hardness of Skolem problem. The coNP-hardness of Positivity follows from noting that the square of a linear recurrence is also a linear recurrence. The complement of the Skolem problem reduces to the Positivity problem because $u_n \neq 0$ if and only if $u_n^2 > 0$ for all n. A closer observation of the proof of hardness yields the following important property of $u_{A,T}$: *the roots of $\chi_{u_{A,T}}$ are roots of unity.* This follows by observing that the characteristic polynomial of $\langle u^i \rangle$ is precisely $x^{p_i} - 1 = 0$. Hence all its roots are the p_i-th roots of unity. Now by Lemma 2, $\chi_{u_{A,T}}$ is a factor of the product $(x-1)\prod_{i=1}^{m}(x^{p_i} - 1)$ (here the term $x - 1$ is contributed by the integer $-T$ in the subset sum instance). The LRS $u_{A,T}$ is hence an instance[2] of Skolem_ω.

For the positivity problem, we have to square $u_{A,T}$ and since the characteristic roots of $(u_{A,T}^2)_n = \left(\beta_0 + \sum_{j=1}^{m} \beta_j e^{i\theta_j}\right)^2 = \sum_{j,\ell \in [m]} \beta_j \beta_\ell e^{i\theta_j} e^{i\theta_\ell}$, the characteristic roots of $u_{A,T}^2$ are also roots of unity. Hence the positivity problem for the LRS derived from the subset sum is actually an instance of Pos_ω. It is easy to see that for periodic LRS, the questions of positivity and ultimate positivity are equivalent. Since $u_{A,T}$ constructed in our proof is periodic, the coNP-hardness of Pos_ω also entails the same hardness for UPos_ω. ◀

To complement the hardness result from Theorem 6, we now prove that Skolem_ω(respectively Pos_ω and UPos_ω) is decidable in NP(respectively coNP). It is worthwhile to contrast this with the case of arbitrary recurrences for which decidability is open. We have the following

▶ **Theorem 8.** *Skolem_ω is in NP, Pos_ω and UPos_ω are in coNP.*

The rest of this section will prove the above theorem. We start with some basic properties. Consider the general form of the n^{th} term of an LRS as given in Equation 2. When the eigenvalues are roots of unity, this simplifies to

$$u_n = \sum_{j=1}^{d} q_j(n) e^{in\theta_j} \tag{3}$$

[2] In fact, it is easy to transform our recurrence $u_{A,T}$ in to another recurrence $u'_{A,T}$, while maintaining the property YES and NO instances of subset sum are mapped to YES and NO instance of Skolem_ω for $u'_{A,T}$, such that $u'_{A,T}$ is also a simple LRS.

where q_j are polynomials of degree at most $k-1$ and $\sum_{j=1}^{d}(\deg(q_j)+1) = k$ and $\theta_j = \frac{a_j 2\pi}{b_j}$ as roots of characteristic equation are roots of unity. In order to prove an NP upper bound, it suffices to show that there exists an $N \leq 2^{\mathsf{poly}(||u||)}$ such that if u is zero at all, then $u_N = 0$ and this can verified in P. Recall that $||u||$ denotes the size of the bit representation of the coefficients of u_n, namely $a_0, a_1, \ldots, a_{k-1}$ and the initial conditions $u_0, u_1, \ldots, u_{k-1}$. We first note a few easy observations about the characteristic roots:

▶ **Proposition 9.** *If $e^{i\theta_j}$ is a characteristic root of multiplicity ρ_j of an LRS of order k, with $\theta_j = \frac{2\pi a_j}{b_j}$, $gcd(a_j, b_j) = 1$ then*

- *For any $1 \leq a < b_j$ such that $gcd(a, b_j) = 1$, and $\theta = \frac{2\pi a}{b_j}$, $e^{i\theta}$ is also a root with multiplicity ρ_j.*
- *$\rho_j b_j \leq k^3$*

This implies that the characteristic roots can be partitioned into multisets $\vartheta_j = \{e^{\frac{i2\pi a}{b_j}} \mid gcd(a, b_j) = 1\}$ and $|\vartheta_j| = \rho_j \phi(b_j)$, where ϕ is Euler's totient function.

Proof. The elements in multiset ϑ_j are exactly $e^{i\theta_j}$ and their Galois conjugates hence they must all occur, with same multiplicity. The cardinality of such numbers is exactly $\phi(b_j) \geq \sqrt{\frac{b_j}{2}}$ (where ϕ is Euler's totient function). Since the number of roots is k and each element in ϑ_j occurs ρ_j times, we obtain that each $\rho_j \phi(b_j) \leq k$. As $\phi(b_j) \geq \sqrt{\frac{b_j}{2}}$ we get $\rho_j b_j \leq k^3$. ◀

The solution set of Skolem_ω instances are very structured given the fact that the characteristic roots are roots of unity. Consider for each $m \in \mathbb{N}$, a polynomial $P_m = \sum_{j=1}^{d} q_j(x) e^{im\theta_j}$ and let P denote the set of polynomials $\{P_m \mid m \in \mathbb{N}\}$. We have the following

▶ **Lemma 10.** *The set P is finite i.e., $P = \{P_m \mid m \in [0, k^{3k}]\}$ where $k^{3k} < 2^{\mathsf{poly}(||u||)}$.*

Proof. When characteristic roots are roots of unity, each θ_j is of the form $\frac{2\pi a_j}{b_j}$ for some (positive) integers $a_j \leq b_j$. Now $b_j \leq k^3$ by Proposition 9. Each $e^{im\theta_j} = e^{i(m+b_j)\theta_j}$ for all m, i.e., they repeat after b_j steps. Hence $P_m = P_{m+t}$ for $t = \mathrm{lcm}\{b_1, \ldots b_d\}$. Therefore $|P| \leq t = \mathrm{lcm}\{b_1, \ldots, b_d\} \leq b_1 \cdots b_d \leq k^{3k} \leq 2^{k^4}$. Since $k < \mathsf{poly}(||u||)$, the result follows. ◀

Note that even though q_j could be polynomials with complex coefficients, the coefficients of polynomials in P are rational. This is because, all the polynomials P_m evaluate to integer values at infinitely many integers via Equation 3, since the recurrence always evaluates to integer values). By interpolation, these coefficients have to be rational.

Hence deciding Skolem_ω essentially boils down to finding the union of zero sets of all the polynomials in P. This naturally leads us to the problem of bounding the coefficients of polynomials in P since this immediately yields a bound on the roots. A natural way to proceed here would be to use interpolation to bound the coefficients (see e.g., [14]). The problem with this approach is that this yields an expression for the coefficients q_{kj} of the polynomials q_j in terms of linear combinations of λ_j, which are algebraic numbers. Standard techniques (see for example, the work of Tiwari on the sign problem [24]) however, do not yield a lower bound which is exponential in d, the degree of the roots. Thus, while this suffices to obtain an NP upper bound for LRS of fixed order (where d becomes constant), for the case of unbounded order LRS, it does not yield an NP upper bound. In the next two lemmas, we show how to sidestep this issue, by crucially exploiting the fact that our characteristic roots are roots of unity.

First, note that χ_u can be written as

$$\chi_u(x) = \prod_{j=1}^{d}(x - e^{2\pi i \frac{a_j}{b_j}})^{\rho_j} = \prod_{j=1}^{D} \prod_{\substack{1\leq a\leq b_j \\ gcd(a,b_j)=1}} (x - e^{2\pi i \frac{a}{b_j}})^{\rho_j} \tag{4}$$

where D is the number of distinct values of b_j. We know from Lemma 2 that the sum of LRS is again a LRS. We obtain here a partial converse of part 1 of Lemma 2.

▶ **Lemma 11.** *Let $\langle u\rangle$ be a LRS with characteristic polynomial $\chi_u(x) = p_1(x)p_2(x)$ where p_1 and p_2 do not share a common root. Then we can find LRS $\langle u^1\rangle$ and $\langle u^2\rangle$ with characteristic polynomials p_1 and p_2 such that $\langle u\rangle = \langle u^1\rangle + \langle u^2\rangle$.*

Proof. We know that $u_n = \sum_{j=1}^{d} q_j(n)\lambda_j^n$. Let $R(p)$ denote the set of roots of polynomial $\chi_u(x)$. Since p_1 and p_2 do not share a common root i.e. $R(p_1) \cap R(p_2)$ is empty and $R(p_1) \cup R(p_2) = R(p)$, we can rewrite the exponential polynomial solution from Equation 1 as $u_n = \sum_{\lambda_j \in\ R(p_1)} q_j(n)\lambda_j^n + \sum_{\lambda_j \in\ R(p_2)} q_j(n)\lambda_j^n$.

Let us consider the set of LRS defined by the characteristic polynomial p_1 (by fixing all possible initial conditions). This is a vector space and one can see that the set $\{n^i\lambda_j^n : \lambda_j \in\ R(p_1), 0 \leq i \leq$ multiplicity of λ_j in $p_1\}$ describes a basis for this vector space. As $\sum_{\lambda_j \in\ R(p_1)} q_j(n)\lambda_j^n$ is just a linear combination of such terms, it is also a possible LRS with characteristic polynomial p_1, let us call this u_n^1. Similarly $\sum_{\lambda_j \in\ R(p_2)} q_j(n)\lambda_j^n$ defines an LRS u_n^2. Hence u_n can be written as $u_n^1 + u_n^2$. ◀

As none of the inner products in Equation 4 share a root by Lemma 11 we can break the linear recurrence as a sum of linear recurrences. Let $\langle u\rangle = \langle u^1\rangle + \ldots + \langle u^D\rangle$ where the characteristic polynomial of $\langle u^j\rangle$ is exactly $\prod_{\substack{1\leq a\leq b_j \\ gcd(a,b_j)=1}} (x - e^{2\pi i \frac{a}{b_j}})^{\rho_j}$. This is exactly the b_j^{th} cyclotomic polynomial raised to ρ_j. Note that this is a integral polynomial with coefficients bounded by poly($||u||$)-many bits. Now, we have the following:

▶ **Lemma 12.** *The first k^3 values of all u^j are* poly($||u||$)*-bit bounded rationals and can be calculated in* P*.*

Proof. The linear recurrence for u^j has degree $\phi(b_j)\rho_j$. We think of the first $\phi(b_j)\rho_j$ initial values as variables. Fixing them fixes u^j. We can express first k^3 values of u^j as integral combinations of first $\phi(b_j)\rho_j$ values. In this integral combination the weights are poly($||u||$)-bit bounded as weight $<$ (sum of coefficients of $u^j)^{k^3}$. Next we argue that these k^3 initial values of the u^j are poly($||u||$)-bit bounded rationals. We remember that sum of all u^k is u and we have k initial values of u. We know that $\sum_{j=1}^{D} \phi(b_j)\rho_j = k$ as both LHS and RHS represent number of roots of χ_u. The initial values for these D sequences can be found by setting up a system of k linear equations in k variables and solving them where each the n^{th} equation says that $\sum_{j=1}^{D} u_n^k = u_n$. Note that for u^j only first $\phi(b_j)\rho_j$ values are variables not all k, but all of them can be represented as integral combination of first $\phi(b_j)\rho_j$ values with poly($||u||$)-bit bounded weights. Note that since the initial values of $\langle u\rangle$ are given as integers as a part of the input hence they are representable in poly($||u||$)-bit. Hence for the linear equations all coefficients are poly($||u||$)-bit bounded. Hence the initial values of the D linear recurrences are also obtainable as rationals of bit length at most polynomial in $||u||$. As any of the first k^3 values is expressible as integral combinations of first $\phi(b_j)\rho_j$ values with poly($||u||$)-bit bounded weights, hence all of first k^3 values are poly($||u||$)-bit bounded. ◀

Now as we had defined P for original linear recurrence u we can define P^j for u^j. Note that $|P^j| \leq b_j < k^3$ and hence polynomially bounded in k unlike $|P|$ for which we could only give an exponential bound. Similar to P coefficients of P^j are also rationals. The degree of any polynomial is P^j is at most the multiplicity which is ρ_j.

▶ **Lemma 13.** *Coefficients of any polynomial in P^j are* $\mathsf{poly}(||u||)$*-bit bounded rationals and can be calculated in* P.

Proof. By the definition of P^j, $u_n^k = P_q^j(n)$ when n is of the form $n = bp + q$ and $0 \leq q < b$. Now we can interpolate to get coefficients of this polynomial. We need ρ_j values to interpolate where one value occurs every b_j terms. By Proposition 9 $b_j\rho_j < k^3$ and by Lemma 12 we know that first k^3 values are $\mathsf{poly}(||u||)$-bit bounded rationals. The other coefficients in the interpolation are of the form n^i where $n < k^3$ and $i < k$ hence they are also bounded by k^{3k} ($\mathsf{poly}(||u||)$-bit). So the interpolated coefficients will also be $\mathsf{poly}(||u||)$-bit bounded rationals. ◀

We are now ready to prove Theorem 8:

Proof. (of Theorem 8) First, notice that any n such that $u_n = 0$, n is a root of one of the polynomials in P. For any of these polynomials the coefficient is the sum of corresponding coefficients in P^j's, which are $\mathsf{poly}(||u||)$-bit bounded rationals by Lemma 13. Hence their sum i.e. coefficient of any polynomial in P is also $\mathsf{poly}(||u||)$-bit bounded and can be calculated P. Note that as mentioned above, this property also does not hold for arbitrary algebraic numbers. Now as the coefficients of all the polynomials in P can be represented by rational numbers in $\mathsf{poly}(||u||)$ bits hence their roots are bounded in magnitude by $2^{\mathsf{poly}(||u||)}$ (unless one of the polynomials is identically 0). As we are only interested in integer roots, this implies that any integer root n of a non-zero polynomial in P can be written in $\mathsf{poly}(||u||)$ bits. For a zero polynomial $P_m \in P$, at $n = m$ $u_n = P_m(m) = 0$ hence $n = m$ is a zero and can be written in $\mathsf{poly}(||u||)$ bits by Lemma 10. In both cases n is therefore a short ($\mathsf{poly}(||u||)$-bit bounded) certificate for the Skolem problem, guessed by an NP machine.

Now observe that for such a n, $u_n = P_m(n)$. As both the coefficients of P_m and n are $\mathsf{poly}(||u||)$-bit bounded hence u_n is also $\mathsf{poly}(||u||)$-bit bounded. Hence the guessed n can be verified in P by observing that $u_n = e_1(M_u^T)^n v$, where M_u^T is the corresponding companion matrix. $e_1(M_u^T)^n v$ can be calculated in P via repeated squaring : Since the companion matrix M also satisfies the characteristic polynomial of the recurrence by Cayley-Hamilton theorem, its entries satisfy the recurrence u_n. Hence the preceding argument that u_n is $\mathsf{poly}(||u||)$-bit bounded, also works for each of these entries of the $(M_u^T)^n$. This proves that $\mathsf{Skolem}_\omega$is in NP. To see that $\mathsf{Pos}_\omega$is in coNP note that we need the following 2 conditions to ensure positivity:

- Since the zeros of all the polynomials in the set P (which is also exponentially bounded in size) lie in the range $[0, 2^{\mathsf{poly}(||u||)}]$, it suffices to check that for all the polynomials evaluated at all the points in this range evaluate to a positive value
- All polynomials in set P are ultimately positive.

For condition 1 we need to ensure $u_n \neq 0$ for all $n \in [1, 2^{\mathsf{poly}(||u||)}]$. As u_n can be calculated in P for such an n, this can be implemented in coNP. We can ensure condition 2 for a P_m by making sure that the first non-zero coefficient in positive. By Lemma 10 we just need to ensure this for $m \in [1, 2^{\mathsf{poly}(||u||)}]$ but as coefficients for any m can be calculated in P we can implement this check also in coNP. Hence Pos_ω is in coNP. UPos_ω requires us to just check condition 2, hence it is also in coNP. ◀

Combining the above theorem with Theorem 6, we obtain our completeness results as stated, i.e., if all characteristic roots of an LRS are roots of unity, then Skolem (resp. Positivity, Ultimate Positivity) for such recurrences is NP-complete (resp. coNP-complete).

4 Integer Polytope Containment Problem

In this section, we consider a new problem on dynamical systems. We start by fixing some notation. A *(convex) polytope* is an intersection of finitely many half-spaces. A polytope is said to be *bounded* if the region enclosed in it is bounded.

▶ **Definition 14** (Integer Polytope Containment Problem)**.** Given a bounded polytope $V_1 \subset \mathbb{Z}^d$ and a (possibly unbounded) polytope $V_2 \subseteq \mathbb{Z}^d$ and a $d * d$ matrix M with integer entries, the Polytope Containment Problem asks if for all $v \in V_1$ (for $v \in \mathbb{Z}^d$), does there exist a positive integer n such that $vM^n \in V_2$.

As before, we restrict ourselves to a subclass of this problem, where the eigenvalues are all complex roots of unity.

▶ **Definition 15** ($\mathsf{Contain}_\omega$)**.** $\mathsf{Contain}_\omega$ is the subclass of Polyhedron Containment Problem when the corresponding matrix M has roots of unity and 0 as eigenvalues.

▶ **Theorem 16.** $\mathsf{Contain}_\omega$ *is* Π_2^P*-hard.*

For definitions of coNP, $\Pi_2^P$and other standard complexity classes, we refer the reader to the excellent text due to Arora and Barak [4]. Interestingly, our proof can be seen as an application or generalization of our earlier technique to obtain the reduction of the Skolem problem from the Subset Sum problem. Indeed, since we use our NP-hard instance of the Skolem for this reduction, it is not clear how we can do a similar lift from the earlier NP-hardness proof of Blondel-Portier [7], or indeed any other existing approach.

The rest of this section forms the proof of the above theorem. We start by considering the following generalized form of the subset sum (GSS) problem, which is known to be Π_2^P-complete [22]. Given two vectors $b = (b_1, \ldots b_\ell)$ and $a = (a_1, \ldots a_m)$ and $\alpha \in \mathbb{Z}$, for all $x \in \{0,1\}^\ell$, does there exist $y \in \{0,1\}^m$ such that $x \cdot b + y \cdot a = \alpha$?[3] We will also use the set notation $B = \{b_1, \ldots b_\ell\}$ and $A = \{a_1, \ldots, a_m\}$ when convenient.

Our goal is to reduce this above problem to $\mathsf{Contain}_\omega$. In order to do so, we will use the Subset-sum to Skolem_ω reduction from Section 3. Consider the LRS u_A whose n^{th} term is $\sum_{i=1}^{m} u_n^i$, where u_n^i is the LRS constructed in Section 3. We can observe the following properties about this LRS:

(F1) Each entry of u_A gives a sum of a subset of elements from A, i.e., $y \cdot a$ for some $y \in \{0,1\}^m$.

(F2) Every subset of sum of elements of A occurs as some entry of u_A.

(F3) The LRS u_A is periodic, i.e., the elements repeat after a certain period (product of the first m-primes to be precise). Thus the bound that they repeat after or the period is exponentially bounded.

(F4) The LRS u_A can be written in matrix form as a matrix M_A such that for all $n \geq 0$, $\langle u_A \rangle_{n+1} = vM^n w$ where v is the first m entries of the LRS and $w = (1, 0, \ldots 0, 0)$. Further, the roots of the characteristic polynomial of u_A (which were noted earlier to be roots of unity) are the eigenvalues of M_A. The proof of these facts follows by simple linear algebra and can be found for instance in [25]. Further, we may also observe that

[3] In fact, to be precise, [22] defines the complement of this problem.

the entries in the matrix are exponentially bounded in the input-size of the subset-sum instance and can be computed in poly-time. It follows that the sequence of numbers vM^nw for all $n \geq 0$ satisfy the three properties (F1–F3) listed above.

Given an instance of GSS problem, we will build an instance of $\mathsf{Contain}_\omega$as follows. Define a square matrix G of dimension $\ell+m+2$, as shown below. Note that the eigenvalues of G are all roots of unity and 0. This follows from the fact that G is a block upper triangular matrix hence $det(G - \lambda I) = det(I_{(\ell+1)\times(\ell+1)} - \lambda I_{(\ell+1)\times(\ell+1)})det(M_{m\times m} - \lambda I_{m\times m})det(-\lambda)$, which implies that the eigenvalues are 1, eigenvalues of M and 0. We fix V_1 to be the set of all vectors $\{(x_1, \ldots x_\ell, 1, v, 0) \mid x_i \in \{0,1\}, \text{ for all } 1 \leq i \leq \ell\}$. Note that this is a polytope, i.e., an intersection of half spaces. Next, we fix V_2 to be the polytope $\{y \in \mathbb{Z}^{\ell+m+2} \mid y \cdot e_{\ell+m+2} = 0\}$, where $e_{\ell+m+2}$ is the $\ell + m + 2$-dimension vector $(0, \ldots 0, 1)$.

$$G = \left[\begin{array}{c|c|c} I_{(\ell+1)\times(\ell+1)} & 0 & \begin{matrix} b_1 \\ b_2 \\ \vdots \\ b_\ell \\ -\alpha \end{matrix} \\ \hline 0 & M_{m\times m} & w \\ \hline 0 & 0 & 0 \end{array}\right]$$

Let $z = (x, 1, v, 0) \in V_1$ for some $x \in \{0,1\}^\ell$. By induction, we obtain that for all $n \geq 1$, $z \cdot G^n = (x, 1, v \cdot M_A^n, x \cdot b - \alpha + v \cdot M_A^{n-1} \cdot w)$. Now GSS has a solution iff for all $x \in \{0,1\}^\ell$, there exists $y \in \{0,1\}^m$ such that $x \cdot b + y \cdot a = \alpha$. From facts above, it follows that for each such $y \in \{0,1\}^m$, there exists $n \geq 0$ such that $y \cdot a = vM^nw$. In other words, GSS has a solution iff for all $x \in \{0,1\}^\ell$, there exists $n \geq 0$ such that $x \cdot b + vM^nw = \alpha$. That is, GSS has a solution iff for all $z \in V_1$, there exists $n \geq 0$ such that $z \cdot G^{n+1} \cdot e_{\ell+m+2} = 0$ iff for all $z \in V_1$, there exists $n' \geq 1$ such that $z \cdot G^{n'} \in V_2$. This gives the solution for our instance of the $\mathsf{Contain}_\omega$problem and completes the proof of correctness of the reduction.

From this we immediately obtain, that the Integer Polytope Containment Problem is Π_2^P hard. Finally, we will show that:

▶ **Theorem 17.** $\mathsf{Contain}_\omega$ *is* Π_2^P*-complete.*

Proof. We have already shown hardness for $\mathsf{Contain}_\omega$ in Theorem 16 so now we only need to show inclusion in Π_2^P. Description size of any integer value $x \in V_1$ i.e. $||x||$ is $\mathsf{poly}(||V_1||)$ where V_1 is a bounded polytope and is specified as intersection of hyperplanes. This because the corner points are solutions of linear equations which is bounded by $\mathsf{poly}(||V_1||)$-bits. Hence all integral points inside are also bounded by $\mathsf{poly}(||V_1||)$-bits. As checking if a particular $x \in V_1$ can be done P we can go over all $x \in V_1$ in coNP. After fixing x, this the problem reduces to Skolem_ω which, by Theorem 8, is in NP. Hence $\mathsf{Contain}_\omega$ is in $\mathsf{coNP}^{\mathsf{NP}} = \Pi_2^P$. ◀

While our hardness results lift to the non-integer case, our containment proofs do not extend immediately to the non-integer case. However, we conjecture that using techniques from [8], we can obtain similar decidability results, for this subclass for rational/real cases.

5 Conclusion

In this paper, we investigate linear recurrence sequences whose characteristic roots which are complex roots of unity. We show that the Skolem problem (resp. positivity, ultimate positivity) restricted to this subclass of LRS is NP-complete (resp. coNP-complete). The lower bound is via a novel reduction from subset sum, which we are also able to extend to show Π_2^P-hardness for a more general yet interesting problem on LRS. Note that this lower bound (as well as the one in [7]), requires LRS to be of arbitrary or unbounded orders. One interesting open question is whether one could show any non-trivial lower bound (e.g., NP-hardness) for LRS of a fixed order.

Our approach for upper-bounds can be extended further to tackle LRS whose characteristic roots are complex roots of any real number, i.e., complex numbers whose phases are rational multiples of π. However, we get more relaxed upper-bounds, without matching lower-bounds. While disappointing, this is not surprising since any improvement in the lower-bound would be a highly remarkable result as commented in the conclusion of [21].

Acknowledgements. We would like to thank the anonymous reviewers for their insightful comments and helpful remarks.

References

1 Manindra Agrawal, S. Akshay, Blaise Genest, and P. S. Thiagarajan. Approximate verification of the symbolic dynamics of markov chains. *J. ACM*, 62(1):2:1–2:34, 2015.

2 S. Akshay, Timos Antonopoulos, Joël Ouaknine, and James Worrell. Reachability problems for markov chains. *Inf. Process. Lett.*, 115(2):155–158, 2015.

3 S. Akshay, Blaise Genest, Bruno Karelovic, and Nikhil Vyas. On regularity of unary probabilistic automata. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 8:1–8:14, 2016.

4 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. URL: `http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264`.

5 M. Artin. *Algebra*. Pearson Prentice Hall, 2011. URL: `https://books.google.de/books?id=QsOfPwAACAAJ`.

6 Amir M. Ben-Amram, Samir Genaim, and Abu Naser Masud. On the termination of integer loops. *ACM Trans. Program. Lang. Syst.*, 34(4):16:1–16:24, December 2012.

7 V. D. Blondel and N. Portier. The presence of a zero in an integer linear recurrent sequence is NP-hard to decide. In *Linear Algebra and its Applications*, pages 351–352. Elsevier, 2002.

8 Mark Braverman. Termination of integer linear programs. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 372–385, 2006.

9 Ventsislav Chonev, Joël Ouaknine, and James Worrell. The polyhedron-hitting problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 940–956, 2015. `doi:10.1137/1.9781611973730.64`.

10 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the complexity of the orbit problem. *J. ACM*, 63(3):23:1–23:18, 2016. `doi:10.1145/2857050`.

11 Henri Cohen. *A course in computational algebraic number theory*, volume 138. Springer Science & Business Media, 2013.

12 Graham Everest, Alfred J. van der Poorten, Igor E. Shparlinski, and Thomas Ward. *Recurrence Sequences*, volume 104 of *Mathematical surveys and monographs*. American Mathematical Society, 2003. URL: http://www.ams.org/bookstore?fn=20&arg1=survseries&item=SURV-104.

13 Godfrey H. Hardy and Edward M. Wright. *An introduction to the theory of numbers (5. ed.)*. Clarendon Press, 1995.

14 Dan Kalman. The generalized vandermonde matrix. *Mathematics Magazine*, 57(1):15–21, 1984.

15 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

16 Joël Ouaknine, João Sousa Pinto, and James Worrell. On termination of integer linear loops. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 957–969, 2015. doi:10.1137/1.9781611973730.65.

17 Joël Ouaknine and James Worrell. Decision problems for linear recurrence sequences. In *Reachability Problems - 6th International Workshop, RP 2012, Bordeaux, France, September 17-19, 2012. Proceedings*, pages 21–28, 2012. doi:10.1007/978-3-642-33512-9_3.

18 Joël Ouaknine and James Worrell. On the positivity problem for simple linear recurrence sequences,. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 318–329, 2014. doi:10.1007/978-3-662-43951-7_27.

19 Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 366–379, 2014. doi:10.1137/1.9781611973402.27.

20 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 330–341, 2014. doi:10.1007/978-3-662-43951-7_28.

21 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *SIGLOG News*, 2(2):4–13, 2015.

22 Marcus Schaefer and Christopher Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT news*, 33(3):32–49, 2002.

23 Ashish Tiwari. Termination of linear programs. In *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, pages 70–82, 2004.

24 Prasoon Tiwari. A problem that is easier to solve on the unit-cost algebraic RAM. *J. Complexity*, 8(4):393–397, 1992. doi:10.1016/0885-064X(92)90003-T.

25 M.Hirvensalo V.Halava, T.Harju and J.Karhumäki. Skolem's problem on the border between decidability and undecidability. In *TUCS Technical Report Number 683*, 2005.

Being Even Slightly Shallow Makes Life Hard*

Irene Muzi[1], Michael P. O'Brien[2], Felix Reidl[3], and Blair D. Sullivan[4]

1 **University of Rome, "La Sapienza", Rome, Italy**
irene.muzi@gmail.com
2 **North Carolina State University, Raleigh, USA**
mpobrie3@ncsu.edu
3 **North Carolina State University, Raleigh, USA**
felix.reidl@gmail.com
4 **North Carolina State University, Raleigh, USA**
blair_sullivan@ncsu.edu

Abstract

We study the computational complexity of identifying dense substructures, namely *r/2-shallow topological minors* and *r-subdivisions*. Of particular interest is the case $r = 1$, when these substructures correspond to very localized relaxations of subgraphs. Since Densest Subgraph can be solved in polynomial time, we ask whether these slight relaxations also admit efficient algorithms.

In the following, we provide a negative answer: Dense r/2-Shallow Topological Minor and Dense r-Subdivision are already NP-hard for $r = 1$ in very sparse graphs. Further, they do not admit algorithms with running time $2^{o(\mathbf{tw}^2)}n^{O(1)}$ when parameterized by the treewidth of the input graph for $r \geqslant 2$ unless ETH fails.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Topological minors, NP Completeness, Treewidth, ETH, FPT algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.79

1 Introduction

Many structural graph theory results stem from the principle that excluding certain (dense) substructures affords algorithmic tractability [7, 8, 12, 13, 11, 16, 9, 20]. It is therefore natural to ask for each of these substructures whether computing its densest occurrence is efficiently possible. In this paper, we characterize the complexity of this problem for substructures formed by contracting short disjoint paths.

There has been significant recent work on a slightly more general substructure, r-shallow minors, which are formed by contracting disjoint connected *subgraphs* of radius at most r. Shallow minors form a gradient between the locality of subgraphs (0-shallow minors) and the global nature of (∞-shallow) minor containment. Because finding r-shallow minors of density/degeneracy at least d is polynomial time solveable at $r = 0$ [15, 14] but NP-complete at $r = \infty$ [3], one might expect the problem to be fixed-parameter tractable with respect to r. However, Dvořák proved that for any fixed $r > 0$ both variations are NP-complete already in graphs of maximum degree four and $d \geqslant 4$ ($d \geqslant 2$ if degeneracy is the measure) [10]. Accordingly, a parameterization by d also cannot possibly yield an fpt-algorithm—a sharp

* This work supported in part by the DARPA GRAPHS Program and the Gordon & Betty Moore Foundation's Data-Driven Discovery Initiative through Grants SPAWAR-N66001-14-1-4063 and GBMF4560 to Blair D. Sullivan.

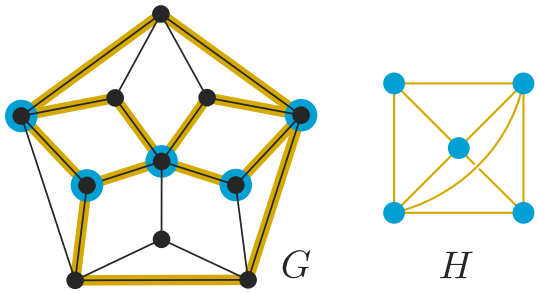


Figure 1 The graph H is a 1-shallow topological minor of G, as witnessed by the model marked with blue nails and golden paths.

contrast to unrestricted minors [3]. Dvořák further showed that the problem is in FPT if parameterized by the treewidth **tw** of the input graph and designed an $O(4^{\mathbf{tw}^2}n)$ dynamic programming algorithm.

We consider whether more conservative relaxations of subgraph containment remain efficiently solvable, specifically $r/2$-shallow *topological* minors and *r-subdivisions*. A graph H is an $r/2$-shallow topological minor of a graph G if a $(\leqslant r)$-subdivision of H is isomorphic to some subgraph of G. The case of $r = 1$ is of particular interest because it generalizes subgraphs to a proper subset of 1-shallow minors.

We show that Dense $r/2$-Shallow Topological Minor (Dense $r/2$-STM) and Dense r-Subdivision (Dense r-SD) are NP-complete already in subcubic apex-graphs[1] for $r \geqslant 1$ via a reduction from Positive 1-in-3SAT. Accordingly, a parameterization by the target density d does not make these problems fixed-parameter tractable. The same reduction also implies that neither problem can be solved in time $O(2^{o(n)})$ unless the ETH fails. In other words, finding dense substructures which are just slightly 'less local' than subgraphs seems to be intrinsically difficult. Following Dvořák's results, we then consider a parameterization by treewidth and ask whether an algorithm with running time better than $O(2^{\mathbf{tw}^2}n)$ is possible. Surprisingly, we can rule out such an algorithm already for Dense 1-Shallow Topological Minor: unless the ETH fails, no algorithm with running time $O(2^{o(\mathbf{tw}^2)}n)$ can exist.

2 Preliminaries

For a graph G we use $|G|$ to denote the number of vertices and $\|G\|$ to denote the number of edges in G. A graph H appears as an *r-subdivision* in a graph G if the graph obtained from H by subdividing every edge r times is isomorphic to some subgraph of G. Similarly, H is a *$r/2$-shallow topological minor*[2] of G if a graph obtained from H by subdividing every edge *up to* r times is isomorphic to a subgraph of G. In both cases, the subgraph witnessing the minor is the *model* and we call those vertices in it that correspond to (subdivided) edges *subdivision vertices* and all other vertices *nails*. If S_{uv} is the set of subdivision vertices on a subdivided uv edge, we say S_{uv} is *smoothed* into the uv edge.

The following two problems are the focus of this paper:

[1] That is, a graph in which the removal of a single vertex results in a subcubic planar graph.
[2] The somewhat cumbersome convention of letting an r-shallow topological minor contract paths of length $2r + 1$ is convenient in the broader context of sparse graph classes (*cf.* [18]).

Dense $r/2$-Shallow Topological Minor (Dense $r/2$-STM)

Input: A graph G and a rational number d.
Question: Is there an $r/2$-shallow topological minor H of G with density $\|H\|/|H| \geqslant d$?

Dense r-Subdivision (Dense r-SD)

Input: A graph G and a rational number d.
Question: Is there a graph H that is contained in G as an r-subdivision with density $\|H\|/|H| \geqslant d$?

The following variant, which we prove to be NP-complete in Section 4.1, might be of independent interest:

Dense Bipartite Subdivision

Input: A bipartite graph (X, Y, E) and a rational number d.
Question: Are there subsets $X' \subseteq X, Y' \subseteq Y$ such that all vertices in X' can be smoothed into unique edges in Y' and $|X'|/|Y'| \geqslant d$?

Our main tool will be linear reductions from the following SAT-variant:

Positive 1-in-3SAT

Input: A CNF boolean formula ψ with only positive literals.
Question: Does ϕ have a satisfying assignment such that each clause contains exactly one true variable?

Mulzer and Rote showed [17] that Positive 1-in-3SAT remains NP-hard when restricted to *planar* formulas. A formula ϕ is planar if the graph obtained from ϕ by creating one vertex for each clause and variable and connecting a variable-vertex to a clause-vertex if the clause contains said variable is planar.

Schaefer [21] provided a linear reduction from 3SAT to 1-in-3SAT. We can further easily transform a formula ϕ with negative literals into one with only positive literals as follows: for each variable x, introduce the variables x^+, x^-, a_x, b_x, c_x. Replace every occurrence of x with x^+ and every occurrence of $\bar{x}$ with x^- and add the clauses

$$\{x^+, x^-, a_x\}, \{x^+, x^-, b_x\}, \{a_x, b_c, c_x\},$$

to the formula. It is easy to verify that exactly one of x^+, x^- must be true in a 1-in-3 satisfying assignment and that the resulting formula ϕ' has size linear in $|\phi|$. In conclusion, there exists a linear reduction from 3SAT to Positive 1-in-3SAT which implies that under ETH, Positive 1-in-3SAT cannot be solved in time $2^{o(n)}(n+m)^{O(1)}$, where n is the number of variables and m is the number of clauses. Using sparsification one can further show that the ETH excludes algorithms for 3SAT with running time $2^{o(m)}(n+m)^{O(1)}$ (see *e.g.* the survey by Cygan *et al.* [5]). The above reduction implies the following lower bound:

▶ Proposition 1. Unless the ETH is false, Positive 1-in-3SAT cannot be solved in time $2^{o(m)}(n+m)^{O(1)}$.

3 Algorithmic considerations

We start with a basic observation about the problems in question with the smallest sensible depths of $r = 1$:

▶ **Lemma 1.** *The densest ½-shallow minor or* 1*-subdivision on a given set of nails can be computed in polynomial time.*

Proof. Assume we are to find the densest 1-subdivision with nail set X in a graph G. We construct an auxilliary bipartite graph $\hat{G}$ with vertex set $V(G) \setminus X$ and $\binom{X}{2}$ where the vertex $v \in V(G) \setminus X$ is connected to $xy \in \binom{X}{2}$ iff $\{x, y\} \subseteq N(v)$ in G, that is, if v can be contracted into the edges x, y. Now simply note that a matching of cardinality ℓ in $\hat{G}$ corresponds to a 1-subdivision in G with ℓ subdivisions. Finding a maximal matching in $\hat{G}$ therefore provides us with the densest 1-subdivision in G with nail set X. The same proof works for ½-shallow minors if we subdivide all edges existing inside X and then construct $\hat{G}$. ◀

Consequently, Dense 1-SD and Dense ½-STM both admit a simple $2^n n^{O(1)}$-algorithm: we guess the nail set X and apply the matching construction from Lemma 1. For the same reasons, both problems are in XP when parameterized by the number of nails. We cannot hope for much better since for $r = 0$ and $d \sim k^2$ we simply recover the problem of finding a k-clique. Besides being W[1]-hard and thus probably not in FPT, k-Clique further does not admit algorithms with running time $f(k)n^{o(k)}$ unless the ETH fails [4].

The approach of guessing the nail sets also fails for larger depths: knowing the nails of a, say, 1-shallow minors leaves us with the problem of contracting paths of length two into X, which cannot be represented as a simple matching problem. The reduction presented in Section 4.2 proves as a corollary that Dense 1-STM remains NP-hard when the nail set of the densest minor is known.

Finally, as we will see in Section 4.1, both problems are already NP-complete for very small densities d, making them paraNP-complete under this parameterization. Therefore none of the input variables will work well as a parameterization, and it is sensible to consider *structural* parameters, meaning parameters derived from the input graph. A good contender for such parameters are *width measures* like tree-, path-, or cliquewidth. Indeed, we can express the problem of finding a dense shallow minor or a dense subdivision in MSO_2 and apply variants of Courcelle's theorem to obtain the following:

▶ Proposition 2. Dense $r/2$-STM and Dense r-SD are in FPT when parameterized by the treewidth of the input graph.

Proof. We can express a model for an r-shallow minor in MSO_2 as follows: it consists of a vertex-set W and an edge set F, where F induces a set of paths. We can further easily express that the paths formed by F are a) of length at most r, b) disjoint, and c) have endpoints in W. Lastly, we demand that for every pair $x, y \in W$ there exists at most one path in F that has x and y as endpoints.

From an optimization perspective, we can therefore express the *feasible* solutions to Dense $r/2$-STM (and Dense r-SD with small modifications). In order to express our optimization goal, let us introduce one more set of vertices C with the property that every path induced by F contains at most one vertex from C—for example, we can express in MSO_2 that vertices of C are not pairwise reachable via the graph induced by F. With this auxilliary set, the density of the resulting minor is at least $|C|/|W|$ and exactly the density if C is maximial with respect to our choice of F. Accordingly, we find that there exists an r-shallow topological minor of density at least d if $|C| - d|W| \geqslant 0$. This constraint and the aforementioned MSO_2-description of a minor fall within the expressive power of the EMSO-framework introduced by Arnborg, Lagergren, and Seese [1] and we conclude that both Dense $r/2$-STM and Dense r-SD are fpt when parameterized by treewidth. ◀

Furthermore, it is not difficult (albeit tedious) to design a dynamic programming algorithm that solves Dense $r/2$-STM and Dense r-SD in time $2^{O(\mathbf{tw}^2)}n$. The quadratic dependence on the treewidth stems from the fact that we have to keep track of which edges we have contracted so far and there is no obvious way to circumvent this. The important question was whether any of the known techniques to reduce the complexity of connectivity-problems [6, 2, 19] could be applied here. The answer is, to our surprise, negative as we will discuss in Section 4.2.

4 Hardness results

4.1 NP-hardness and ETH lower bounds

This section will be dedicated to the proof the following theorems which both follow directly via a linear reduction from Positive 1-in-3SAT.

▶ **Theorem 2.** Dense $r/2$-STM *and* Dense r-SD *are* NP*-hard for* $r \geqslant 1$*, even when restricted to graphs that can be turned into subcubic planar graphs by deleting a single vertex.*

▶ **Theorem 3.** Dense $r/2$-STM *and* Dense r-SD *cannot be solved in time* $2^{o(n)}n^{O(1)}$ *on bipartite graphs unless the ETH fails.*

A special case of our result might be of independent interest:

▶ **Theorem 4.** Dense Bipartite Subdivision *is* NP*-hard even on instances* $\big((X, Y, E), d\big)$ *where vertices in* X *have degree at most* 3 *and* $d \geqslant 3$*.*

In the following, we present two reduction from Positive 1-in-3SAT that depend on the parity of r. We describe the reduction for $r \in \{1, 2\}$ and then argue how to modify the construction for arbitrary values of r. Note that the resulting instances are such that the densest graph H that appears as a $r/2$-shallow topological minor appears, in fact, as an r-subdivision and thus the reductions work for both problems.

Reduction for r odd

Let ψ be a Positive 1-in-3SAT instance with clauses $C_1, \dots C_m$ and variables $x_1, \dots x_p$. We assume that every variable in ψ appears in at least 3 clauses; if not, we can duplicate clauses to achieve this without changing the satisfiability of ψ. We construct a graph G from ψ in the following manner (*cf.* Figure 2):

1. For each variable x_i, create a cycle D_i with as many vertices as the frequency of x_i in ψ.
2. Create an apex-vertex a that is connected to every vertex of the cycles $D_1, \dots, D_p$.
3. For each clause $\{x_i, x_j, x_k\}$, add a vertex u_{ijk} to the graph and connect it to one vertex in D_i, D_j, D_k each that has not yet been connected to any clause-vertex.
4. Subdivide every edge appearing in the cycles $D_1, \dots, D_p$ and all edges incident to the apex a.

For easier presentation, let us color the vertices of G as follows: the vertices introduced in the first step are *white*, the vertices introduced in the third step *gray*, and the subdivision vertices created in the last step *black* (the apex vertex a remains uncolored). Note that the graph G is bipartite, where one side of the partition contains exactly the white vertices and a.

Note that if the input formula ψ is planar, then the constructed graph is planar and subcubic after removing the apex vertex a.

▶ **Lemma 5.** *If* ψ *is satisfiable then* G *has a topological minor at depth* $1/2$ *of density* $\frac{5m}{2m+1}$*.*

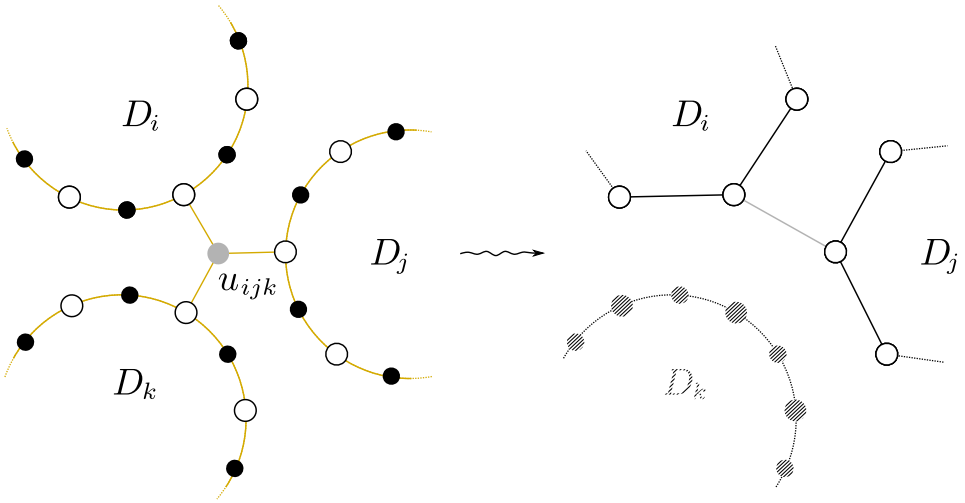


Figure 2 Three variable gadgets D_i, D_j, D_j connected by the gray clause vertex u_{ijk}. Setting variable x_k to true and x_i, x_j to false corresponds to the contractions on the right-hand side. The apex vertex a is not shown.

Proof. We construct the minor H by first smoothing each black vertex. Then, for each variable set to true, we delete the corresponding cycle D_i. Since ψ is satisfiable and each clause has exactly one variable set to true, this step deletes exactly one neighbor from each gray vertex. We complete the construction of H by smoothing out each gray vertex.

$V(H)$ consists of exactly two vertices corresponding to each clause plus the apex a, for a total of $2m+1$ vertices. Since all vertices of H were colored white in G, a has degree $2m$. Aside from the edges incident to a, there are m edges from smoothing gray vertices and $2m$ edges from smoothing black vertices, which yield a total of $5m$ edges. Thus, we have found a minor at depth $1/2$ of density $\frac{5m}{2m+1}$. ◀

▶ **Lemma 6.** *If G has a topological minor at depth $1/2$ of density at least $\frac{5m}{2m+1}$, then the formula ψ is satisfiable.*

Proof. Let H be the densest shallow topological minor at depth $1/2$ and fix some model of H in G. We first argue that the nails of H consist only of white vertex and potentially the apex vertex a.

▶ Claim. The nails of H consist of the apex vertex a and some subset of white vertices.

First, since the density of H is greater than two, its minimum degree is at least three (the removal of a degree-two vertex would increase the density). Since black vertices have degree two in G, the nails of the model forming H therefore cannot be black. Accordingly, every black vertex either does not participate in the formation of H or it is smoothed into an edge.

Let us define G_b to be the graph obtained from G by smoothing all black vertices. Since black vertices have degree exactly two, this operation is uniquely defined. By the previous observation, H can be obtained from G_b by only smoothing gray vertices and taking a subgraph. This of course implies that the nails of H are all either gray, white, or the apex vertex a. Let us now exclude the first of these three cases: assume y is a gray nail of H in G_b. Again, the degree of y in H must be at least three to ensure maximal density of H, and since y has degree three in G it must also have degree exactly three in H. Note that the three neighbors of y are necessarily white and independent in G_b, thus we can smooth y into an (arbitrary) edge between two of its neighbors. The newly obtained graph H' is again a

half-shallow topological minor of G and it contains one vertex and two edges less than H. Since the density of H is greater than two, this implies that the density of H' is greater than that of H, a contradiction. We conclude that the nails of H cannot be gray and therefore only consist of white vertices and, potentially, the apex vertex a. To see that a must be contained in H, simply note that otherwise the maximum degree of H would be three and as thus Hs density would lie strictly below the assumed $\frac{5m}{2m+1}$. In summary: H's nails consist of the apex vertex a and some subset of white vertices of G, proving the claim.

Since the white vertices in G are independent, the above claim further implies that the construction of H can be accomplished without smoothing white vertices. We can therefore divide said construction into two steps: first we smooth all gray and black vertices to construct a graph G_{gb} from G and then we take the subgraph $H \subseteq G_{gb}$. In the following, we will refer to edges in G_{gb} or H as *gray* if they originated from smoothing a gray vertex and *black* if they originated from smoothing a black vertex. Note that the set of black and gray edges partition $E(G_{gb})$ and hence also $E(H)$.

We now denote by v_4 the number of degree-four vertices in H and by v_3 the number of degree-three vertices (as observed above, no vertex with degree lower than three can exist in H and a is the only vertex of degree greater than four). Since the number of gray edges is at most m and a degree-four vertex must be incident to a gray edge, we have that $v_4 \leqslant 2m$. Let $w = v_3 + v_4$ be the number of white vertices in H and $\alpha = v_4/w$ the ratio of degree-four vertices among them. Using these quantities, we can express H's density as

$$\frac{2v_3 + {}^5\!/_2 v_4}{v_3 + v_4 + 1} = 2\frac{w}{w+1} + \frac{\alpha}{2}\frac{w}{w+1} = \Big(2 + \frac{\alpha}{2}\Big)\frac{w}{w+1}$$

which we combine with the density-requirement on H to obtain

$$\Big(2 + \frac{\alpha}{2}\Big)\frac{w}{w+1} \geqslant \frac{5m}{2m+1} \iff \alpha \geqslant 2\Big(\frac{5m}{2m+1}\frac{w+1}{w}\Big) - 4.$$

Note that the right-hand side is equal to one for $w = 2m$, smaller than one for $w > 2m$, and larger than one for $w < 2m$. This last regime would imply the impossible $\alpha > 1$ and we conclude $2m \leqslant w \leqslant 3m$, where the upper bound $3m$ is simply the total number of white vertices in G. Rewriting w as βm for $2 \leqslant \beta \leqslant 3$, we revisit the density-constraint on H:

$$\begin{aligned}\Big(2 + \frac{\alpha}{2}\Big)\frac{\beta m}{\beta m + 1} \geqslant \frac{5m}{2m+1} &\iff \Big(2\beta + \frac{\alpha\beta}{2}\Big)(2m+1) \geqslant 5(\beta m + 1) \\ &\implies (\alpha - 1)m + \frac{5}{2} \geqslant \frac{5}{\beta}. \qquad (\star)\end{aligned}$$

We will now show that α, the fraction of degree-four vertices among all w white vertices, needs to be one in order for $(\star)$ to hold. To that end, we distinguish the following two cases:

Case 1: $\beta = 2$. Assuming $\alpha \neq 1$, the largest possible value for α is achieved when $v_4 = 2m - 2$ (the case of exactly one gray edge missing from H), resulting in $\alpha = (2m-2)/2m = 1 - \frac{1}{m}$. Plugging this value of α and $\beta = 2$ into $(\star)$, we obtain that

$$(1 - 1/m - 1)m + \frac{5}{2} = -1 + \frac{5}{2} \geqslant \frac{5}{2},$$

a contradiction. Smaller values of α lead to the same contradiction, and we conclude that necessarily $\alpha = 1$.

Case 2: $2 < \beta \leqslant 3$. Assuming $\alpha \neq 1$, the largest possible value for α is achieved when $v_4 = 2m$, resulting in $\alpha = 2m/\beta m = \frac{2}{\beta}$. Now $(\star)$ becomes

$$\Big(\frac{2}{\beta} - 1\Big)m + \frac{5}{2} \geqslant \frac{5}{\beta} \iff m \leqslant \frac{10 - 5\beta}{2\beta} \cdot \frac{\beta}{2-\beta} = \frac{5}{2}.$$

Thus for formulas ψ with at least three clauses, we arrive at a contradiction and conclude that $\alpha = 1$.

We have now shown that a) H contains *only* vertices of degree four and b) that $|H| \geqslant 2m$. Since there cannot be more than $2m$ vertices of degree four, we conclude that H has exactly $2m$ vertices. Note that therefore H must consist of a collection of black-edge cycles with a total of $2m$ vertices, each of which is incident to exactly one of the m gray edges. Note that each black cycle B_i in H corresponds to a cycle D_i (associated with variable x_i) in G, where B_i was constructed from D_i by smoothing black vertices. Thus we can associate every black cycle B_i in H with a variable x_i in ψ. We claim that setting all such variables x_i that have a black cycle C_i in H to false and all other variables to true is a 1-in-3 satisfying assignment of ψ.

Consider any clause $\{x_i, x_j, x_k\}$ in ψ. The corresponding gray vertex u_{ijk} in G was smoothed into a gray edge e_{ijk} in H, since all m gray are present in H. Accordingly, exactly two of the three black cycles B_i, B_j, B_k are contained in H. Thus the assignment constructed above will set exactly two of the variables x_i, x_j, x_k to false and one variable to true. This argument holds for every clause in ψ and we conclude that the constructed assignment is 1-in-3 satisfying, proving the lemma. ◀

This concludes the reduction from Positive 1-in-3SAT. Note that an optimal solution in the reduction necessarily does not use any edges from the original graph, but only edges resulting from contractions. Therefore the reduction works for both Dense ½-STM and Dense ½-SD. As noted above, for $r = 1$ the constructed graph is bipartite. Since the latter set has degree at most three, Theorem 4 follows.

In order for the reduction to work for arbitrary odd r, we need to modify the construction in two places: first, we subdivide every edge in the clause gadget $(r-1)/2$ times. Second, instead of subdividing all edges appearing in the cycles $D_1, \ldots, D_p$ and edges incident to the apex a once, we subdivide them r times. The correctness of this reduction follows from easy modifications to Lemma 5 and 6, concluding our proof of Theorem 2 for odd values of r. Finally, to see that the above reduction also proofs Theorem 3 for odd r, simply note that the reduction results in a graph of size $\Theta(m)$ and the ETH lower bound follows from Proposition 1.

Reduction for r even

Let ψ be a Positive 1-in-3SAT instance as described above. Construct graph G in the following manner. We once again create a cycle D_i for each variable x_i, connect an apex vertex a to each vertex on the cycles, and color these vertices white. For this construction, however, we subdivide all edges between white vertices twice i.e. each white-white edge is replaced by a three-edge path. As with our previous construction, the subdivision vertices are all colored black. For each clause $C_i = \{x_j, x_k, x_\ell\}$, we add a triangle u_{ij}, u_{ij}, u_{ik} and connect it to the vertices from D_j, D_k, and D_ℓ corresponding to C_i such that u_{ij} is incident to the vertex from D_j etc. We color each of these vertices gray.

▶ **Lemma 7.** *If ψ is satisfiable then G has a topological minor at depth 1 of density $\frac{5m}{2m+1}$.*

Proof. We construct the minor H by first smoothing each black vertex. Then, for each variable set to true, we delete the corresponding cycle D_i. Since ψ is satisfiable and each clause has exactly one variable set to true, each gray triangle has two vertices of degree three and one of degree two. The degree two gray vertices are deleted, leaving the remaining gray

vertices to lie on three-edge paths between white vertices. These paths are subsequently smoothed to create white-white edges.

$V(H)$ consists of exactly two vertices corresponding to each clause plus a, for a total of $2m+1$ vertices. Since all vertices of H were colored white in G, a has degree $2m$. Aside from the edges incident to a, there are m edges from smoothing gray vertices and $2m$ edges from smoothing black vertices, which yield a total of $5m$ edges. Thus, we have found a minor at depth 1 of density $\frac{5m}{2m+1}$. ◀

▶ **Lemma 8.** *If G has a topological minor at depth 1 of density $\frac{5m}{2m+1}$ then ψ is satisfiable.*

Proof. Let H' be the densest topological minor at depth 1. For the same reasons presented in Lemma 6, H' has no black nails, and thus we can smooth all black vertices into white-white edges. This lack of black nails also implies that no white vertices can be smoothed to form a new edge incident to a gray vertex.

If H' contains all the gray and white vertices, it has $3m$ degree 4 white vertices, $3m$ degree 3 gray vertices, and a with degree $3m$ for a total of $12m$ edges and $6m+1$ vertices. This implies a density below $\frac{5m}{2m+1}$, and thus not all white and gray vertices are nails.

Since the gray vertices induce triangles, there is no way to smooth gray vertices to create a new gray-gray edge. Consider one such triangle T_i. If we smooth two vertices in T_i to create a single gray-white edge, the gray nail has degree 2 and should be deleted instead to increase the density. On the other hand, smoothing exactly one gray vertex to create a gray-white edge cause the remaining gray vertex to have degree two. Thus, any gray nail in H' is adjacent to three white vertices. Note that instead of having a gray nail, we could delete one gray vertex and smooth the other two into a white-white edge. The proof in Lemma 6 already demonstrated that forming the white-white edges is necessary to yield a density of $\frac{5m}{2m+1}$, and thus H' has no gray nails.

Since the gray vertices must be smoothed and deleted to create two degree 4 vertices and one degree 3 vertex per clause, the arguments in Lemma 6 imply that for H' to have density $\frac{5m}{2m+1}$, ψ must be satisfiable. ◀

In order for the reduction to work for arbitrary even r, we again modify the construction in two places: first, we subdivide every edge of the triangle making up the clause gadget $r/2-1$ times. Second, instead of subdividing all edges appearing in the cycles $D_1, \ldots, D_p$ and edges incident to the apex a twice, we subdivide them r times. With both cases of r even or odd covered, we conclude that Theorem 2 and Theorem 3 hold true.

4.2 Excluding a $2^{o(\mathrm{tw}^2)}n^{O(1)}$-algorithm

We show in this section that the ETH implies that we cannot get a single-exponential algorithm parameterized by treewidth for Dense $r/2$-STM for $r \geqslant 2$.

▶ **Theorem 9.** *Unless the ETH fails, there is no algorithm that decides* Dense 1-Shallow Topological Minor *on a graph with treewidth t in time $2^{o(t^2)}n^{O(1)}$.*

Our proof proceeds via a reduction from CNF-SAT. Assume that the CNF formula Φ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ is such that $\sqrt{n}$ is an even integer; if not, we pad Φ with dummy variables that appear in no clauses, which does not affect the answer to Φ. Figure 3 contains a sketch of the construction outlined in the following.

Decision gadget: The reduction will use sequences of vertices connected by decision gadgets. The decision gadget is a path of three vertices d_L, d_C, d_R which we will always connect to

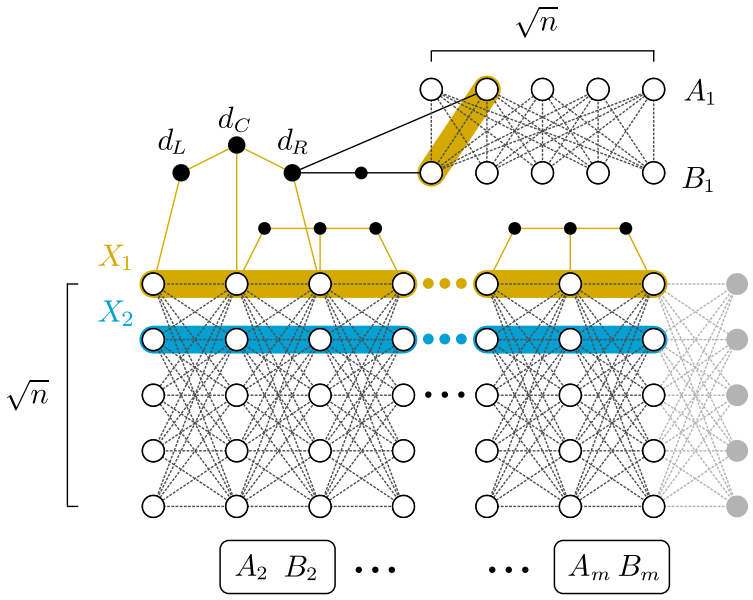


Figure 3 A sketch of the construction for Theorem 9, with an exemplary connection of the variable-path X_1 to the first clause gadget (here, x_1 appears negatively in C_1). Dashed edges denote parts that are actually connected via decision gadgets. 3-paths between the grid R and the clause gadgets (A_i, B_i) are not drawn.

a sequence of three vertices. For a sequence of vertices v_1, v_2, v_3, connecting the decision gadget to the sequence involves adding the edges $\{d_L, v_1\}$, $\{d_C, v_2\}$, and $\{d_R, v_3\}$.

Variable gadgets: We construct a grid of vertices R with $\sqrt{n}$ rows and m columns, denoting with $R[i, j]$ the vertex in the ith row and jth column. Each variable x_i will be represented by a sequence of m vertices X_i, one from each column. We will denote with $X_i[j]$ the jth vertex in the sequence X_i for any $1 \leqslant j \leqslant m$. In order to represent each of the n variables with m vertices using a $\sqrt{n} \times m$ grid, the sequences X_i must overlap and each vertex in R is part of the representation of $\sqrt{n}$ varibles. Specifically, let $X_i[j] = R[(i + j\lfloor i/\sqrt{n} \rfloor) \bmod \sqrt{n}, j]$. In other words, two vertices in successive columns will be in the same row in sequences $X_1, \ldots, X_{\sqrt{n}}$; they will be one row apart ("wrapping around" to the top from the bottom) in sequences $X_{1+\sqrt{n}}, \ldots, X_{2\sqrt{n}}$, two rows apart in $X_{1+2\sqrt{n}}, \ldots, X_{3\sqrt{n}}$, and so on.

For each sequence X_i, we connect $X_i[j-1]$, $X_i[j]$, and $X_i[j+1]$ to a decision gadget. Denote such a decision gadget as $D_{i,j}$. We also "wrap around" X_i by connecting $X_i[m-1], X_i[m], X_i[1]$ and $X_i[m], X_i[1], X_i[2]$ to their own decision gadgets.

Clause gadgets: Each clause C_i will be represented by a bipartition of vertices A_i, B_i where $|A_i| = |B_i| = \sqrt{n}$. Let $A_i[j]$ be the jth vertex in A_i and $B_i[j]$ likewise. Let σ be an ordering of the vertices in $A_i \cup B_i$ corresponding to an Eulerian tour of a biclique with bipartition A_i, B_i. Assume without loss of generality that $\sigma_i = A_i[\sqrt{n}], B_i[1], \ldots, B_i[\sqrt{n}], A_i[\sqrt{n}]$ and note that every pair of vertices $a \in A_i$ and $b \in B_i$ appears consecutively exactly once in σ. For each consecutive triple of vertices in σ_i, attach a decision gadget (but do not "wrap around").

Connecting variables and clauses: For each pair of vertices $(A_i[j], B_i[k])$ for $1 \leqslant j, k \leqslant \sqrt{n}$ assign the pair with a unique variable x_ℓ. Connect $X_\ell[i]$ to $A_i[j]$ and $B_i[k]$ via 3-edge paths. If x_ℓ appears in clause C_i positively, connect d_L of the decision gadget $D_{i,\ell}$ to $B_i[k]$ via an edge and to $A_i[j]$ via a 2-edge path. If it appears negatively, add the same connections to d_R of $D_{i,\ell}$ instead.

With the description of the reduction completed, let us now proof its correctness, *i.e.* We prove that G has a 1-STM of density $\rho = \frac{4\sqrt{n}}{3}$ if and only if Φ is satisfiable.

Forward direction: To prove the forward direction, we show how the satisfying assignment yields a topological minor of the desired density. We note that a cyclical sequence of vertices joined by decision gadgets can form a cycle in one of two ways: by smoothing each d_C and d_L or each d_C and d_R. Let the former be know as the *left configuration* of those sequenced gadgets and the latter the *right configuration*. Create a cycle on the vertices in X_i by choosing the right configuration if x_i is true and the left configuration if x_i is false.

For each clause, pick an arbitrary variable x_ℓ that satisfies it and let a and b be the pair of vertices from A_i and B_i assigned to that variable. If $a <_\sigma b$, set all decision gadgets preceding a in σ to the left configuration and all the decision gadgets succeeding b to the right configuration; do the reverse if $b <_\sigma a$. Thus A_i and B_i form a biclique missing the ab edge. Since there is a 3-edge path from a to b in G through a vertex in $D_{i,\ell}$ that has not been smoothed, we can use that path to form the ab edge. Smooth the remaining 3-edge induced paths in G and delete the vertices from the decision gadgets that were not contracted.

The nails of the resulting minor are exactly $\mathcal{A} \cup \mathcal{B} \cup R$. There are $m\sqrt{n}$ vertices in R and each one participates in $\sqrt{n}$ variable gadgets. Since each variable gadget becomes a cycle there are mn edges within R. The m clause gadgets become bicliques on $2\sqrt{n}$ vertices each and thus contain mn edges in total. Each variable gadget ends up with two edges into each clause gadget, for a total of $2mn$ edges connecting them. In total, this makes $4mn$ edges and $3m\sqrt{n}$ vertices, exactly ρ.

Reverse direction: We now prove the reverse direction by assuming Φ is unsatisfiable. Let H' be a 1-STM with density ρ. Since ρ is $\Theta(\sqrt{n})$ and the vertices in the induced paths and decision gadgets have degree at most 4, we can assume that none of those vertices appears as a nail in H'. Thus, the nail set of H' is a subset of $\mathcal{A} \cup \mathcal{B} \cup R$. The only paths between these nail candidates that use at most three edges do not contain nail candidates as interior vertices, meaning nail candidates are never smoothed. Let $H(\Phi)$ be the minor constructed by the process described in the forward direction proof for an arbitrary satisfying assignment of Φ. Observe that for fixed n and m, the minor $H(\psi)$ is identical for every satisfiable formula ψ; let $\mathcal{H}(n, m)$ be that minor. Moreover, every pair of nail candidates that has a 3-edge path between them in G is adjacent in $\mathcal{H}(n, m)$, meaning that H' is a either a subgraph of or identical to $\mathcal{H}(n, m)$.

We now show that no proper induced subgraph of $\mathcal{H}(n, m)$ has density ρ. If a graph is d-regular and connected, it has edge density $d/2$; deleting part of a degree regular graph leaves vertices with degree less than d, so no proper subgraph reaches that density. Therefore R and each $A_i \cup B_i$ achieve their maximum densities of $\sqrt{n}$ and $\sqrt{n}/2$ only when including the entire subgraph, implying a dense subgraph must contain portions of both vertex and clause gadgets. The only vehicle for increasing density is to use edges between vertex and clause gadgets, which means we should only include a vertex in R in a subgraph if it also contains its neighbors from $\mathcal{A}$ and $\mathcal{B}$. Let G' be the subgraph of G induced on an $i \times j$ subgrid of R and all of its neighbors in $\mathcal{A} \cup \mathcal{B}$. The density within the subset of $\mathcal{A} \cup \mathcal{B}$ is greatest when those vertices induce $j \times j$ bicliques, so we assume they do. Under this assumption, the density of G' is $\frac{4i}{3}$ if $j = m$ and $\frac{(4j-1)i}{3j}$ if $j < m$ since the edges that wrap around R cannot be realized. In either case, the density is strictly less than ρ unless $j = m$ and $i = \sqrt{n}$ i.e. exactly $\mathcal{H}(n, m)$.

A decision gadget connected to sequential vertices v_1, v_2, v_3 can only create the edge v_1, v_2 or the edge v_2, v_3, since d_C needs to be smoothed to construct either edge. Consequently, choosing to set some decision gadgets in the same variable gadget to opposite configurations

(or neither configuration) creates at least one fewer edge than if they were all set to the same decision. Thus, the configurations of the variable gadgets correspond to some truth assignment to Φ in the way intended in $H(\Phi)$. This indicates that there is a clause gadget that has no neighbors in a variable gadget that can be used to be smoothed into an edge in the clause gadget. However, because there is one fewer decision gadget than the number of biclique edges in the clause gadgets of $\mathcal{H}(n, m)$, H' cannot realize all possible edges in the biclique and thus there is no 1-STM of density ρ.

It stands to prove that the above reduction has the proper implications for a parameterization by treewidth. Using cops and robbers, we can show that G has treewidth $O(\sqrt{n})$ as follows: We permanently station $\sqrt{n}$ cops on the first column of R. We use $2\sqrt{n}$ cops to walk through the columns of R sequentially; when a column is completely covered with cops we can explore its corresponding clause gadget with a separate unit of $2\sqrt{n}$ cops. In total, this requires $O(\sqrt{n})$ cops. Although the formula Φ may have been padded with additional variables, it would only have been enough to increase $\sqrt{n}$ by 2. This means the number of variables in the unpadded instance is still $\Theta(n)$. Thus, if an algorithm parameterized by treewidth t could find a dense 1-STM in time $2^{o(t^2)}n^{O(1)}$, then we could use our reduction to solve CNF-SAT in time $2^{o(n)}n^{O(1)}$, violating ETH. This concludes the proof of Theorem 9.

An immediate consequence is that, unlike Dense ½-STM, Dense 1-STM is still NP-hard when the exact nail set is known.

5 Conclusion

We showed that finding dense substructures that are just slightly less local than subgraphs is computationally hard, and even a parameterization by treewidth cannot provide very efficient algorithms. While our first reduction excludes a subexponential exact algorithm assuming the ETH, we could not exclude an algorithm with a running time of $(2-\varepsilon)^n n^{O(1)}$. Is such an algorithm possible for $r = 1$, or can one find a tighter reduction that provides a corresponding SETH lower bound? Our second reduction rules out a $2^{o(\mathbf{tw}^2)}n^{O(1)}$-algorithm for $r = 2$. Is a faster algorithm for $r = 1$ possible?

Finally, we ask whether there is a sensible notion of substructures that fit in between ½-shallow topological minors and subgraphs for which we can find the densest occurrence in polynomial time.

References

1 S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

2 H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In *International Colloquium on Automata, Languages, and Programming*, pages 196–207. Springer, 2013.

3 H. L. Bodlaender, T. Wolle, and A. Koster. Contraction and treewidth lower bounds. *J. Graph Algorithms Appl.*, 10(1):5–49, 2006.

4 J. Chen, X. Huang, I. A. Kanj, and G. Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006.

5 M. Cygan, F.V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Lower bounds based on the exponential-time hypothesis. In *Parameterized Algorithms*, pages 467–521. Springer, 2015.

6 M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, van J.M.M. Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 150–159. IEEE Computer Society, 2011.

7 E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 51(3):292–302, 2008.

8 F. Dorn, F. V. Fomin, and D. M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008.

9 P. G. Drange, M. Dregi, F.V. Fomin, S. Kreutzer, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, F. Reidl, S. Saurabh, F. Sánchez Villaamil, S. Siebertz, and S. Sikdar. Kernelization and sparseness: the case of dominating set. In *33rd Symposium on Theoretical Aspects of Computer Science*, 2016.

10 Z. Dvořák. *Asymptotical Structure of Combinatorial Objects*. PhD thesis, Charles University, Faculty of Mathematics and Physics, 2007.

11 Z. Dvořák, D. Král, and R. Thomas. Deciding first-order properties for sparse graphs. In *Proceedings of the 51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 133–142. IEEE Computer Society, 2010.

12 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Linear kernels for (connected) dominating set on H-minor-free graphs. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 82–93. SIAM, 2012.

13 J. Gajarský, P. Hliněný, J. Obdržálek, S. Ordyniak, F. Reidl, P. Rossmanith, F. Sánchez Villaamil, and S. Sikdar. Kernelization using structural parameters on sparse graph classes. *To appear in Journal of Computer and System Sciences*, 2016.

14 G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, 1989.

15 A. V. Goldberg. *Finding a maximum density subgraph*. University of California Berkeley, CA, 1984.

16 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC)*, pages 89–98, 2014.

17 W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. *Journal of the ACM (JACM)*, 55(2):11, 2008.

18 J. Nešetřil and P. Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

19 M. Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: a logical approach. In *International Symposium on Mathematical Foundations of Computer Science*, pages 520–531. Springer, 2011.

20 F. Reidl. *Structural sparseness and complex networks*. Dr., Aachen, Techn. Hochsch., Aachen, 2016. Aachen, Techn. Hochsch., Diss., 2015. URL: `http://publications.rwth-aachen.de/record/565064`.

21 T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM, 1978.

Walrasian Pricing in Multi-Unit Auctions*

Simina Brânzei[1], Aris Filos-Ratsikas[2], Peter Bro Miltersen[3], and Yulong Zeng[4]

1 **Hebrew University of Jerusalem, Jerusalem, Israel**
simina.branzei@gmail.com
2 **Oxford University, Oxford, United Kingdom**
aris.filos-ratsikas@cs.ox.ac.uk
3 **Aarhus University, Aarhus, Denmark**
bromille@cs.au.dk
4 **Tsinghua Univesity, Beijing, China**
cengyl13@mails.tsinghua.edu.cn

Abstract

Multi-unit auctions are a paradigmatic model, where a seller brings multiple units of a good, while several buyers bring monetary endowments. It is well known that Walrasian equilibria do not always exist in this model, however compelling relaxations such as Walrasian envy-free pricing do. In this paper we design an optimal envy-free mechanism for multi-unit auctions with budgets. When the market is even mildly competitive, the approximation ratios of this mechanism are small constants for both the revenue and welfare objectives, and in fact for welfare the approximation converges to 1 as the market becomes fully competitive. We also give an impossibility theorem, showing that truthfulness requires discarding resources, and in particular, is incompatible with (Pareto) efficiency.

1998 ACM Subject Classification I.2.11 [Distributed Artificial Intelligence] Multiagent Systems, J.4 [Social and Behavioral Sciences] Economics, F.2 [Analysis of Algorithms and Problem Complexity]

Keywords and phrases mechanism design, multi-unit auctions, Walrasian pricing, market share

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.80

1 Introduction

Auctions are procedures for allocating goods that have been studied in economics in the 20th century, and which are even more relevant now due to the emergence of online platforms. Major companies such as Google and Facebook make most of their revenue through auctions, while an increasing number of governments around the world use spectrum auctions to allocate licenses for electromagnetic spectrum to companies. These transactions involve hundreds or thousands of participants with complex preferences, reason for which auctions require more careful design and their study has resurfaced in the computer science literature.

* Simina Brânzei was supported by the ISF grant 1435/14 administered by the Israeli Academy of Sciences and Israel-USA Bi-national Science Foundation (BSF) grant 2014389 and the I-CORE Program of the Planning and Budgeting Committee and The Israel Science Foundation. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 740282). Aris Filos-Ratsikas was supported by the ERC Advanced Grant 321171 (ALGAME). A part of this work was done when Simina Brânzei was visiting the Simons Institute for the Theory of Computing.

In this paper we study a paradigmatic model known as multi-unit auctions with budgets, in which a seller brings multiple units of a good (e.g. apples), while the buyers bring money and have interests in consuming the goods. Multi-unit auctions have been studied in a large body of literature due to the importance of the model, which already illustrates complex phenomena [16, 6, 18, 17, 19].

The main requirements from a good auction mechanism are usually computational efficiency, revenue maximization for the seller, and simplicity of use for the participants, the latter of which is captured through the notion of truthfulness. An important property that is often missing from auction design is fairness, and in fact for the purpose of maximizing revenue it is useful to impose higher payments to the buyers that are more interested in the goods. However, there are studies showing that customers are unhappy with such discriminatory prices (see, e.g., [1]), which has lead to a body of literature focused on achieving fair pricing [25, 22, 14, 23, 38].

A remarkable solution concept that has been used for achieving fairness in auctions comes from free markets, which are economic systems where the prices and allocations are not designed by a central authority. Instead, the prices emerge through a process of adjusting demand and supply such that everyone faces the same prices and the buyers freely purchase the bundles they are most interested in. When the goods are divisible, an outcome where supply and demand are perfectly balanced – known as competitive (or Walrasian) equilibrium [39] – always exists under mild assumptions on the utilities and has the property that the participants face the same prices and can freely acquire their favorite bundle at those prices. The competitive equilibrium models outcomes of large economies, where the goods are divisible and the participants so small (infinitesimal) that they have no influence on the market beyond purchasing their most preferred bundle at the current prices. Unfortunately, when the goods are indivisible, the competitive equilibrium does not necessarily exist (except for small classes of valuations see, e.g., [30, 24]) and the induced mechanism – the Walrasian mechanism [3, 13] – is generally manipulable.

A solution for recovering the attractive properties of the Walrasian equilibrium in the multi-unit model is to relax the clearing requirement of the market equilibrium, by allowing the seller to not sell all of the units. This solution is known as (Walrasian) envy-free pricing [25], and it ensures that all the participants of the market face the same prices[1], and each one purchases their favorite bundle of goods. An envy-free pricing trivially exists by pricing the goods infinitely high, so the challenge is finding one with good guarantees, such as high revenue for the seller or high welfare for the participants.

We would like to obtain envy-free pricing mechanisms that work well with strategic participants, who may alter their inputs to the mechanism to get better outcomes. To this end, we design an optimal truthful and envy-free mechanism for multi-unit auctions with budgets, with high revenue and welfare in competitive environments. Our work can be viewed as part of a general research agenda of *simplicity in mechanism design* [27], which recently proposed item pricing [4, 23] as a way of designing simpler auctions while at the same time avoiding the ill effects of discriminatory pricing [22, 1]. Item pricing is used in practice all over the world to sell goods in supermarkets or online platforms such as Amazon, which provides a strong motivation for understanding it theoretically. Other recent notions of simplicity in mechanism design include the menu-size complexity [26], the competition complexity [20], and verifiability of mechanisms (e.g. that the participants can easily convince themselves that the mechanism has a property, such as being truthful [9, 31]).

[1] The term envy-free pricing has also been used when the pricing is per-bundle, not per-item. We adopt the original definition of [25] which applies to unit-pricing, due to its attractive fairness properties [22].

1.1 Our Results

Our model is a multi-unit auction with budgets, in which a seller owns m identical units of an item. Each buyer i has a budget B_i and a value v_i per unit. The utilities of the buyers are quasi-linear up to the budget cap, while any allocation that exceeds that cap is unfeasible.

We deal with the problem of designing envy-free pricing schemes for the strongest concept of incentive compatibility, namely dominant strategy truthfulness. The truthful mechanisms are in the *prior-free* setting, i.e. they do not require any prior distribution assumptions. We evaluate the efficiency of mechanisms using the notion of *market share*, s^*, which captures the maximum buying power of any individual buyer in the market. A market share of at most 50% roughly means that no buyer can purchase more than half of the resources when competition is maximal, i.e. at the minimum envy-free price. Our main theorem can be summarized as follows.

▶ **Theorem 1** (Main Theorem (informal)). *For linear multi-unit auctions with known monetary endowments:*

- *There exists no (Walrasian) envy-free mechanism that is both truthful and non-wasteful.*
- *There exists a truthful (Walrasian) envy-free auction, which attains a fraction of at least* $\max\left\{2, \frac{1}{1-s^*}\right\}$ *of the optimal revenue and at least* $1 - s^*$ *of the optimal welfare on any market, where* $0 < s^* < 1$ *is the market share. This mechanism is optimal for both the revenue and welfare objectives when the market is even mildly competitive (i.e. with market share* $s^* \leq 50\%$*), and its approximation for welfare converges to* 1 *as the market becomes fully competitive.*

In the statement above, optimal means that there is no other truthful envy-free auction mechanism with a better approximation ratio. A mechanism is non-wasteful if it allocates as many units as possible at a given price. The impossibility theorem implies in particular that truthfulness is incompatible with Pareto efficiency. Our positive results are for *known* budgets, similarly to [16]. In the economics literature budgets are viewed as hard information (quantitative), as opposed to the valuations, which represent soft information and are more difficult to verify (see, e.g., [37]).

1.2 Related Work

The multi-unit setting has been studied in a large body of literature on auctions ([16, 6, 18, 17, 19]), where the focus has been on designing truthful auctions with good approximations to some desired objective, such as the social welfare or the revenue. Quite relevant to ours is the paper by [16], in which the authors study multi-unit auctions with budgets, however with no restriction to envy-free pricing or even item-pricing. They design a truthful auction (that uses discriminatory pricing) for *known budgets*, that achieves near-optimal revenue guarantees when the influence of each buyer in the auction is bounded, using a notion of buyer *dominance*, which is conceptually close to the market share notion that we employ. Their mechanism is based on the concept of clinching auctions [2].

Attempts at good prior-free truthful mechanisms for multi-unit auctions are seemingly impaired by their general impossibility result which states that truthfulness and efficiency are essentially incompatible when the budgets are *private*. Our general impossibility result is very similar in nature, but it is not implied by the results in [16] for the following two reasons: (a) our impossibility holds for *known* budgets and (b) our notion of efficiency is weaker, as it is naturally defined with respect to envy-free allocations only. This also means that our impossibility theorem is not implied by their uniqueness result, even for two buyers.

Multi-unit auctions with budgets have also been considered in [17] and [6], and without budgets ([19, 5, 18]); all of the aforementioned papers do not consider the envy-freeness constraint.

The effects of strategizing in markets have been studied extensively over the past few years ([7, 8, 12, 33, 34]). For more general envy-free auctions, besides the multi-unit case, there has been some work on truthful mechanisms in the literature of envy-free auctions ([25]) and ([28]) for *pair envy-freeness*, a different notion which dictates that no buyer would want to swap its allocation with that of any other buyer [32]. It is worth noticing that there is a body of literature that considers envy-free pricing as a purely optimization problem (with no regard to incentives) and provides approximation algorithms and hardness results for maximizing revenue and welfare in different auction settings [22, 15].

It is worth mentioning that the good approximations achieved by our truthful mechanism are a *prior-free setting* ([29]), i.e. we don't require any assumptions on prior distributions from which the input valuations are drawn. Good prior-free approximations are usually much harder to achieve and a large part of the literature is concerned with auctions under distributional assumptions, under the umbrella of *Bayesian mechanism design* ([10, 11, 29, 35]).

2 Preliminaries

In a linear multi-unit auction with budgets there is a set of buyers, denoted by $N = \{1, \dots, n\}$, and a single seller with m indivisible units of a good for sale. Each buyer i has a valuation $v_i > 0$ and a budget $B_i > 0$, both drawn from a discrete domain $\mathbb{V}$ of rational numbers: $v_i, B_i \in \mathbb{V}$. The valuation v_i indicates the value of the buyer for one unit of the good.

An *allocation* is an assignment of units to the buyers denoted by a vector $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{Z}_+^n$, where x_i is the number of units received by buyer i. We are interested in feasible allocations, for which: $\sum_{i=1}^{n} x_i \leq m$.

The seller will set a price p per unit, such that the price of purchasing ℓ units is $p \cdot \ell$ for any buyer. The interests of the buyers at a given price are captured by the demand function.

▶ **Definition 2** (Demand)**.** The *demand* of buyer i at a price p is a set consisting of all the possible bundle sizes (number of units) that the buyer would like to purchase at this price:

$$D_i(p) = \begin{cases} \min\{\lfloor \frac{B_i}{p} \rfloor, m\}, & \text{if } p < v_i \\ 0, \dots, \min\{\lfloor \frac{B_i}{p} \rfloor, m\}, & \text{if } p = v_i \\ 0, & \text{otherwise.} \end{cases}$$

If a buyer is indifferent between buying and not buying at a price, then its demand is a set of all the possible bundles that it can afford, based on its budget constraint.

▶ **Definition 3** (Utility)**.** The *utility* of buyer i given a price p and an allocation $\mathbf{x}$ is

$$u_i(p, x_i) = \begin{cases} v_i \cdot x_i - p \cdot x_i, & \text{if } p \cdot x_i \leq B_i \\ -\infty, & \text{otherwise} \end{cases}$$

(Walrasian) Envy-free Pricing. An allocation and price $(\mathbf{x}, p)$ represent a (Walrasian) *envy-free pricing* if each buyer is allocated a number of units in its demand set at price p, i.e. $x_i \in D_i(p)$ for all $i \in N$. A price p is an *envy-free price* if there exists an allocation $\mathbf{x}$ such that $(\mathbf{x}, p)$ is an envy-free pricing.

While an envy-free pricing always exists (just set $p = \infty$), it is not always possible to sell *all* the units in an envy-free way. We illustrate this through an example.

▶ **Example 4** (Non-existence of envy-free clearing prices)**.** Let $N = \{1, 2\}$, $m = 3$, valuations $v_1 = v_2 = 1.1$, and $B_1 = B_2 = 1$. At any price $p > 0.5$, no more than 2 units can be sold in total because of budget constraints. At $p \leq 0.5$, both buyers are interested and demand at least 2 units each, but there are only 3 units in total.

Objectives. We are interested in maximizing the *social welfare* and *revenue* objectives attained at envy-free pricing. The *social welfare* at an envy-free pricing $(\mathbf{x}, p)$ is the total value of the buyers for the goods allocated, while the *revenue* is the total payment received by the seller, i.e. $\mathcal{SW}(\mathbf{x}, p) = \sum_{i=1}^{n} v_i \cdot x_i$ and $\mathcal{REV}(\mathbf{x}, p) = \sum_{i=1}^{n} x_i \cdot p$.

Mechanisms. The goal of the seller will be to obtain money in exchange for the goods, however, it can only do that if the buyers are interested in purchasing them. The problem of the seller will be to obtain accurate information about the preferences of the buyers that would allow optimizing the pricing. Since the inputs (valuations) of the buyers are private, we will aim to design auction mechanisms that incentivize the buyers to reveal their true preferences [36].

An auction *mechanism* is a function $M : \mathbb{V}^n \to \mathbb{O} \times \mathbb{Z}_+^n$ that maps the valuations reported by the buyers to a price $p \in \mathbb{O}$, where $\mathbb{O}$ is the space from which the prices are drawn[2], and an allocation vector $\mathbf{x} \in \mathbb{Z}_+^n$.

▶ **Definition 5** (Truthful Mechanism)**.** A mechanism M is *truthful* if it incentivizes the buyers to reveal their true inputs, i.e. $u_i(M(\mathbf{v})) \geq u_i(M(v_i', v_{-i}))$, for all $i \in N$, any alternative report $v_i' \in \mathbb{V}$ of buyer i and any vector of reports v_{-i} of all the other buyers.

Requiring incentive compatibility from a mechanism can lead to worse revenue, so our goal will be to design mechanisms that achieve revenue close to that attained in the pure optimization problem (of finding a revenue optimal envy-free pricing without incentive constraints).

Types of Buyers. The next definitions will be used extensively in the paper. Buyer i is said to be *hungry* at price p if $v_i > p$ and *semi-hungry* if $v_i = p$. Given an allocation $\mathbf{x}$ and a price p buyer i is *essentially hungry* if it is either semi-hungry with $x_i = \min\{\lfloor B_i/p \rfloor, m\}$ or hungry. In other words, a buyer is essentially hungry if its value per unit is at least as high as the price per unit and, moreover, the buyer receives the largest non-zero element in its demand set.

3 An optimal envy-free and truthful mechanism

In this section, we present our main contribution, an envy-free and truthful mechanism, which is optimal among all truthful mechanisms and achieves small constant approximations to the optimal welfare and revenue. The approximation guarantees are with respect to the *market-share* s^*, which intuitively captures the maximum purchasing power of any individual buyer in the auction. The formal definition is postponed to the corresponding subsection.

[2] In principle the spaces $\mathbb{V}$ and $\mathbb{O}$ can be the same but for the purpose of getting good revenue and welfare, it is useful to have the price to be drawn from a slightly larger domain; see Section 3.

▶ **Theorem 6.** *There exists a truthful (Walrasian) envy-free auction, which attains a fraction of at least*

- $\max\left\{2, \frac{1}{1-s^*}\right\}$ *of the optimal revenue, and*
- $1 - s^*$ *of the optimal welfare*

on any market. This mechanism is optimal for both the revenue and welfare objectives when the market is even mildly competitive (i.e. with market share $s^* \leq 50\%$), and its approximation for welfare converges to 1 *as the auction becomes fully competitive.*

Consider the following mechanism.

All-or-Nothing:

Given as input the valuations of the buyers, let p be the minimum envy-free price and $\mathbf{x}$ the allocation obtained as follows:

- For every hungry buyer i, set x_i to its demand.
- For every buyer i with $v_i < p$, set $x_i = 0$.
- For every semi-hungry buyer i, set $x_i = \lfloor B_i/p \rfloor$ if possible, otherwise set $x_i = 0$ taking the semi-hungry buyers in lexicographic order.

In other words, the mechanism always outputs the minimum envy-free price but if there are semi-hungry buyers at that price, they get either all the units they can afford at this price or 0, even if there are still available units, after satisfying the demands of the hungry buyers.

▶ **Lemma 7.** *The minimum envy-free price does not exist when the price domain is* $\mathbb{R}$.

Proof. If the price can be any real number, consider an auction with $n = 2$ buyers, $m = 2$ units, valuations $v_1 = v_2 = 3$ and budgets $B_1 = B_2 = 2$. At any price $p \leq 1$, there is overdemand since each buyer is hungry and demands at least 2 units, while there are only 2 units in total. At any price $p \in (1, 2]$, each buyer demands at most one unit due to budget constraints, and so all the prices in the range $(1, 2]$ are envy-free. This is an open set, and so there is no minimum envy-free price. Note however, that by making the output domain discrete, e.g. with 0.1 increments starting from zero, then the minimum envy-free price output is 1.01. At this price each buyer purchases 1 unit. ◀

Given the example above, we will consider the discrete domain $\mathbb{V}$ as an infinite grid with entries of the form $k \cdot \epsilon$, for $k \in \mathbb{N}$ and some sufficiently small[3] ϵ. For the output of the mechanism, we will assume a slightly finer grid, e.g. with entries $k \cdot \delta = k(\epsilon/2)$, for $k \in \mathbb{N}$. The minimum envy-free price can be found in time which is polynomial in the input and $\log(1/\epsilon)$, using binary search[4] and the mechanism is optimal with respect to discrete domain that we operate on. Operating on a grid is actually without loss of generality in terms of the objectives; even if we compare to the optimal on the continuous domain, if our discretization is fine enough, we don't lose any revenue or welfare. This is established by the following theorem; the proof is omitted due to lack of space (see full version).

[3] For most of our results, any discrete domain is sufficient for the results to hold; for some results we will need to a number of grid points that polynomial in the size of the input grid.

[4] In the full version, we describe a faster procedure that finds the minimum envy-free without requiring to do binary search over the grid.

▶ **Theorem 8.** *When the valuation and budget of each buyer are drawn from a discrete grid with entries $k \cdot \epsilon$, and the price is is drawn from a finer grid with entries $k \cdot \epsilon/2$, for $k \in \mathbb{N}$, then the welfare and revenue loss of the* All-or-Nothing *mechanism due to the discretization of the output domain is zero. The mechanism always runs in time polynomial in the input and* $\log(1/\epsilon)$.

Truthfulness of the All-or-Nothing Mechanism

The following theorem establishes the truthfulness of All-or-Nothing.

▶ **Theorem 9.** *The* All-or-Nothing *mechanism is truthful.*

Proof. First, we will prove the following statement. If p is any envy-free price and p' is an envy-free price such that $p \leq p'$ then the utility of any essentially hungry buyer i at price p is at least as large as its utility at price p'. The case when $p' = p$ is trivial, since the price (and the allocation) do not change. Consider the case when $p < p'$. Since p is an envy-free price, buyer i receives the maximum number of items in its demand. For a higher price p', its demand will be at most as large as its demand at price p and hence its utility at p' will be at most as large as its utility at p.

Assume now for contradiction that Mechanism All-or-Nothing is not truthful and let i be a deviating buyer who benefits by misreporting its valuation v_i as v_i' at some valuation profile $\mathbf{v} = (v_1, \ldots, v_n)$, for which the minimum envy-free price is p. Let p' be the new minimum envy free price and let $\mathbf{x}$ and $\mathbf{x}'$ be the corresponding allocations at p and p' respectively, according to All-or-Nothing. Let $\mathbf{v}' = (v_i', v_{-i})$ be the valuation profile after the deviation.

We start by arguing that the deviating buyer i is essentially hungry. First, assume for contradiction that i is neither hungry nor semi-hungry, which means that $v_i < p$. Clearly, if $p' \geq p$, then buyer i does not receive any units at p' and there is no incentive for manipulation; thus we must have that $p' < p$. This implies that every buyer j such that $x_j > 0$ at price p is hungry at price p' and hence $x_j' \geq x_j$. Since the demand of all players does not decrease at p', this implies that p' is also an envy-free price on instance $\mathbf{v}$, contradicting minimality of p.

Next, assume that buyer i is semi-hungry but not essentially hungry, which means that $v_i = p$ and $x_i = 0$, by the allocation of the mechanism. Again, in order for the buyer to benefit, it has to hold that $p' < p$ and $x_i' > 0$ which implies that $x_i' = \lfloor B_i/p' \rfloor$, i.e. buyer i receives the largest element in its demand set at price p'. But then, since $p' < p$ and p' is an envy-free price, buyer i could receive $\lfloor B_i/p \rfloor$ units at price p without violating the envy-freeness of p, in contradiction with each buyer i being essentially hungry at p.

From the previous two paragraphs, the deviating buyer must be essentially hungry. This means that $x_i > 0$ and $v_i \geq p$. By the discussion in the first paragraph of the proof, we have $p' < p$. Since $x_i > 0$, the buyer does not benefit from reporting v_i' such that $v_i' < p'$. Thus it suffices to consider the case when $v_i' \geq p'$. We have two subcases:

- $v_i' > p$: Buyer i is essentially hungry at price p according to v_i and hungry at price p' according to v_i'. The reports of the other buyers are fixed and B_i is known; similarly to above, price p' is an envy-free price on instance $\mathbf{v}$, contradicting the minimality of p.

- $v_i' = p'$: Intuitively, an essentially hungry buyer at price p is misreporting its valuation as being lower trying to achieve an envy-free price p' equal to the reported valuation. Since $v_i' = p'$, Mechanism All-or-Nothing gives the buyer either as many units as it can afford at this price or zero units. In the first case, since p' is envy-free and B_i is known,

buyer i at price p' receives the largest element in its demand set and since the valuations of all other buyers are fixed, p' is also an envy-free price on input $\mathbf{v}$, contradicting the minimality of p. In the second case, the buyer does not receive any units and hence it does not benefit from misreporting.

Thus there are no improving deviations, which concludes the proof of the theorem. ◀

Performance of the All-or-Nothing Mechanism

Next, we show that the mechanism has a good performance for both objectives. We measure the performance of a truthful mechanism by the standard notion of approximation ratio, i.e.

$$\text{ratio(M)} = \sup_{\mathbf{v}\in\mathbb{R}^{\mathbf{n}}} \frac{\max_{\mathbf{x},p} \mathcal{OBJ}(\mathbf{v})}{\mathcal{OBJ}(M(\mathbf{v}))},$$

where $\mathcal{OBJ} \in \{\mathcal{SW}, \mathcal{REV}\}$ is either the social welfare or the revenue objective. Obviously, a mechanism that outputs a pair that maximizes the objectives has approximation ratio 1. The goal is to construct truthful mechanisms with approximation ratio as close to 1 as possible.

We remark here that for the approximation ratios, we only need to consider valuation profiles that are not "trivial", i.e. input profiles for which at any envy-free price, no hungry or semi-hungry buyers can afford a single unit and hence the envy-free price can be anything; on trivial profiles, both the optimal price and allocation and the price and allocation output by Mechanism All-or-Nothing obtain zero social welfare or zero revenue.

Market Share. A well-known notion for measuring the competitiveness of a market is the *market share*, understood as the percentage of the market accounted for by a specific entity (see, e.g., [21], Chapter 2).

In our model, the maximum purchasing power (i.e. number of units) of any buyer in the auction occurs at the minimum envy-free price, p_{min}. By the definition of the demand, there are many ways of allocating the semi-hungry buyers, so when measuring the purchasing power of an individual buyer we consider the maximum number of units that buyer can receive, taken over the set of all feasible maximal allocations at p_{min}. Let this set be $\mathcal{X}$. Then the market share of buyer i can be defined as:

$$s_i = \max_{\mathbf{x}\in\mathcal{X}} \left(\frac{x_i}{\sum_{k=1}^{n} x_k} \right).$$

Then, the *market share* is defined as $s^* = \max_{i=1}^{n} s_i$. Roughly speaking, a market share $s^* \leq 1/2$ means that a buyer can never purchase more than half of the resources.

▶ **Theorem 10.** *The* All-or-Nothing *mechanism approximates the optimal revenue within a factor of* 2 *whenever the market share,* s^**, is at most* 50%*.*

Proof. Let OPT be the optimal revenue, attained at some price p^* and allocation $\mathbf{x}$, and $\mathcal{REV}(AON)$ the revenue attained by the All-or-Nothing mechanism. By definition, mechanism All-or-Nothing outputs the minimum envy-free price p_{min}, together with an allocation $\mathbf{z}$. For ease of exposition, let $\alpha_i = B_i/p_{min}$ and $\alpha_i^* = B_i/p^*$, $\forall i \in N$. There are two cases, depending on whether the optimal envy-free price, p^*, is equal to the minimum envy-free price, p_{min}:

Case 1: $p^* > p_{min}$. Denote by L the set of buyers with valuations at least p^* that can afford at least one unit at the optimal price. Note that the set of buyers that get allocated

at p_{min} is a superset of L. Moreover, the optimal revenue is bounded by the revenue attained at the (possibly infeasible) allocation where all the buyers in L get the maximum number of units in their demand. These observations give the next inequalities:

$$\mathcal{REV}(AON) \geq \sum_{i \in L} \lfloor \alpha_i \rfloor \cdot p_{min} \quad \text{and} \quad OPT \leq \sum_{i \in L} \lfloor \alpha_i^* \rfloor \cdot p^*.$$

Then the revenue is bounded by:

$$\begin{aligned} \frac{\mathcal{REV}(AON)}{OPT} &\geq \frac{\sum_{i \in L} \lfloor \alpha_i \rfloor \cdot p_{min}}{\sum_{i \in L} \lfloor \alpha_i^* \rfloor \cdot p^*} \geq \frac{\sum_{i \in L} \lfloor \alpha_i \rfloor \cdot p_{min}}{\sum_{i \in L} \alpha_i^* \cdot p^*} = \frac{\sum_{i \in L} \lfloor \alpha_i \rfloor \cdot p_{min}}{\sum_{i \in L} B_i} \\ &= \frac{\sum_{i \in L} \lfloor \alpha_i \rfloor}{\sum_{i \in L} \alpha_i} \geq \frac{\sum_{i \in L} \lfloor \alpha_i \rfloor}{\sum_{i \in L} 2 \lfloor \alpha_i \rfloor} = \frac{1}{2}, \end{aligned}$$

where we used that the auction is non-trivial, i.e. for any buyer $i \in L$, $\lfloor \alpha_i \rfloor \geq 1$, and so $\alpha_i \leq \lfloor \alpha_i \rfloor + 1 \leq 2 \lfloor \alpha_i \rfloor$.

Case 2: $p^* = p_{min}$. The hungry buyers at p_{min}, as well as the buyers with valuations below p_{min}, receive identical allocations under All-or-Nothing and the optimal allocation, $\mathbf{x}$. However there are multiple ways of assigning the semi-hungry buyers to achieve an optimal allocation. Recall that $\mathbf{z}$ is the allocation made by All-or-Nothing. Without loss of generality, we can assume that $\mathbf{x}$ is an optimal allocation with the property that $\mathbf{x}$ is a superset of $\mathbf{z}$ and the following condition holds:

- *the number of buyers not allocated under* $\mathbf{z}$*, but that are allocated under* $\mathbf{x}$*, is minimized.*

We argue that $\mathbf{x}$ allocates at most one buyer more compared to $\mathbf{z}$. Assume by contradiction that there are at least two semi-hungry buyers i and j, such that $0 < x_i < \lfloor \alpha_i \rfloor$ and $0 < x_j < \lfloor \alpha_j \rfloor$. Then we can progressively take units from buyer j and transfer them to buyer i, until either buyer i receives $x_i' = \lfloor \alpha_i \rfloor$, or buyer j receives $x_j' = 0$. Hence we can assume that the set of semi-hungry buyers that receive non-zero, non-maximal allocations in the optimal solution $\mathbf{x}$ is either empty or a singleton. If the set is empty, then All-or-Nothing is optimal. Otherwise, let the singleton be ℓ; denote by $\tilde{x}_\ell$ the maximum number of units that ℓ can receive in any envy-free allocation at p_{min}. Since the number of units allocated by any maximal envy-free allocation at p_{min} is equal to $\sum_{i=1}^n x_i$, but $x_\ell \leq \tilde{x}_\ell$, we get:

$$\frac{x_\ell}{\sum_{i=1}^n x_i} \leq \frac{\tilde{x}_\ell}{\sum_{i=1}^n x_i} = s_i^*.$$

Thus

$$\begin{aligned} \frac{\mathcal{REV}(AON)}{OPT} &= \frac{OPT - x_\ell \cdot p_{min}}{OPT} \geq \frac{OPT - \tilde{x}_\ell \cdot p_{min}}{OPT} = 1 - \frac{\tilde{x}_\ell \cdot p_{min}}{\sum_{i=1}^n x_i \cdot p_{min}} \\ &= 1 - \frac{\tilde{x}_\ell}{\sum_{i=1}^n x_i} = 1 - s_i^* \geq 1 - s^* \end{aligned}$$

Combining the two cases, the bound follows. This completes the proof. ◀

▶ **Corollary 11.** *The performance of the* All-or-Nothing *mechanism is* $\max\{2, 1/(1-s^*)\}$ *on any market (i.e. with market share* $0 < s^* < 1$*).*

Proof. From the proof of Theorem 10, since the arguments of Case 1 do not use the market share s^*, it follows that the ratio of All-Or-Nothing for the revenue objective can alternatively be stated as $\max\{2, 1/(1-s^*)\}$ and therefore it degrades gracefully with the increase in the market share. ◀

The next theorem establishes that the approximation ratio for welfare is also constant.

▶ **Theorem 12.** *The approximation ratio of Mechanism* All-or-Nothing *with respect to the social welfare is at most* $1/(1-s^*)$*, where the market share* $s^* \in (0,1)$*. The approximation ratio goes to* 1 *as the market becomes fully competitive.*

Proof. For social welfare we have, similarly to Theorem 10, that

$$\frac{\mathcal{SW}(AON)}{OPT} = \frac{OPT - x_\ell \cdot v_\ell}{OPT} \geq \frac{OPT - \tilde{x}_\ell \cdot v_\ell}{OPT} = 1 - \frac{\tilde{x}_\ell \cdot v_\ell}{\sum_{i=1}^n x_i \cdot v_i} \geq 1 - \frac{\tilde{x}_\ell \cdot v_\ell}{\sum_{i=1}^n x_i \cdot v_\ell}$$
$$= 1 - \frac{\tilde{x}_\ell}{\sum_{i=1}^n x_i} = 1 - s_i^* \geq 1 - s^*,$$

where OPT is now the optimal welfare, $\mathbf{x}$ the corresponding allocation at OPT, and we used the fact that $v_\ell \leq v_i$ for all $i \in L$. ◀

Finally, All-or-Nothing is optimal among all truthful mechanisms for both objectives whenever the market share s^* is at most $1/2$.

▶ **Theorem 13.** *Let* M *be any truthful mechanism that always outputs an envy-free pricing scheme. Then the approximation ratio of* M *for the revenue and the welfare objective is at least* $2 - \frac{4}{m+2}$.

Proof. Consider an auction with equal budgets, B, and valuation profile $\mathbf{v}$. Assume that buyer 1 has the highest valuation, v_1, buyer 2 the second highest valuation v_2, with the property that $v_1 > v_2 + \epsilon$, where ϵ is set later. Let $v_i < v_2$ for all buyers $i = 3, 4, \ldots, n$. Set B such that $\lfloor \frac{B}{v_2} \rfloor = \frac{m}{2} + 1$ and ϵ such that $\lfloor \frac{B}{v_2+\epsilon} \rfloor = \frac{m}{2}$. Informally, the buyers can afford $\frac{m}{2} + 1$ units at prices v_2 and $v_2 + \epsilon$. Note that on this profile, Mechanism All-or-Nothing outputs price v_2 and allocates $\frac{m}{2} + 1$ units to buyer 1. For a concrete example of such an auction, take $m = 12$, $v_1 = 1.12$, $v_2 = 1.11$ (i.e. $\epsilon = 0.01$) and $B = 8$ (the example can be extended to any number of units with appropriate scaling of the parameters).

Let M be any truthful mechanism, p_M its price on this instance, and p^* the optimal price (with respect to the objective in question). The high level idea of the proof, for both objectives, is the following. We start from the profile $\mathbf{v}$ above, where $p_{min} = v_2$ is the minimum envy-free price, and argue that if $p^* \neq v_2$, then the bound follows. Otherwise, $p^* = v_2$, case in which we construct a series of profiles $\mathbf{v}, \mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(k)}$ that only differ from the previous profile in the sequence by the reported valuation $v_2^{(j)}$ of buyer 2. We argue that in each such profile, either the mechanism allocates units to buyer 1 only, case in which the bound is immediate, or buyer 2 is semi-hungry. In the latter case, truthfulness and the constraints on the number of units will imply that any truthful mechanism must allocate to buyer 2 zero items, yielding again the required bound.

First, consider the social welfare objective. Observe that for the optimal price p^* on profile $\mathbf{v}$, it holds that $p^* = v_2$. We have a few subcases:

Case 1: $p_M < v_2$. Then M is not an envy-free mechanism, since in this case there would be over-demand for units.

Case 2: $p_M > v_2$: Then M allocates units only to buyer 1, achieving a social welfare of at most $(\frac{m}{2} + 1)v_2$. The maximum social welfare is $m \cdot v_2$, so the approximation ratio of M is at least $\frac{m}{(m/2)+1} = 2 - \frac{4}{m+2}$.

Case 3: $p_M = v_2$: Let x_2 be the number of units allocated to buyer 2 at price v_2; note that since buyer 2 is semi-hungry at v_2, any number of units up to $\frac{m}{2} - 1$ is a valid allocation. If $x_2 = 0$, then M allocates units only to buyer 1 at price v_2 and for the same reason as in Case 2, the ratio is greater than or equal to $2 - \frac{4}{m+2}$; so we can assume $x_2 \geq 1$.

Next, consider valuation profile $\mathbf{v}^{(1)}$ where for each buyer $i \neq 2$, we have $v_i^{(1)} = v_i$, while for buyer 2, $v_2 < v_2^{(1)} < v_2 + \epsilon$. By definition of B, the minimum envy-free price on $\mathbf{v}^{(1)}$ is $v_2^{(1)}$. Let $p_M^{(1)}$ be the price output by M on valuation profile $\mathbf{v}^{(1)}$ and take a few subcases:

(a) $p_M^{(1)} > v_2^{(1)}$: Then using the same argument as in Case 2, the approximation is at least $2 - \frac{4}{m+2}$.

(b) $p_M^{(1)} < v_2^{(1)}$: This cannot happen because by definition of the budgets, $v_2^{(1)}$ is the minimum envy-free price.

(c) $p_M^{(1)} = v_2^{(1)}$: Let $x_2^{(1)}$ be the number of units allocated to buyer 2 at profile $\mathbf{v^{(1)}}$; we claim that $x_2^{(1)} \geq 2$. Otherwise, if $x_2^{(1)} \leq 1$, then on profile $\mathbf{v}^{(1)}$ buyer 2 would have an incentive to report v_2, which would move the price to v_2, giving buyer 2 at least as many units (at a lower price), contradicting truthfulness.

Consider now a valuation profile $\mathbf{v}^{(2)}$, where for each buyer $i \neq 2$, it holds that $v_i^{(2)} = v_i^{(1)} = v_i$ and for buyer 2 it holds that $v_2^{(1)} < v_2^{(2)} < v_2 + \epsilon$. For the same reasons as in Cases a-c, the behavior of M must be such that:

- the price output on input $\mathbf{v}^{(2)}$ is $v_2^{(2)}$ (otherwise M only allocates to buyer 1, and the bound is immediate), and
- the number of units $x_2^{(2)}$ allocated to buyer 2 is at least 3 (otherwise truthfulness would be violated).

By iterating through all the profiles in the sequence constructed in this manner, we arrive at a valuation profile $\mathbf{v}^{(k)}$ (similarly constructed), where the price is $v_2^{(k)}$ and buyer 2 receives at least $m/2$ units. However, buyer 1 is still hungry at price $v_2^{(k)}$ and should receive at least $\frac{m}{2} + 1$ units, which violates the unit supply constraint. This implies that in the first profile, $\mathbf{v}$, M must allocate 0 units to buyer 2 (by setting the price to v_2 or to something higher where buyer 2 does not want any units). This implies that the approximation ratio is at least $2 - \frac{4}{m+2}$.

For the revenue objective, the argument is exactly the same, but we need to establish that at any profile $\mathbf{v}$ or $\mathbf{v^{(i)}}$, $i = 1, \ldots, k$ that we construct, the optimal envy-free price is equal to the second highest reported valuation, i.e. v_2 or $v_2^{(i)}$, $i = 1, \ldots, k$ respectively. To do that, choose v_1 such that $v_1 = v_2 + \delta$, where $\delta > \epsilon$, but small enough such that $\lfloor \frac{B}{v_2+\delta} \rfloor = \lfloor \frac{B}{v_2} \rfloor$, i.e. any hungry buyer at price $v_2 + \delta$ buys the same number of units as it would buy at price v_2. Furthermore, ϵ and δ can be chosen small enough such that $(\frac{m}{2} + 1)(v_2 + \delta) < m \cdot v_2$, i.e. the revenue obtained by selling $\frac{m}{2} + 1$ units to buyer 1 at price $v_2 + \delta$ is smaller than the revenue obtained by selling $\frac{m}{2} + 1$ units to buyer 1 and $\frac{m}{2} - \epsilon$ units to buyer 2 at price v_2. This establishes the optimal envy-free price is the same as before, for every profile in the sequence and all arguments go through.

Given that we are working over a discrete domain, for the proof to go through, it suffices to assume that there are m points of the domain between v_1 and v_2, which is easily the case if the domain is not too sparse. Specifically, for the concrete example presented at the first paragraph of the proof, assuming that the domain contains all the decimal floating point numbers with up to two decimal places suffices. ◀

4 Impossibility Results

In this section, we state our impossibility results, which imply that truthfulness can only be guaranteed when there is some kind of wastefulness; a similar observation was made in [6] for a different setting.

▶ **Theorem 14.** *There is no Pareto efficient, truthful mechanism that always outputs an envy-free pricing, even when the budgets are known.*

The proof of the theorem is left for the full version. The next theorem provides a stronger impossibility result. First, we provide the necessary definitions. A buyer i on profile input v is called *irrelevant* if at the minimum envy-free price p on v, the buyer can not buy even a single unit. A mechanism is called *in-range* if it always outputs an envy-free price in the interval $[0, v_j]$ where v_j is the highest valuation among all buyers that are not irrelevant. Finally, a mechanism is *non-wasteful* if at a given price p, the mechanism allocates as many items as possible to the buyers. Note that Pareto efficiency implies in-range and non-wastefulness, but not the other way around. In a sense, while Pareto efficiency also determines the price chosen by the mechanism, non-wastefulness only concerns the allocation given a price, whereas in-range only restricts prices to a "reasonable" interval.

▶ **Theorem 15.** *There is no in-range, non-wasteful and truthful mechanism that always outputs an envy-free pricing scheme, even when the budgets are known.*

We leave the proof for the full version. To prove the impossibility, we first obtain a necessary condition; any mechanism in this class must essentially output the minimum envy-free price (or the next highest price on the output grid). Then we can use this result to construct and example where the mechanism must leave some items unallocated in order to satisfy truthfulness.

5 Discussion

Our results show that it is possible to achieve good approximate truthful mechanisms, under reasonable assumptions on the competitiveness of the auctions which retain some of the attractive properties of the Walrasian equilibrium solutions. The same agenda could be applied to more general auctions, beyond the case of linear valuations or even beyond multi-unit auctions. It would be interesting to obtain a complete characterization of truthfulness in the case of private or known budgets; for the case of private budgets, we can show that a class of order statistic mechanisms are truthful, but the welfare or revenue guarantees for this case may be poor. Finally, in the full version, we present an interesting special case, that of *monotone auctions*, in which Mechanism All-Or-Nothing is optimal among all truthful mechanisms for both objectives, regardless of the market share.

Acknowledgements. We would like to thank the MFCS reviewers for useful feedback.

References

1 E. Anderson and D. Simester. Price stickiness and customer antagonism. *Available at SSRN 1273647*, 2008.
2 L. M. Ausubel. An efficient ascending-bid auction for multiple objects. *The American Economic Review*, 94(5):1452–1475, 2004.
3 M. Babaioff, B. Lucier, N. Nisan, and R. Paes Leme. On the efficiency of the walrasian mechanism. In *ACM EC*, pages 783–800. ACM, 2014.
4 M. F. Balcan, A. Blum, and Y. Mansour. Item pricing for revenue maximization. In *ACM EC*, pages 50–59. ACM, 2008.
5 Y. Bartal, R. Gonen, and N. Nisan. Incentive compatible multi unit combinatorial auctions. In *TARK*, pages 72–87, 2003.

6 C. Borgs, J. Chayes, N. Immorlica, M. Mahdian, and A. Saberi. Multi-unit auctions with budget-constrained bidders. In *ACM EC*, pages 44–51, 2005.

7 A. Borodin, O. Lev, and T. Strangway. Budgetary effects on pricing equilibrium in online markets. In *AAMAS*, 2016.

8 S. Brânzei, Y. Chen, X. Deng, A. Filos-Ratsikas, S. K. S. Frederiksen, and J. Zhang. The fisher market game: Equilibrium and welfare. In *AAAI*, pages 587–593, 2014.

9 Simina Brânzei and Ariel D. Procaccia. Verifiably truthful mechanisms. In *ITCS*, pages 297–306, 2015.

10 Y. Cai, C. Daskalakis, and M. Weinberg. Optimal multi-dimensional mechanism design: Reducing revenue to welfare maximization. In *FOCS*, pages 130–139, 2012.

11 Y. Cai, C. Daskalakis, and M. Weinberg. Reducing revenue to welfare maximization: Approximation algorithms and other generalizations. In *SODA*, pages 578–595, 2013.

12 N. Chen, X. Deng, and J. Zhang. How profitable are strategic behaviors in a market? In *ESA*, pages 106–118. Springer, 2011.

13 M. Cheung and C. Swamy. Approximation algorithms for single-minded envy-free profit-maximization problems with limited supply. In *FOCS*, pages 35–44, 2008.

14 V. Cohen-Addad, A. Eden, M. Feldman, and A. Fiat. The invisible hand of dynamic market pricing. In *ACM EC*, pages 383–400, 2016.

15 R. Colini-Baldeschi, S. Leonardi, P. Sankowski, and Q. Zhang. Revenue maximizing envy-free fixed-price auctions with budgets. In *WINE*, pages 233–246. Springer, 2014.

16 S. Dobzinski, R. Lavi, and N. Nisan. Multi-unit auctions with budget limits. *GEB*, 74(2):486–503, 2012.

17 S. Dobzinski and R. P. Leme. Efficiency guarantees in auctions with budgets. In *ICALP*, pages 392–404. Springer, 2014.

18 S. Dobzinski and N. Nisan. Mechanisms for multi-unit auctions. In *ACM EC*, pages 346–351, 2007.

19 S. Dobzinski and N. Nisan. Multi-unit auctions: beyond roberts. *JET*, 156:14–44, 2015.

20 A. Eden, M. Feldman, O. Friedler, I. Talgam-Cohen, and M. Weinberg. The competition complexity of auctions: A bulow-klemperer result for multi-dimensional bidders. In *ACM EC*, 2017.

21 P. Farris, N. Bendle, P. Pfeifer, and D. Reibstein. *Marketing metrics: The definitive guide to measuring marketing performance*. Pearson Education, 2010.

22 M. Feldman, A. Fiat, S. Leonardi, and P. Sankowski. Revenue maximizing envy-free multi-unit auctions with budgets. In *ACM EC*, pages 532–549, 2012.

23 M. Feldman, N. Gravin, and B. Lucier. Combinatorial auctions via posted prices. In *SODA*, pages 123–135, 2015.

24 F. Gul and E. Stacchetti. Walrasian equilibrium with gross substitutes. *Journal of Economic Theory*, 87(1):95–124, 1999.

25 V. Guruswami, J. Hartline, A. Karlin, D. Kempe, C. Kenyon, and F. McSherry. On profit-maximizing envy-free pricing. In *SODA*, pages 1164–1173, 2005.

26 Sergiu Hart and Noam Nisan. The menu-size complexity of auctions. In *ACM EC*, pages 565–566, 2013.

27 J. Hartline and T. Roughgarden. Simple versus optimal mechanisms. In *ACM EC*, pages 225–234. ACM, 2009.

28 J. Hartline and Q. Yan. Envy, truth, and profit. In *ACM EC*, pages 243–252, 2011.

29 J. D. Hartline. Mechanism design and approximation. *Book draft. October*, 122, 2013.

30 A. S. Kelso and V. P. Crawford. Job matching, coalition formation, and gross substitutes. *Econometrica*, 50:1483–1504, 1982.

31 Shengwu Li. Obviously strategy-proof mechanisms, 2015. Working paper.

32 E. Markakis and O. Telelis. Envy-free revenue approximation for asymmetric buyers with budgets. In *SAGT*, pages 247–259, 2016.

33 R. Mehta and M. Sohoni. Exchange markets: Strategy meets supply-awareness. In *WINE*, pages 361–362. Springer, 2013.

34 R. Mehta, N. Thain, L. Végh, and A. Vetta. To save or not to save: The fisher game. In *WINE*, pages 294–307. Springer, 2014.

35 R. B. Myerson. Optimal auction design. *MOR*, 6(1):58–73, 1981.

36 N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani. *Algorithmic Game Theory*. Cambridge Univ. Press, (editors) 2007.

37 M. Petersen. Information: Hard and soft, 7 2004.

38 Shreyas Sekar. Posted pricing sans discrimination. In *IJCAI, to appear*, 2017.

39 L. Walras. Elements d'economie politique pure, ou theorie de la richesse sociale (in french), 1874. English translation: Elements of pure economics; or, the theory of social wealth. American Economic Association and the Royal Economic Society, 1954.

Distributed Strategies Made Easy

Simon Castellan[1], Pierre Clairambault[2], and Glynn Winskel[3]

1 **Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France**
2 **Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France**
3 **Computer Laboratory, University of Cambridge, UK**

Abstract

Distributed/concurrent strategies have been introduced as special maps of event structures. As such they factor through their "rigid images", themselves strategies. By concentrating on such "rigid image" strategies we are able to give an elementary account of distributed strategies and their composition, resulting in a category of games and strategies. This is in contrast to the usual development where composition involves the pullback of event structures explicitly and results in a bicategory. It is shown how, in this simpler setting, to extend strategies to probabilistic strategies; and indicated how through probability we can track nondeterministic branching behaviour, that one might otherwise think lost irrevocably in restricting attention to "rigid image" strategies.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Modes of Computation

Keywords and phrases Games, Strategies, Event Structures, Probability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.81

1 Introduction

Traditionally in understanding and analysing a large system, whether it be in computer science, physics, biology or economics, the system's behaviour is thought of as going through a sequence of actions as time progresses. This is bound up with our experience of the world as individuals; in our conscious understanding of the world we experience and narrate our individual history as a sequence, or total order, of events, one after the other. However, a complex system is often much more than an individual agent. It is better thought of as several or many agents interacting together and distributed over various locations. In which case it can be fruitful to abandon the view of its behaviour as caught by a total order of events and instead think of the events of the systems system as comprising a partial order. The partial order expresses the causal dependency between events, how an event depends on possibly several previous events. The view that causal dependency should be paramount over an often incidental temporal order has been discovered and rediscovered in many disciplines: in physics in the understanding of the causal structure of space time; in biology and chemistry in the description of biochemical pathways; in computer science, originally in the work of Petri on Petri nets, and later in the often more mathematically amenable event structures.

Interacting systems are often represented mathematically via games. A system operates in an unknown environment, so often a prescription for its intended behaviour can be expressed as a strategy in which the system is Player against (an unpredictable) Opponent, standing for the environment. Games and their strategies are ubiquitous. They appear in logic (proof theory, set theory, ...), computer science (semantics, algorithmics, ...), biology, economics, *etc.*. They codify the mathematics of interacting systems. But they almost always follow the traditional line of representing the history of a play of the game as a sequence of moves, most often alternating between Player and Opponent. Until recently there was no mathematical

theory of games based on partial orders of causal dependency between move occurrences. This handicapped their use in modelling and analysing a system of distributed agents.

What was lacking was a mathematical theory of distributed games in which Player and Opponent are more accurately thought of as teams of players, distributed over different locations, able to move and communicate with each other. Although there are glimpses of such a mathematical theory of distributed games in earlier work of Abramsky, Mellies and Mimram [1, 13], Faggian and Piccolo [8], and others, a breakthrough occurred with the systematic use of event structures to formalise distributed games and strategies [14]. This meant that we could harness the mathematical techniques developed around event structures in an early mathematical foundation for work on synchronising processes [18]; the move from total to partial orders brings in its wake a lot of technical difficulty and potential for undue complexity unless it's done artfully.

But here we meet an obstacle for many people. Distributed/concurrent strategies have been based on maps of event structures and composition on pullback, which in the case of event structures has to be defined rather indirectly. Then, one obtains not a category but a bicategory of games and strategies. At what seems like an increasingly slight cost, a more elementary treatment can be given. Its presentation is the purpose of this article. The maps and pullbacks are still there of course, but pushed into the background.

The realisation that a more elementary presentation will often suffice has been a gradual one. It is based on the fact that a strategy, presented as a map of event structures, has a "rigid image" in the game and that in many cases this image can stand as a proxy for the original strategy [25]. True some branching behaviour is lost, just as it, and possible deadlock and divergence, can be lost in the composition of strategies. But extra structure on strategies generally remedies this. For example, the introduction of probability to strategies allows the detection of divergence in composition, or hidden branching, through leaks of probability. One can go far with rigid images of strategies. They permit the elementary development presented here.

In their CONCUR'16 paper [2] Castellan and Clairambault used the simple presentation of "rigid image" strategies here. Meanwhile rigid images of strategies had come to play an increasing role in Winskel's ECSYM notes [25]. Before this, Nathan Bowler recognised essentially the same subcategory of games and "rigid image" strategies, within the bicategory of concurrent games and strategies. (At the time, Winskel thought that too much of the nondeterministic branching behaviour would be lost irrecoverably to be very enthusiastic.)

Finally, an apology: we obtained the results here by specialising more general results on strategies to their rigid-images [25]; elementary proofs of the results would be desirable for a fully self-contained presentation, and should be written up shortly.

2 Event structures

An *event structure* comprises $(E, \leq, \mathrm{Con})$, consisting of a set E of *events* which are partially ordered by $\leq$, the *causal dependency relation*, and a nonempty *consistency* relation Con consisting of finite subsets of E. The relation $e' \leq e$ expresses that event e causally depends on the previous occurrence of event e'. That a finite subset of events is consistent conveys that its events can occur together by some stage in the evolution of the process. Together

the relations satisfy several axioms:

$[e] =_{\text{def}} \{e' \mid e' \leq e\}$ is finite for all $e \in E$,

$\{e\} \in \text{Con}$ for all $e \in E$,

$Y \subseteq X \in \text{Con}$ implies $Y \in \text{Con}$, and

$X \in \text{Con}$ & $e \leq e' \in X$ implies $X \cup \{e\} \in \text{Con}$.

There is an accompanying notion of state, or history, those events that may occur up to some stage in the behaviour of the process described. A *configuration* is a, possibly infinite, set of events $x \subseteq E$ which is: *consistent,* $X \subseteq x$ and X is finite implies $X \in \text{Con}$; and *down-closed,* $e' \leq e \in x$ implies $e' \in x$.

Two events e, e' are considered to be causally independent, and called *concurrent* if the set $\{e, e'\}$ is in Con and neither event is causally dependent on the other; then we write $e \mathrel{co} e'$. In games the relation of *immediate* dependency $e \rightarrowtriangle e'$, meaning e and e' are distinct with $e \leq e'$ and no event in between, plays a very important role. We write $[X]$ for the down-closure of a subset of events X. Write $\mathcal{C}^\infty(E)$ for the configurations of E and $\mathcal{C}(E)$ for its finite configurations. (Sometimes we shall need to distinguish the precise event structure to which a relation is associated and write, for instance, $\leq_E$, $\rightarrowtriangle_E$ or co_E.)

We can describe a computation path by an *elementary* event structure, which is a partial order $p = (|p|, \leq_p)$ for which the set $\{e' \in |p| \mid e' \leq_p e\}$ is finite for all $e \in |p|$. We can regard an elementary event structure as an event structure in which the consistency relation consists of all finite subsets of events. There is a useful subpath order of rigid inclusion of one elementary event structure in another. Let $p = (|p|, \leq_p)$ and $q = (|q|, \leq_q)$ be elementary event structures. Write

$$p \hookrightarrow q \text{ iff } |p| \subseteq |q| \;\&\; \forall e \in |p|, e' \in |q|.\ e' \leq_p e \iff e' \leq_q e\,.$$

We shall often view a configuration x of E as an elementary event structure, *viz.* a partial order with underlying set x and partial order the causal dependency of E restricted to x.

In an interactive context a configuration x may be subject to causal dependencies beyond those of E. It will become an elementary event structure $p = (|p|, \leq_p)$ comprising an underlying set $|p| = x$ with a partial order $\leq_p$ which augments that from E:

$$\forall e \in |p|, e' \in E.\ e' \leq_E e \implies e' \leq_p e\,.$$

Write $\text{Aug}(E)$ for the set of such *augmentations* associated with E. The order of rigid inclusion of one augmentation in another expresses when one augmentation is a sub-behaviour of another.

It will be useful to combine augmentations, in effect subjecting a configuration simultaneously to the causal dependencies of the two augmentations – provided this does not lead to causal loops. Define a key partial operation

$$\wedge : \text{Aug}(E) \times \text{Aug}(E) \rightharpoonup \text{Aug}(E)$$

by taking

$$p \wedge q = \begin{cases} (|p|, (\leq_p \cup \leq_q)^*) & \text{if } |p| = |q| \;\&\; (\leq_p \cup \leq_q)^* \text{ is antisymmetric,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

▶ **Lemma 1.** *Letting $p, q \in \text{Aug}(E)$ for which $p \wedge q$ is defined, $e' \rightarrowtriangle_{p\wedge q} e$ implies*

$$[e' \rightarrowtriangle_p e \;\&\; (e' \rightarrowtriangle_q e \text{ or } e' \mathrel{co}_q e)] \text{ or } [e' \rightarrowtriangle_q e \;\&\; (e' \rightarrowtriangle_p e \text{ or } e' \mathrel{co}_p e)]\,.$$

In fact we can see $\mathrm{Aug}(E)$ as an event structure in its own right. Its events are those augmentations with a top event, their causal dependency and consistency induced given by rigid inclusion [20]. The remark is an instance of a general fact:

▶ **Proposition 2.** *A* rigid family $\mathcal{R}$ *comprises a non-empty subset of finite elementary event structures which is down-closed w.r.t. rigid inclusion, i.e.* $p \hookrightarrow q \in \mathcal{R}$ *implies* $p \in \mathcal{R}$*. A rigid family determines an event structure* $\Pr(\mathcal{R})$ *whose order of finite configurations is isomorphic to* $(\mathcal{R}, \hookrightarrow)$*. The event structure* $\Pr(\mathcal{R})$ *has events those elements of* $\mathcal{R}$ *with a top event; its causal dependency is given by rigid inclusion; and its consistency by compatibilty w.r.t. rigid inclusion. The order isomorphism* $\theta_{\mathcal{R}} : \mathcal{C}(\Pr(\mathcal{R})) \cong \mathcal{R}$ *is given by* $\theta_{\mathcal{R}}(x) = \bigcup x$*, the union of (the consistent) augmentations in* $x \in \mathcal{C}(\Pr(\mathcal{R}))$*.*

3 Event structures with polarity

An event structure with polarity comprises (A, pol) where A is an event structure with a polarity function $pol_A : A \to \{+, -, 0\}$ ascribing a polarity + (Player), − (Opponent) or 0 (neutral) to its events. The events correspond to (occurrences of) moves. It will be technically useful to allow events of neutral polarity; they arise, for example, in a play between a strategy and a counterstrategy. A *game* shall be represented by an event structure with polarity in which no moves are neutral.

▶ **Notation 3.** *In an event structure with polarity* (A, pol)*, with configurations* x *and* y*, write* $x \subseteq^- y$ *to mean inclusion in which all the intervening events are moves of Opponent. Write* $x \subseteq^+ y$ *for inclusion in which the intervening events are neutral or moves of Player.*

3.1 Operations

We introduce two fundamental operations on event structures with polarity. We shall adopt the same operations for elementary event structures, and also for configurations, regarding a configuration as an elementary event structure with the order of the ambient event structure.

3.1.1 Dual

The *dual*, $A^\perp$, of A, an event structure with polarity, comprises the same underlying event structure A but with a reversal of polarities, events of neutral polarity remaining neutral.

We shall implicitly adopt the view of Player and understand a strategy in a game A as strategy for Player. A counterstrategy in a game A is a strategy for Opponent in the game A, *i.e.* a strategy (for Player) in the game $A^\perp$.

3.1.2 Simple parallel composition

This operation simply juxtaposes two event structures with polarity. Let $(A, \leq_A, \mathrm{Con}_A, pol_A)$ and $(B, \leq_B, \mathrm{Con}_B, pol_B)$ be event structures with polarity. The events of $A \| B$ are $(\{1\} \times A) \cup (\{2\} \times B)$, their polarities unchanged, with the only relations of causal dependency given by $(1, a) \leq (1, a')$ iff $a \leq_A a'$ and $(2, b) \leq (2, b')$ iff $b \leq_B b'$; a subset of events C is consistent in $A \| B$ iff $\{a \mid (1, a) \in C\} \in \mathrm{Con}_A$ and $\{b \mid (2, b) \in C\} \in \mathrm{Con}_B$. The empty event structure with polarity, written $\varnothing$, is the unit w.r.t. $\|$.

4 Strategies

A strategy in a game will be a (special) subset of plays in the game.

▶ **Definition 4.** A *play* in A, an event structure with polarity, comprises an augmentation, a finite elementary event structure $p = (|p|, \leq_p)$ with underlying set $|p| \in \mathcal{C}(A)$, which may augment with extra causal dependencies provided it does so *courteously*:

$$\forall a, a' \in |p|.\ a' \rightarrowtriangle_p a \ \&\ pol_A(a') = + \text{ or } pol_A(a) = - \implies a' \rightarrowtriangle_A a\,.$$

Note A, and so p, may involve neutral moves.

If A is a game, so with no neutral moves, the only augmentations allowed of a play p to the immediate causal dependency of A are those of the form $\ominus \rightarrowtriangle \oplus$.

The order of rigid inclusion between plays, $p \hookrightarrow q$, expresses that p is a subplay of q. We shall write

$$p \hookrightarrow^+ q \text{ iff } p \hookrightarrow q \ \&\ |p| \subseteq^+ |q|\,,$$

so when the extension only involves neutral or Player moves, and similarly $p \hookrightarrow^- q$ when only Opponent moves are involved.

▶ **Definition 5.** A *bare strategy* in A, an event structure with polarity, is a rigid family of plays, so a nonempty subset $\sigma \subseteq \mathrm{Plays}(A)$ satisfying $p \hookrightarrow q \in \sigma \implies p \in \sigma$, which is also

- receptive, $p \in \sigma \ \&\ |p| \subseteq^- x \in \mathcal{C}(A) \implies \exists q \in \sigma.\ p \hookrightarrow q \in \sigma \ \&\ |q| = x\,.$
  (Note that q is unique by courtesy.)

Write $\sigma : A$ when σ is a bare strategy of A. When A is a game, so an event structure with polarity without neutral moves, we say σ is a *strategy*.

One simple example of a strategy $\sigma : A$ in a game A is got by taking σ to consist of all the finite configurations of A regarded as elementary event structures in which their order of causal dependency is inherited from A. (Bare strategies, with neutral events, have been called "partial strategies" in [25] and an "uncovered strategies" in [16].)

We shall regard a strategy in the compound game $A^\perp \| B$, where A and B are games as a strategy *from* the game A *to* the game B [7, 12].

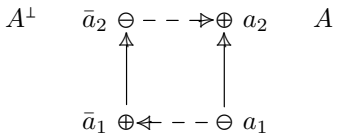
4.1 Copycat

We shall shortly define the composition of strategies. Identities w.r.t. composition are given by copycat strategies. Let A be a game. The copycat strategy $\mathfrak{cc}_A : A^\perp \| A$ is an instance of a strategy. We obtain copycat from the finite configurations of an event structure $\mathfrak{CC}_A$ based on the idea that Player moves, of +ve polarity, in one component of the game $A^\perp \| A$ always copy previous corresponding moves of Opponent, of −ve polarity in the other component.

For $c \in A^\perp \| A$ we use $\bar{c}$ to mean the corresponding copy of c, of opposite polarity, in the alternative component, *i.e.* $\overline{(1,a)} = (2,a)$ and $\overline{(2,a)} = (1,a)$. Define $\mathfrak{CC}_A$ to comprise the event structure with polarity $A^\perp \| A$ together with extra causal dependencies $\bar{c} \leq_{\mathfrak{CC}_A} c$ for all events c with $pol_{A^\perp \| A}(c) = +$. Take a finite subset to be consistent in $\mathfrak{CC}_A$ iff its down-closure w.r.t. the relation $\leq_{\mathfrak{CC}_A}$ is consistent in $A^\perp \| A$.

▶ **Example 6.** We illustrate the construction of $\mathfrak{CC}_A$ for the event structure A comprising the single immediate dependency $a_1 \rightarrowtriangle a_2$ from an Opponent move a_1 to a Player move

a_2. The event structure $!A$ is obtained from $A^\perp \| A$ by adjoining the additional immediate dependencies shown:



▶ **Lemma 7.** *Let A be an event structure with polarity. Then, $!A$ is an event structure with polarity. Moreover,*

$$x \in \mathcal{C}(!A) \text{ iff } x \in \mathcal{C}(A^\perp \| A) \;\&\; \forall c \in x.\ pol_{A^\perp \| A}(c) = + \implies \bar{c} \in x\,.$$

The *copycat* strategy $\mathrm{cc}_A : A^\perp \| A$ is defined by taking

$$\mathrm{cc}_A = \{(x, \leq_{!A} \upharpoonright x) \mid x \in \mathcal{C}(!A)\}\,.$$

In other words, cc_A consists of all the finite configurations of $!A$, each understood as a finite partial order through inheriting the causal dependency of $!A$.

5 Composition of strategies

A play of a strategy σ in a game $A^\perp \| B$ and a play of a strategy τ in a game $B^\perp \| C$ can interact at the common game B, where the two strategies adopt complementary views, in which one sees a move of Player the other sees a move of Opponent, and *vice versa*. In effect, the two plays synchronise at common moves in B, one strategy being receptive to the Player moves of the other. Together they produce a play in the event structure with polarity $A^\perp \| B^0 \| C$ – the event structure with polarity B^0 has the same underlying event structure as B but where all events now carry neutral polarity. This is because the interaction over the game B produces moves which are no longer open to Player or Opponent.

We can express this interaction through a partial operation

$$\circledast : \mathrm{Plays}(B^\perp \| C) \times \mathrm{Plays}(A^\perp \| B) \rightharpoonup \mathrm{Plays}(A^\perp \| B^0 \| C)$$

defined as follows. Let $p \in \mathrm{Plays}(A^\perp \| B)$, $q \in \mathrm{Plays}(A^\perp \| B)$ with $|p| = x_{A^\perp} \| x_B$ and $|q| = y_{B^\perp} \| y_C$. Take

$$q \circledast p =_{\mathrm{def}} (p \| y_C) \wedge (x_{A^\perp} \| q)\,,$$

where we understand the configurations y_C and $x_{A^\perp}$ to inherit the partial order of their ambient event structures. Notice that $q \circledast p$ is defined only if $x_B = y_{B^\perp}$, and then only if no causal loops are introduced.

▶ **Lemma 8.** *Let $p \in \mathrm{Plays}(A^\perp \| B)$ and $q \in \mathrm{Plays}(B^\perp \| C)$. Then, if defined, $q \circledast p \in \mathrm{Plays}(A^\perp \| B^0 \| C)$.*

Define the *projection*

$$(\_)\downarrow : \mathrm{Plays}(A^\perp \| B^0 \| C) \to \mathrm{Plays}(A^\perp \| C)\,,$$

of a play p in $A^\perp \| B^0 \| C$,with $|p| = x_{A^\perp} \| x_B \| x_C$, to a play $p\downarrow$ in $A^\perp \| C$, to be the restriction of the order on p to the set $x_{A^\perp} \| x_C$.

Define a partial operation

$$\odot : \mathrm{Plays}(B^\perp \| C) \times \mathrm{Plays}(A^\perp \| B) \rightharpoonup \mathrm{Plays}(A^\perp \| C)$$

by

$$q \odot p = (q \circledast p)\downarrow$$

for $p \in \mathrm{Plays}(A^\perp \| B)$ and $q \in \mathrm{Plays}(B^\perp \| C)$.

▶ **Lemma 9.** *Let $p \in \mathrm{Plays}(A^\perp \| B)$ and $q \in \mathrm{Plays}(B^\perp \| C)$. Then, if defined, $q \odot p \in \mathrm{Plays}(A^\perp \| C)$.*

Let $\sigma : A^\perp \| B$ and $\tau : B^\perp \| C$ be strategies. Define their composition

$$\tau \odot \sigma = \{q \odot p \mid p \in \sigma \ \&\ q \in \tau \ \&\ q \odot p \text{ is defined}\}\,.$$

It is sometimes useful to consider their composition *without hiding*, the *interaction*

$$\tau \circledast \sigma = \{q \circledast p \mid p \in \sigma \ \&\ q \in \tau \ \&\ q \circledast p \text{ is defined}\}\,,$$

which is like the strategy $\tau \odot \sigma$, but before hiding the neutral moves over the game B.

▶ **Lemma 10.** *The interaction of strategies $\sigma : A^\perp \| B$ and $\tau : B^\perp \| C$ yields a bare strategy $\tau \circledast \sigma : A^\perp \| B^0 \| C$.*

▶ **Theorem 11.** *The composition of strategies $\sigma : A^\perp \| B$ and $\tau : B^\perp \| C$ yields a strategy $\tau \odot \sigma : A^\perp \| C$. Taking objects to be games and arrows from a game A to a game B to be strategies in the game $A^\perp \| B$, with composition as above, yields a category in which copycat is identity. (This is in contrast to the* bicategory *of [14].)*

5.1 Deterministic strategies

Let A be an event structure with polarity. A bare strategy $\sigma : A$ is *deterministic* iff

$$p \hookrightarrow^+ q \ \&\ p \hookrightarrow r \text{ in } \sigma \implies \exists s \in \sigma.\ q \hookrightarrow s \ \&\ r \hookrightarrow s\,.$$

The interaction of deterministic bare strategies is deterministic. Similarly, the composition of deterministic strategies is deterministic. However, for general games A, the copycat strategy need not be deterministic. It will be deterministic iff A is *race-free*, *i.e.*,

$$x \subseteq^+ y \ \&\ x \subseteq^- z \implies y \cup z \in \mathcal{C}(A)\,.$$

Restricting to race-free games as objects and deterministic strategies as arrows we obtain a category. Deterministic strategies coincide with the *receptive* ingenuous strategies of Melliès and Mimram [13] and are closely related to the strategies of Faggian and Piccolo [8], and Abramsky and Melliès' strategies as closure operators [1].

The subcategory of deterministic strategies on games which countable and purely positive, *i.e.* for which there are no Opponent moves, is isomorphic to that of Berry's dI-domains and stable functions. If we restrict the subcategory further to objects in which causal dependency is simply the identity relation we obtain Girard's qualitative domains with linear maps and if yet further insist that consistency Con is determined in a binary fashion, *i.e.*

$$X \in \mathrm{Con} \iff \forall a_1, a_2 \in X.\ \{a_1, a_2\} \in Con\,,$$

his coherence spaces. In this sense we can see strategies as extending the world of stable domain theory. The relationship with the broader world of traditional domain theory, following in the footsteps of Scott, is more subtle. In [23], it is shown how a strategy determines a presheaf and a strategy between games a profunctor, giving a relationship with a form of generalised domain theory [10, 4].

6 Strategies as maps of event structures

A strategy σ in a game A is a rigid family and so, by Proposition 2, determines an event structure S whose events are those plays in σ which have a top element. Each top element is an event of the game A so there is a function from the events of S to those of A; this function is a total map of event structures and indeed a *concurrent strategy* in the sense of [14]. Not all the concurrent strategies of [14] are obtained this way. But any concurrent strategy of [14] has a rigid image [25] which corresponds to a strategy as presented here. Though not essential to the rest of the paper, we now explain this summary of the relation with the concurrent strategies of [14] in more detail.

Recall a *(total) map* of event structures $f : E \to E'$ is a function f from E to E' such that the image of a configuration x is a configuration fx and any event of fx arises as the image of a unique event of x. Maps compose as functions. Write $\mathcal{E}$ for the ensuing category.

A map $f : E \to E'$ reflects causal dependency locally, in the sense that if e, e' are events in a configuration x of E for which $f(e') \leq f(e)$ in E', then $e' \leq e$ also in E; the event structure E inherits causal dependencies from the event structure E' via the map f. Consequently, a map $f : E \to E'$ preserves concurrency: if two events are concurrent, $e_1 \mathrel{co_E} e_2$, then their images are also concurrent, $f(e_1) \mathrel{co_{E'}} f(e_2)$. In general a map of event structures need not preserve causal dependency; when it does we say it is *rigid*. Write $\mathcal{E}_r$ for the subcategory of rigid maps.

The inclusion functor $\mathcal{E}_r \hookrightarrow \mathcal{E}$ has a right adjoint ([20], Proposition 2.3): There is an obvious map of event structures $\epsilon_B : \Pr(\mathrm{Aug}(B)) \to B$ taking an event of $\Pr(\mathrm{Aug}(B))$ to its top element. Post-composition by ϵ_B yields a bijection

$$\epsilon_B \circ \_ : \mathcal{E}_r(A, \Pr(\mathrm{Aug}(B))) \cong \mathcal{E}(A, B)\,,$$

furnishing the data required for an adjunction. Hence $\Pr(\mathrm{Aug}(\_))$ extends to a right adjoint to the inclusion $\mathcal{E}_r \hookrightarrow \mathcal{E}$. From the bijection of the adjunction, we have a correspondence between maps $f : A \to B$ and rigid maps $\bar{f} : A \to \Pr(\mathrm{Aug}(B))$. The adjunction is unchanged by the addition of polarity to event structures; maps are assumed to preserve polarity.

A strategy determines a map and indeed a "concurrent strategy"as in [14]:

▶ **Proposition 12.** *Let $\sigma : A$ be a strategy in a game A. The function $f_\sigma : \Pr(\sigma) \to A$, taking an event of $\Pr(\sigma)$ to its top element, is a map of event structures with polarity. It is a concurrent strategy in the sense of [14], viz. a map which is*

- courteous, *$s' \rightarrowtriangle s$ and $pol(s') = +$ or $pol(s) = -$ in $\Pr(\sigma)$ implies $f_\sigma(s') \rightarrowtriangle_A f_\sigma(s)$ in A, (called* "innocent" *in [14]), and*
- receptive, *$f_\sigma x \sqsubseteq^- y$ in $\mathcal{C}(A)$, for $x \in \mathcal{C}(\Pr(\sigma))$, implies there is a unique $x' \in \mathcal{C}(\Pr(\sigma))$ such that $f_\sigma x' = y$.*

Not all the concurrent strategies of [14] are obtained in the manner of Proposition 12. However, from any concurrent strategy $f : S \to A$ in a game A there is $\sigma : A$ obtained as the image

$$\sigma =_{\mathrm{def}} \{\theta(\bar{f}x) \in \mathrm{Aug}(A) \mid x \in \mathcal{C}(S)\}$$

of the finite configurations of S as augmentations of A; recall from Proposition 2, the order isomorphism $\theta : \mathcal{C}(\Pr(\mathrm{Aug}(A))) \cong \mathrm{Aug}(A)$. From the definition of σ, the rigid map $\bar{f} : S \to \Pr(\mathrm{Aug}(A))$ cuts down to a rigid map $\bar{f} : S \to \Pr(\sigma)$. The concurrent strategy f factors through its "rigid image" $f_\sigma : \Pr(\sigma) \to A$ in that

$$f : S \xrightarrow{\bar{f}} \Pr(\sigma) \xrightarrow{f_\sigma} A\,,$$

where the rigid image f_σ is itself a concurrent strategy. The simple strategies of this article correspond to such rigid image strategies.

The determination of a strategy, call it σ_f, from a concurrent strategy f is functorial: identity, copycat, strategies are preserved and if concurrent strategies f and g are composable then $\sigma_{g \odot f} = \sigma_g \odot \sigma_f$. Often extra structure on a concurrent strategy f can be pushed forward along the rigid map $\bar{f}$ from to its rigid image, so to a simple strategy of this article. For example, probabilistic structure (in the form of a valuation – see the next section) making a concurrent strategy probabilistic can be pushed forward along the rigid map $\bar{f}$ from S to $\Pr(\sigma_f)$, and so to σ_f [25]. As a consequence, in the next section, we are able to develop probabilistic strategies in the simpler framework of this paper.

A major result of [14] is that receptivity and courtesy (called innocence there) are necessary and sufficient conditions in order for copycat to behave as identity w.r.t. composition; this motivated the definition of concurrent strategy there. That article directly spawned work on games with winning conditions and payoff [5, 6], imperfect information [21], probabilistic strategies [24], "stopping configurations" [3] and "essential events" [16] – the latter two concerned with capturing the liveness behaviour of concurrent strategies viewed as processes. (Concurrent strategies are currently being extended to cope with quantum computation of the kind addressed in the quantum lambda calculus [15].) As an indication of how much of the work ensuing from [14] could be reformulated in terms of the simple strategies on which this article concentrates we next address the issue of how to make strategies probabilistic. Probabilistic strategies developed in this simpler framework, instead of that of concurrent strategies [14], do not suffer from any loss of information *e.g.* with regard to expected payoff.

7 Probabilistic strategies

As a first step we describe how to make event structures probabilistic, in itself an issue, as event structures lie outside the models of probabilistic processes most commonly considered.

7.1 Probabilistic event structures

A probabilistic event structure essentially comprises an event structure together with a continuous valuation on the Scott-open sets of its domain of configurations.[1] The continuous valuation assigns a probability to each open set and can then be extended to a probability measure on the Borel sets [11]. However open sets are several levels removed from the events of an event structure, and an equivalent but more workable definition is obtained by considering the probabilities of sub-basic open sets, generated by single finite configurations; for each finite configuration x this specifies $\mathrm{Prob}(x)$ the probability of obtaining events x, so as result a configuration which extends the finite configuration x. Such valuations on configuration determine the continuous valuations from which they arise, and can be characterised through the device of "drop functions" which measure the drop in probability across certain generalised intervals. The characterisation yields a workable general definition of probabilistic event structure as event structures with *configuration-valuations*, *viz.* functions

[1] A *Scott-open* subset of configurations is upwards-closed w.r.t. inclusion and such that if it contains the union of a directed subset S of configurations then it contains an element of S. A *continuous valuation* is a function w from the Scott-open subsets of $\mathcal{C}^\infty(E)$ to $[0,1]$ which is *((normalized)* $w(\mathcal{C}^\infty(E)) = 1$; *(strict)* $w(\varnothing) = 0$; *(monotone)* $U \subseteq V \implies w(U) \leq w(V)$; *(modular)* $w(U \cup V) + w(U \cap V) = w(U) + w(V)$; and *(continuous)* $w(\bigcup_{i \in I} U_i) = \sup_{i \in I} w(U_i)$, for *directed* unions.

from finite configurations to the unit interval for which the drop functions are always nonnegative [22].

In detail, a *probabilistic event structure* comprises an event structure E with a *configuration-valuation*, a function v from the finite configurations of E to the unit interval which is

- *(normalized)* $v(\varnothing) = 1$ and satisfies the
- *(drop condition)* $d_v[y; x_1, \cdots, x_n] \geq 0$ when $y \subseteq x_1, \cdots, x_n$ for finite configurations $y, x_1, \cdots, x_n$;

where the "drop" across the generalized interval starting at y and ending at one of the $x_1, \cdots, x_n$ is given by

$$d_v[y; x_1, \cdots, x_n] =_{\text{def}} v(y) - \sum_I (-1)^{|I|+1} v(\bigcup_{i \in I} x_i)$$

– the index I ranges over nonempty $I \subseteq \{1, \cdots, n\}$ such that the union $\bigcup_{i \in I} x_i$ is a configuration. The "drop" $d_v[y; x_1, \cdots, x_n]$ gives the probability of the result being a configuration which includes the configuration y and does not include any of the configurations $x_1, \cdots, x_n$.

If $x \subseteq y$ in $\mathcal{C}(E)$, then, provided $v(x) \neq 0$, the conditional probability $\text{Prob}(y \mid x)$ is $v(y)/v(x)$; this is the probability that the resulting configuration includes the events y conditional on it including the events x.

7.2 Probability with an Opponent

This prepares the ground for a definition of probabilistic distributed strategies. Firstly though, we should restrict to race-free games, in particular because without copycat being deterministic there would be no probabilistic identity strategies. A probabilistic strategy in a game A, is a strategy $\sigma : A$ in which we endow σ with probability, while taking account of the fact that in the strategy Player can't be aware of the probabilities assigned by Opponent. To this end we notice that σ, being a rigid family, has the form of a family of configurations. We can't just regard σ as a probabilistic event structure however. This is because Player is oblivious to the probabilities of Opponent moves beyond those determined by causal dependencies of σ. An appropriate *valuation* for σ needs to take account of Opponent moves. It turns out to be useful to extend the concept of valuation to *bare* strategies, which may also have neutral moves.

Let $\sigma : A$ be a bare strategy in A, an event structure with polarity; so both A and σ may involve neutral moves. A *valuation* on σ is a function v, from σ to the unit interval, which is

- *(normalized)* $v(\varnothing) = 1$,
- *(oblivious)* $v(p) = v(q)$ when $p \hookrightarrow^- q$ for $p, q \in \sigma$, and satisfies the
- *(drop condition)* $d_v[q; p_1, \cdots, p_n] \geq 0$ when $q \hookrightarrow^+ p_1, \cdots, p_n$ for elements of σ.

When $p \hookrightarrow^+ q$ in σ, we can still express $\text{Prob}(q \mid p)$, the conditional probability of the additional neutral or Player moves making the play q given p, as $v(q)/v(p)$, provided $v(p) \neq 0$. The game being race-free and the valuation being oblivious ensure the probabilistic independence of Player or neutral moves and Opponent moves with which are concurrent.

For a race-free game A, the copycat strategy is deterministic and we obtain a valuation on cc_A by taking v_{cc_A} to be the function which is constantly 1.

7.3 Composing probabilistic strategies

Let A, B and C be race-free games. Assume $\sigma : A^\perp \| B$, with valuation v_σ, and $\tau : B^\perp \| C$, with valuation v_τ, are probabilistic strategies. To define their interaction and composition we must define the valuations $v_\tau \circledast v_\sigma$ on $\tau \circledast \sigma$ and $v_\tau \odot v_\sigma$ on $\tau \odot \sigma$, respectively.

▶ **Lemma 13.** *For $r \in \tau \circledast \sigma$, defining*

$$(v_\tau \circledast v_\sigma)(r) =_{\text{def}} \sum \{v_\tau(q).v_\sigma(p) \mid q \circledast p = r\},$$

yields a valuation on $\tau \circledast \sigma$.

▶ **Lemma 14.** *For $r \in \tau \odot \sigma$, defining*

$$(v_\tau \odot v_\sigma)(r) =_{\text{def}} \sum \{v_\tau(q).v_\sigma(p) \mid q \odot p = r\},$$

yields a valuation on $\tau \odot \sigma$.

▶ **Theorem 15.** *For race-free games A, B and C, we define the composition of probabilistic strategies σ from A to B, with valuation v_σ, and τ from B to C, with valuation v_τ, to be $\tau \odot \sigma$, with valuation $v_\tau \odot v_\sigma$. Taking objects to be games and arrows from a game A to a game B to be probabilistic strategies in the game $A^\perp \| B$, with composition as above, yields a category in which copycat, with the constantly-1 valuation, is identity.*

The next example illustrates how through probability leaks we can track deadlocks, or divergences, that can arise in the composition of strategies. (Such branching behaviour might otherwise be lost in the composition of strategies and through concentrating on rigid images.)

▶ **Example 16.** Let B be the game consisting of two concurrent Player events b_1 and b_2, and C the game with a single Player event c. We illustrate the composition of two probabilistic strategies σ from the empty game $\varnothing$ to B and τ from B to C. The strategy $\sigma : \varnothing^\perp \| B$ plays b_1 with probability 2/3 and b_2 with probability 1/3 (and plays both with probability 0). The strategy $\tau : B^\perp \| C$ does nothing if just b_1 is played and plays the single Player event c of C with certainty, probabilty 1, if b_2 is played. Their composition yields the strategy $\tau \odot \sigma : \varnothing^\perp \| C$ which plays c with probability 1/3, so has a 2/3 chance of doing nothing.

One way in which the probabilistic interaction of strategies is important is in calculating the expected outcome of the competition between a probabilistic strategy and a counterstrategy, the subject of the following example.

▶ **Example 17.** Given a probabilistic strategy $\sigma : A$, with valuation v_σ, and a counterstrategy $\tau : A^\perp$, with valuation v_τ, we obtain a valuation $v_\tau \circledast v_\sigma$ on their interaction $\tau \circledast \sigma : A^0$, where now all the events of the interaction are neutral. Via the order isomorphism $\theta : \mathcal{C}(\Pr(\tau \circledast \sigma)) \cong \tau \circledast \sigma$ we obtain a configuration-valuation $(v_\tau \circledast v_\sigma) \circ \theta$, making $\Pr(\tau \circledast \sigma)$ a probabilistic event structure. As such we get a probability measure $\mu_{\sigma,\tau}$ on the Borel sets of its configurations. Assuming a *payoff* given as a Borel measurable function X from $\mathcal{C}^\infty(A)$ to the real numbers, the *expected payoff* is obtained as the Lebesgue integral

$$\mathbf{E}_{\sigma,\tau}(X) =_{\text{def}} \int_{x \in \mathcal{C}^\infty(\Pr(\tau \circledast \sigma))} X(|x|)\, d\mu_{\sigma,\tau}(x),$$

where $|x| \in \mathcal{C}^\infty(A)$ is the configuration of A over which $x \in \mathcal{C}^\infty(\Pr(\tau \circledast \sigma))$ lies.

8 Conclusion

We have provided an elementary account of a form of distributed strategies by choosing only to represent the rigid images of concurrent strategies. Is anything irredeemably lost through this simplification? (In the sense that it can't be regained through adding extra structure, in the way that probabilistic structure recovers hidden branching.) Not obviously. Though, for

instance, we couldn't exactly reproduce the result of [3], establishing a bijection between events of a strategy and derivations in an operational semantics. Though an elementary account is more accessible, a more abstract, categorical account can be helpful too. As often, there are pros and cons. To some extent, one pays for the elementary treatment in not seeing the abstract picture, the wood for the trees.

On another tack, the account of strategies here reveals an alternative way to develop strategies while capturing noneteristic branching explicitly, *viz.* as (pre)sheaves over plays rather than subsets, in the form of rigid families. For instance, we can recover the concurrent strategies of [14] as certain separated presheaves in the manner of [19]; this brings us close to the developments of Hirschowitz and Pous [9] and Ong and Tsukada [17].

Acknowledgments. Thanks to Nathan Bowler, Jonathan Hayman and Martin Hyland for advice and encouragement, and to the ERC for the Advanced Grant "Events, Causality and Symmetry" (ECSYM).

References

1 Samson Abramsky and Paul-André Melliès. Concurrent games and full completeness. In *LICS '99*. IEEE Computer Society, 1999.
2 Simon Castellan and Pierre Clairambault. Causality vs interleavings in concurrent games semantics. In *CONCUR'16*, 2016.
3 Simon Castellan, Jonathan Hayman, Marc Lasson, and Glynn Winskel. Strategies as concurrent processes. *ENTCS*, 308, 2014.
4 Gian Luca Cattani and Glynn Winskel. Profunctors, open maps and bisimulation. *Mathematical Structures in Computer Science*, 15(3):553–614, 2005.
5 Pierre Clairambault, Julian Gutierrez, and Glynn Winskel. The winning ways of concurrent games. In *LICS 2012: 235-244*, 2012.
6 Pierre Clairambault and Glynn Winskel. On concurrent games with payoff. *Electr. Notes Theor. Comput. Sci. 298: 71-92*, 2013.
7 John Conway. *On Numbers and Games*. Wellesley, MA: A K Peters, 2000.
8 Claudia Faggian and Mauro Piccolo. Partial orders, event structures and linear strategies. In *TLCA '09*, volume 5608 of *LNCS*. Springer, 2009.
9 Tom Hirschowitz and Damien Pous. Innocent strategies as presheaves and interactive equivalences for CCS. *Sci. Ann. Comp. Sci.*, 22(1):147–199, 2012.
10 Martin Hyland. Some reasons for generalising domain theory. *Mathematical Structures in Computer Science*, 20(2):239–265, 2010.
11 Claire Jones and Gordon Plotkin. A probabilistic powerdomain of valuations. In *LICS '89*. IEEE Computer Society, 1989.
12 Andre Joyal. Remarques sur la théorie des jeux à deux personnes. *Gazette des sciences mathématiques du Québec, 1(4)*, 1997.
13 Paul-André Melliès and Samuel Mimram. Asynchronous games: Innocence without alternation. In *CONCUR*, volume 4703 of *LNCS*, pages 395–411, 2007.
14 Silvain Rideau and Glynn Winskel. Concurrent strategies. In *LICS 2011*.
15 Peter Selinger and Benoît Valiron. Quantum lambda calculus. In Simon Gay and Ian Mackie, editors, *Semantic Techniques in Quantum Computation*, chapter 4, pages 135–172. Cambridge University Press, 2009.
16 Jonathan Hayman Simon Castellan, Pierre Clairambault and Glynn Winskel. Non-angelic concurrent game semantics. 2016.

17 Takeshi Tsukada and C.-H. Luke Ong. Nondeterminism in game semantics via sheaves. In *LICS 2015*. IEEE Computer Society, 2015.

18 Glynn Winskel. Event structure semantics for CCS and related languages. In *ICALP'82*, LNCS 140, 1982.

19 Glynn Winskel. Event structures as presheaves -two representation theorems. In *CONCUR '99*, 1999.

20 Glynn Winskel. Event structures with symmetry. *Electr. Notes Theor. Comput. Sci. 172: 611-652*, 2007.

21 Glynn Winskel. Winning, losing and drawing in concurrent games with perfect or imperfect information. In *Festschrift for Dexter Kozen*, volume 7230 of *LNCS*. Springer, 2012.

22 Glynn Winskel. Distributed probabilistic and quantum strategies. *ENTCS 298*, 2013.

23 Glynn Winskel. Strategies as profunctors. In *FOSSACS 2013*, volume 7794 of *LNCS*. Springer, 2013.

24 Glynn Winskel. Probabilistic and quantum event structures. In *Festschrift for Prakash Panangaden*, volume 8464 of *LNCS*. Springer, 2014.

25 Glynn Winskel. *ECSYM Notes: Event Structures, Stable Families and Concurrent Games*. http://www.cl.cam.ac.uk/~gw104/ecsym-notes.pdf, 2016.

On Definable and Recognizable Properties of Graphs of Bounded Treewidth

Michał Pilipczuk

Institute of Informatics, University of Warsaw, Warsaw, Poland
michal.pilipczuk@mimuw.edu.pl

Abstract

This is an overview of the invited talk delivered at the 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017).

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases monadic second-order logic, treewidth, recognizability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.82

Category Invited Talk

1 Overview of the talk

The foundational observation of the field of automata and logic is that on many well-behaved classes of structures the notion of *recognizability* of a structure's property by a finite-state automaton is equivalent to *definability* of this property in *monadic second-order logic* (MSO). This equivalence holds for properties of words and trees, both finite and infinite, and provides means for the algorithmic treatment of MSO-definable properties in these classes of structures.

It is natural to ask what aspects of a class of structures imply that the notions of MSO-definability and recognizability by (appropriately defined) finite automata coincide on this class. In early 90s, Courcelle [2] proved that on any class of structures of bounded treewidth, that is, where structures roughly look like trees with trunks of width bounded by a constant, one implication holds: MSO-definability implies recognizability. This fundamental result already provides most of the desired algorithmic corollaries, most notably that MSO-definable properties of structures of bounded treewidth can be decided in linear fixed-parameter time, where treewidth is the parameter. However, the reverse implication became known as the *Courcelle's conjecture* and remained open until very recently, despite multiple attempts and some incomplete proofs [3, 4]. Finally, last year together with Bojańczyk we resolved the conjecture in affirmative [1].

The main obstacle when approaching the Courcelle's conjecture is that the constructed MSO formula expressing the recognizable property in question has to work only on the structure, and not on its tree decomposition certifying the constant upper bound on the treewidth. More precisely, if we were given such a tree decomposition, then we could just existentially quantify an accepting run of the automaton recognizing the property. However, while graphs of treewidth 1, that is, forests, can be perfectly understood, for structures of larger treewidth computing an optimum-width tree decomposition is a highly nontrivial combinatorial task, and an appropriate tree decomposition cannot be immediately defined from the structure. Therefore, the main technical contribution of [1] is a proof that an approximate tree decomposition of a structure can be constructed by means of a *nondeterministic MSO transduction*, which is a formalism that captures MSO-definable

transformations of relational structures. The crucial ingredient of this proof is an application of Simon's factorization forest theorem [5].

During the talk, we will discuss the relation between MSO-definability and recognizability on various classes of structures, in particular on classes of bounded treewidth. We will also give a sketch of the proof of the Courcelle's conjecture, focusing on the role played by Simon's factorization forest theorem.

References

1 Mikołaj Bojańczyk and Michał Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In *LICS'16*, pages 407–416. ACM, 2016. Full version available as arXiv preprint 1605.03045.

2 Bruno Courcelle. The Monadic Second-Order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

3 Valentine Kabanets. Recognizability equals definability for partial k-paths. In *ICALP'97*, volume 1256 of *LNCS*, pages 805–815. Springer, 1997.

4 Denis Lapoire. Recognizability equals Monadic Second-Order definability for sets of graphs of bounded tree-width. In *STACS'98*, volume 1373 of *LNCS*, pages 618–628. Springer, 1998.

5 Imre Simon. Factorization forests of finite height. *Theor. Comput. Sci.*, 72(1):65–94, 1990.

Hardness and Approximation of High-Dimensional Search Problems

Rasmus Pagh*

IT University of Copenhagen, Denmark
pagh@itu.dk

Abstract

The need to perform search in a collection of high-dimensional vectors arises in many areas of computer science including databases, image analysis, information retrieval, and machine learning. In contrast to lower-dimensional settings, we do not know of worst-case efficient data structures for such search problems. In fact, if we make no assumptions on the input there is no known way of doing significantly better than brute force. In this talk I survey recent developments in the theoretical study of high dimensional search problems, including:

- Conditional hardness results linking search problems to well-known computationally hard problems such as k-SAT.
- Upper bounds for *approximate* high-dimensional search using locality-sensitive maps and filters, and work towards derandomizing these algorithms.
- Surprising upper bounds in *batched* settings where there are many simultaneous searches.

The talk ends by sketching directions for future research, connecting to other areas of theoretical computer science but also attempting to obtain better theoretical models that explain the performance of search algorithms that are used in practice.

1998 ACM Subject Classification E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems – Geometrical problems and computations

Keywords and phrases Similarity search, hardness, approximation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.83

Category Invited Talk

* Supported by the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331.

Temporal Logics for Multi-Agent Systems

Nicolas Markey[*†]

IRISA – CNRS & INRIA & Univ. Rennes 1, France

Abstract

This is an overview of an invited talk delivered during the 42nd International Conference on Mathematical Foundations of Computer Science (MFCS 2017).

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic

Keywords and phrases Temporal logics, Verification, Game theory, Strategic reasoning

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.84

Category Invited Talk

1 Overview of the talk

Temporal logics have been widely used in model checking over the last 40 years, as a formalism for reasoning about executions of computer systems. They are sufficiently powerful to specify most properties one may want to check of reactive systems, while enjoying reasonably-efficient verification algorithms [21, 11, 22, 12, 7, 6]. Temporal logics and model checking have had a major impact in computer science (as witnessed by two Turing awards won by Pnueli in 1996, and by Clarke, Emerson and Sifakis in 2007), and have been applied in numerous industrial cases.

Several attempts have been made to extend temporal logics to multi-agent systems, where several components interact: while the *Computation-Tree Logic* (CTL) can only express the existence (or absence) of executions of the global system having certain properties, the aim here is to quantify over the possible behaviours of the individual components interacting in the system (be it in a collaborative or adversarial way).

In 1997, CTL has been extended into the *Alternating-time Temporal Logic* (ATL), with the introduction of *strategy quantifiers* [3, 4]. In ATL, strategy quantifiers express the existence (or absence) of a behaviour of one of the agents (or of a coalition) so that any resulting execution in the global system satisfies a given property (notice in particular that such an existential quantification over strategies involves an implicit universal quantification over the resulting executions). The semantics of ATL formulas as defined in [4] is *bottom-up*: when evaluating a formula with nested strategy quantifiers, the innermost quantifiers are evaluated first. While this allows for efficient model-checking algorithms, this prevents strategic interactions: the innermost quantifier being evaluated first, it can be replaced with a fresh atomic proposition labelling those states where the subformula holds true.

During the 2000's, several adaptations of ATL have been proposed in order to introduce strategy interactions [23, 24, 1, 20], until the development of a *top-down semantics*, storing selected strategies in a context for later interaction with other strategies [5, 2, 9, 13]. This

* Supported by ERC grant EQualIS (308087).

† This abstract is based on joint works with Patricia Bouyer, Patrick Gardy, and François Laroussinie.

results in a much richer framework, suitable for expressing classical game-theoretic properties (such as the existence of Nash equilibria) and many extra properties mixing collaborative and adversarial interactions (such as the interactions between a server and several clients competing for accessing some shared resource). Such an expressiveness has a cost, and checking if a formula in ATL with strategy contexts holds in a given model is in k-EXPTIME, where k is the number of nested strategy quantifiers in the formula.

Simultaneously, an orthogonal approach has been defined and explored: it allows to manipulate strategies explicitly, quantifying over them and assigning them to agents [10, 19, 17]. The resulting logic, called *Strategy Logic* (SL), has similar algorithmic properties as ATL with strategy contexts [17, 16, 15], but allows for even more expressive power (e.g., strategies can be revoked and applied again later, or two players can follow the same strategy). However, recent works have shown that slight natural variations in the semantics of SL may have significant impact both on the algorithmics and on the expressiveness of the logic [18, 8, 14].

During this talk, we survey these results, giving a uniform presentation of the verification and expressiveness results for those logics and their semantic variants.

References

1. T. Ågotnes, V. Goranko, and W. Jamroga. Alternating-time temporal logics with irrevocable strategies. In TARK'07, p. 15–24, 2007.
2. T. Ågotnes, V. Goranko, and W. Jamroga. Strategic commitment and release in logics for multi-agent systems (extended abstract). Technical Report 08-01, Clausthal University of Technology, Germany, 2008.
3. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In FOCS'97, p. 100–109. IEEE Comp. Soc. Press, 1997.
4. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. of the ACM*, 49(5):672–713, 2002.
5. Ch. Baier, T. Brázdil, M. Größer, and A. Kučera. Stochastic game logic. In QEST'07, p. 227–236. IEEE Comp. Soc. Press, 2007.
6. Ch. Baier and J.-P. Katoen. *Principles of Model-Checking*. MIT Press, 2008.
7. B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer, 2001.
8. P. Bouyer, P. Gardy, and N. Markey. On the semantics of strategy logic. *Information Processing Letters*, 116(2):75–79, 2016.
9. Th. Brihaye, A. Da Costa, F. Laroussinie, and N. Markey. ATL with strategy contexts and bounded memory. In LFCS'09, LNCS 5407, p. 92–106. Springer, 2009.
10. K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In CONCUR'07, LNCS 4703, p. 59–73. Springer, 2007.
11. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In LOP'81, LNCS 131, p. 52–71. Springer, 1982.
12. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 2000.
13. A. Da Costa, F. Laroussinie, and N. Markey. ATL with strategy contexts: Expressiveness and model checking. In FSTTCS'10, LIPIcs 8, p. 120–132. Leibniz-Zentrum für Informatik, 2010.
14. P. Gardy. *Semantics of Strategy Logic*. PhD thesis, Lab. Spécification & Vérification, Univ. Paris-Saclay, France, 2017.
15. F. Laroussinie and N. Markey. Quantified CTL: expressiveness and complexity. *Logical Methods in Computer Science*, 10(4), 2014.

16 F. Laroussinie and N. Markey. Augmenting ATL with strategy contexts. *Inf. & Comp.*, 245:98–123, 2015.

17 F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. Reasoning about strategies: On the model-checking problem. *ACM Transactions on Computational Logic*, 15(4):34:1–34:47, 2014.

18 F. Mogavero, A. Murano, and L. Sauro. A behavioral hierarchy of strategy logic. In CLIMA'14, LNAI 8624, p. 148–165. Springer, 2014.

19 F. Mogavero, A. Murano, and M. Y. Vardi. Reasoning about strategies. In FSTTCS'10, LIPIcs 8, p. 133–144. Leibniz-Zentrum für Informatik, 2010.

20 S. Pinchinat. A generic constructive solution for concurrent games with expressive constraints on strategies. In ATVA'07, LNCS 4762, p. 253–267. Springer, 2007.

21 A. Pnueli. The temporal logic of programs. In FOCS'77, p. 46–57. IEEE Comp. Soc. Press, 1977.

22 J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In SOP'82, LNCS 137, p. 337–351. Springer, 1982.

23 W. van der Hoek, W. Jamroga, and M. Wooldridge. A logic for strategic reasoning. In AAMAS'05, p. 157–164. ACM Press, 2005.

24 D. Walther, W. van der Hoek, and M. Wooldridge. Alternating-time temporal logic with explicit strategies. In TARK'07, p. 269–278, 2007.

Ideal-Based Algorithms for the Symbolic Verification of Well-Structured Systems

Philippe Schnoebelen

LSV, CNRS & ENS Paris Saclay, Cachan, France
phs@lsv.fr

Abstract

We explain how the downward-closed subsets of a well-quasi-ordering $(X, \leq)$ can be represented via the ideals of X and how this leads to simple and efficient algorithms for the verification of well-structured systems.

1998 ACM Subject Classification F.1.1 Models of Computation, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Well-structured systems and verification, Order theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.85

Category Invited Talk

1 Summary of the talk

Well-structured systems, also known under the acronym "WSTSs", are a family of infinite-state models for which generic verification algorithms exist [1, 2, 13, 18, 23]. With WSTSs, the main ingredient for decidability is the existence of an ordering on configurations that enjoys two properties:

- it is a well-quasi-ordering (a WQO): every infinite sequence $c_0, c_1, c_2, \ldots$ of configurations contains an increasing pair $c_i \leq c_j$ with $i < j$;
- transitions are monotonic: if the system can perform a step $c \to c'$ then from any configuration $d \geq c$, a "similar" step is possible, i.e., there is some $d \to d'$ with $d' \geq c'$.

The most well-known instances of WSTSs are some families of counter machines or vector addition systems [8, 12]. For simplicity, we shall assume that the WQO set of configurations for these systems is $\mathit{Conf} = (\mathbb{N}^d, \leq_\times)$ for some dimension $d \in \mathbb{N}$, where the *component-wise ordering* $\leq_\times$ is given by $\boldsymbol{u} = (u_1, \ldots, u_d) \leq_\times \boldsymbol{v} = (v_1, \ldots, v_d) \stackrel{\text{def}}{\Leftrightarrow} u_1 \leq v_1 \wedge \cdots \wedge u_d \leq v_d$.

Another well-known instance are the lossy channel systems [4, 7], where for simplicity we assume that the set of configurations is $(\Sigma^*, \leq_*)$ for some finite alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots\}$ of messages, and where $\leq_*$ is the *subword ordering* [1] given by

$$u \leq v \stackrel{\text{def}}{\Leftrightarrow} \exists \mathtt{a}_1, \ldots, \mathtt{a}_\ell \in \Sigma : \exists v_0, \ldots, v_\ell \in \Sigma^* : u = \mathtt{a}_1\mathtt{a}_2 \cdots \mathtt{a}_\ell \wedge v = v_0\mathtt{a}_1 v_1 \mathtt{a}_2 \cdots \mathtt{a}_\ell v_\ell \,.$$

Algorithms for the verification of safety properties of WSTSs usually involve reasoning and computing with *upward-closed* and/or *downward-closed* sets of configurations. A set $U \subseteq \mathit{Conf}$ is upward-closed $\stackrel{\text{def}}{\Leftrightarrow} c \in U \wedge c \leq c' \implies c' \in U$, and there is a similar definition

[1] That $(\Sigma^*, \leq_*)$ is a WQO is known as Higman's Lemma.

for downward-closed subsets. These sets are usually infinite (like *Conf* itself) and symbolic representations or data structures are needed in algorithms handling them.

For upward-closed subsets, a well-known representation relies on the existence of *minimal bases*, i.e., the fact that the set of minimal elements of any subset is finite and unique (modulo equivalence). This representation is generic: it works for any WQO. Furthermore, it enjoys several nice algorithmic properties, e.g., testing inclusion between upward-closed subsets reduces to a quadratic number of comparisons between individual configurations, and the union of upward-closed sets is very easy to compute. In the case of $(\mathbb{N}^d, \leq_\times)$ or $(\Sigma^*, \leq_*)$, algorithms for computing intersections reduce to easy computations of least upper bounds between elements.

For downward-closed subsets, one cannot rely on a mirror notion of maximal elements and this makes symbolic computations harder to envision. The question of finding a generic approach for computing with downward-closed sets was first raised in [14].

In the case of $(\mathbb{N}^d, \leq_\times)$, a symbolic technique was popularized by Karp and Miller with their classic algorithm for coverability in VAS [19]. They define $\mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$ —where the set of natural numbers is completed with a new infinite element ω that is larger than any finite number— and consider d-tuples over $\mathbb{N}_\omega$. It turns out that this is exactly what we need to represent downward-closed subsets of $\mathbb{N}^d$. For $\sigma = (s_1, \ldots, s_d) \in \mathbb{N}_\omega^d$, we let $\downarrow\sigma = \{c \in \mathbb{N}^d \mid c \leq_\times \sigma\}$ denote the downward-closed subset of $\mathbb{N}^d$ generated by σ and call it an *ideal* of $(\mathbb{N}^d, \leq_\times)$. Then downward-closed subsets of $\mathbb{N}^d$ can be denoted in a unique way by finite unions of incomparable ideals. Computing unions and intersections with such representations, and deciding inclusion between them, use simple algorithms that are uncannily similar to what happened with the finite-basis representation for upward-closed subsets.

If we now consider $(\Sigma^*, \leq_*)$, a very elegant representation for downward-closed subsets was proposed by Abdulla et al. in [3]. They show that any downward-closed $D \subseteq \Sigma^*$ can be represented by a *simple regular expression* (a SRE), obtained as a union of concatenations of *atoms* of the form Γ^* for a subalphabet $\Gamma \subseteq \Sigma$, or of the form $\mathtt{a} + \epsilon$ for some letter $\mathtt{a} \in \Sigma$. Furthermore, these SREs support simple and efficient algorithms for unions, intersections, comparisons, and more.

It turns out that concatenations of atoms denote exactly the ideals of $(\Sigma^*, \leq_*)$. Formally, an ideal of a WQO $(X, \leq)$ is a nonempty downward-closed directed subset $D \subseteq X$. Being directed means that for all $x, y \in D$ there is some $z \in D$ with $x \leq z \wedge y \leq z$. Given any WQO $(X, \leq)$, the downward-closed subsets of X can be written as unions of finitely many pairwise incomparable ideals, and this decomposition is unique. This property explains the nice algorithmic properties we observed with $\mathbb{N}_\omega^d$ and the SREs over Σ, and it generalizes to any WQO where we can provide effective characterizations for the ideals.

In the second part of the talk, we show how such effective characterizations exist for most of the WQOs one encounters in practice. This is done by considering the most common ways of constructing new WQOs from previous ones (sequence extension, powerset, but also substructures and quotients) and characterizing the ideals of the new WQOs in terms of the ideals of the earlier ones.

We illustrate these constructions with lesser known WSTSs like priority channel systems and higher-order channel systems [17], or data nets [20] and timed-arc Petri nets [5].

Acknowledgments. This talk is based on joint work with J. Goubault-Larrecq, S. Halfon, P. Karandikar, K. Narayan Kumar, S. Schmitz, and it has further profited from many discussions with A. Finkel, J. Leroux and G. Sutre. Most of the presented definitions and

results can be found in recent works like [6, 9, 10, 11, 16, 21, 22]. A full version of these notes is in preparation [15].

References

1. P. A. Abdulla. Well (and better) quasi-ordered transition systems. *Bull. Symbolic Logic*, 16(4):457–515, 2010. doi:10.2178/bsl/1294171129.
2. P. A. Abdulla, K. Čerāns, B. Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1/2):109–127, 2000. doi:10.1006/inco.1999.2843.
3. P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004. doi:10.1023/B:FORM.0000033962.51898.1a.
4. P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996. doi:10.1006/inco.1996.0053.
5. B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. The expressive power of time Petri nets. *Theoretical Computer Science*, 474, 2012. doi:10.1016/j.tcs.2012.12.005.
6. M. Blondin, A. Finkel, and P. McKenzie. Well behaved transition systems. arXiv:1608.02636 [cs.LO], August 2016. To appear in Logical Meth. Comp. Sci. URL: http://arxiv.org/abs/1608.02636.
7. G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996. doi:10.1006/inco.1996.0003.
8. C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of Reset P/T nets. In *CONCUR '99, LNCS 1644*, pages 301–310. Springer, 1999. doi:10.1007/3-540-48523-6_27.
9. A. Finkel. The ideal theory for WSTS. In *RP 2016, LNCS 9899*, pages 1–22. Springer, 2016. doi:10.1007/978-3-319-45994-3_1.
10. A. Finkel and J. Goubault-Larrecq. Forward analysis for WSTS, part I: Completions. In *STACS 2009, LIPIcs 3*, pages 433–444. Leibniz-Zentrum für Informatik, 2009. doi:10.4230/LIPIcs.STACS.2009.1844.
11. A. Finkel and J. Goubault-Larrecq. Forward analysis for WSTS, part II: Complete WSTS. *Logical Methods in Comp. Science*, 8(4), 2012. doi:10.2168/LMCS-8(3:28)2012.
12. A. Finkel, P. McKenzie, and C. Picaronny. A well-structured framework for analysing Petri nets extensions. *Information and Computation*, 195(1–2):1–29, 2004. doi:10.1016/j.ic.2004.01.005.
13. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.
14. G. Geeraerts, J.-F. Raskin, and L. Van Begin. Expand, enlarge and check: New algorithms for the coverability problem of WSTS. *Journal of Computer and System Sciences*, 72(1):180–203, 2006. doi:10.1016/j.jcss.2005.09.001.
15. J. Goubault-Larrecq, S. Halfon, P. Karandikar, K. Narayan Kumar, and Ph. Schnoebelen. The ideal approach to computing closed subsets in well-quasi-orderings. In preparation, 2017.
16. J. Goubault-Larrecq and S. Schmitz. Deciding piecewise testable separability for regular tree languages. In *ICALP 2016, LIPIcs 55*, pages 97:1–97:15. Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.97.
17. Ch. Haase, S. Schmitz, and Ph. Schnoebelen. The power of priority channel systems. *Logical Methods in Comp. Science*, 10(4:4), 2014. doi:10.2168/LMCS-10(4:4)2014.
18. T. A. Henzinger, R. Majumdar, and J.-F. Raskin. A classification of symbolic transition systems. *ACM Trans. Computational Logic*, 6(1):1–32, 2005. doi:10.1145/1042038.1042039.

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

20 R. Lazić, T. Newcomb, J. Ouaknine, A. W. Roscoe, and J. Worrell. Nets with tokens which carry data. *Fundamenta Informaticae*, 88(3):251–274, 2008.

21 R. Lazić and S. Schmitz. The complexity of coverability in ν-Petri nets. In *LICS 2016*, pages 467–476. ACM Press, 2016. doi:10.1145/2933575.2933593.

22 J. Leroux and S. Schmitz. Ideal decompositions for vector addition systems. In *STACS 2016, LIPIcs 47*, pages 1:1–1:13. Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.1.

23 S. Schmitz and Ph. Schnoebelen. The power of well-structured systems. In *CONCUR 2013, LNCS 8052*, pages 5–24. Springer, 2013. doi:10.1007/978-3-642-40184-8_2.