44th International Symposium on Mathematical Foundations of Computer Science

MFCS 2019, August 26–30, 2019, Aachen, Germany

Edited by

Peter Rossmanith
Pinar Heggernes
Joost-Pieter Katoen

LIPIcs

Editors

Peter Rossmanith
RWTH Aachen University, Germany
rossmani@cs.rwth-aachen.de

Pinar Heggernes
University of Bergen, Norway
pinar.heggernes@uib.no

Joost-Pieter Katoen
RWTH Aachen University, Germany
katoen@cs.rwth-aachen.de

ACM Classification 2012
Theory of computation

ISBN 978-3-95977-117-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-117-7.

Publication date
August, 2019

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

Digital Object Identifier: 10.4230/LIPIcs.MFCS.2019.0

ISBN 978-3-95977-117-7 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

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

Preface

The International Symposium on Mathematical Foundations of Computer Science (MFCS conference series) is a well-established venue for presenting research papers in theoretical computer science. The broad scope of the conference encourages interactions between researchers who might not meet at more specialized venues. The first MFCS conference was organized in 1972 in 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 2018 it was held in Liverpool, United Kingdom. MFCS 2019, the 44th edition of MFCS, is the first MFCS being held in Germany.

Out of 198 submitted papers, 78 have been finally accepted. The authors of the submitted papers represent nearly 40 countries. Each paper was assigned to three PC members, who reviewed and discussed them thoroughly over a period of nearly seven weeks. As the co-chairs of the program committee, we would like to express our deep gratitude to all the committee members for their hard, dedicated work. The quality of the submitted papers was very high and many good papers had to be rejected.

MFCS 2019 features five invited talks, by Kurt Mehlhorn (Max-Plack Institute Saarbrücken, Germany), Alexandra Silva (University College London, UK), Daniel Lokshtanov (University of California at Santa Barbara, USA), Kavitha Telikepalli (Tata Institute of Fundamental Research, India) and Jérôme Leroux (LaBRI, France). We are looking forward to their excellent talks.

Since 2016, the MFCS 2019 proceedings are published in the Dagstuhl/LIPIcs series. We would like to thank Michael Wagner and the LIPIcs team for all their kind help and support. We also like to thank Birgit Willms for her dedicated support in the local organization of MFCS 2019 in Aachen.

Peter Rossmanith
Pigar Heggernes
Joost-Pieter Katoen

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Conference Organization

Program Commitee

Nathalie Bertrand	INRIA
Benedikt Bollig	LSV, ENS Cachan, CNRS
Marthe Bonamy	CNRS, LaBRI, Bordeaux
Flavia Bonomo	Universidad de Buenos Aires – CONICET
Véronique Bruyère	University of Mons
Tiziana Calamoneri	Sapienza University of Rome
Supratik Chakraborty	IIT Bombay
Christophe Crespelle	Université Claude Bernard Lyon 1
Pedro R. D'Argenio	Universidad Nacional de Córdoba – CONICET
Konrad Kazimierz Dabrowski	Durham University
Khaled Elbassioni	Masdar Institute
Edith Elkind	University of Oxford
Leah Epstein	University of Haifa
Henning Fernau	University of Trier
Dana Fisman	Ben-Gurion University
Fedor Fomin	University of Bergen
Serge Gaspers	UNSW Sydney and Data61, CSIRO
Archontia Giannopoulou	TU Berlin
Pinar Heggernes	University of Bergen
Joost-Pieter Katoen	RWTH Aachen University
Eun Jung Kim	CNRS - Paris Dauphine
Ramanujan M. S.	University of Warwick
Radu Mardare	Aalborg University
Arnaud Mary	University of Lyon
Roland Meyer	TU Braunschweig
Martin Milanič	UP IAM and UP FAMNIT, University of Primorska
Neeldhara Misra	Indian Institute of Science
Andrzej Murawski	University of Oxford
Michał Pilipczuk	University of Warsaw
Dieter Rautenbach	University of Ulm
Felix Reidl	University of London
Peter Rossmanith	RWTH Aachen University
Davide Sangiorgi	University of Bologna
Ignasi Sau	CNRS, LIRMM, Montpellier
Lutz Schröder	Friedrich-Alexander-Universität Erlangen-Nürnberg
Hadas Shachnai	Technion
Mahsa Shirmohammadi	CNRS
Pawel Sobocinski	University of Southampton
B Srivathsan	Chennai Mathematical Institute
Ryuhei Uehara	Japan Advanced Institute of Science and Technology
Tarmo Uustalu	Reykjavik University
Franck van Breugel	York University
Erik Jan van Leeuwen	Utrecht University

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Igor Walukiewicz	CNRS, LaBRI
Mingsheng Ying	University of Technology, Sydney
Meirav Zehavi	Ben-Gurion University

External Reviewers

Ábrahám, Erika
Adsul, Bharat
Agrawal, Akanksha
Akshay, S.
Almagor, Shaull
Amarilli, Antoine
Antoniadis, Antonios
Ashok, Pradeesha
Backurs, Arturs
Bamas, Étienne
Bampis, Evripidis
Banik, Aritra
Barmpalias, George
Barrington, David A. Mix
Barto, Libor
Baste, Julien
Belardinelli, Francesco
Bell, Paul
Ben Basat, Ran
Ben-David, Shai
Benedikt, Michael
Bergougnoux, Benjamin
Berwanger, Dietmar
Bhangale, Amey
Blondin, Michael
Blumensath, Achim
Bodlaender, Hans L.
Boiret, Adrien
Bonchi, Filippo
Bonnet, Edouard
Bougeret, Marin
Brazdil, Tomas
Bresolin, Davide
Bressan, Marco
Buchin, Kevin
Bulteau, Laurent
Cadilhac, Michaël
Çağirici, Onur
Cao, Yixin
Capobianco, Silvio
Carlucci, Lorenzo
Carton, Olivier
Carvalho, Marcelo
Catalano, Costanza
Chadha, Rohit
Chakraborty, Shantanav
Chalopin, Jérémie
Chaplick, Steven
Chiarelli, Nina
Chini, Peter
Chistikov, Dmitry
Chitnis, Rajesh
Courcelle, Bruno
Dallard, Clément
Damian, Mirela
Dantchev, Stefan
Day, Adam
Day, Joel
de Haan, Ronald
de Lima, Paloma
de Rezende, Susanna F.
de Wolf, Ronald
Defrain, Oscar
Deligkas, Argyrios
Denkinger, Tobias
Dennunzio, Alberto
Dibek, Cemil
Disser, Yann
Doczkal, Christian
Domaratzki, Mike
Doumane, Amina
Dürr, Christoph
Eftekhari, Mahsa
Ehard, Stefan
Eiben, Eduard
Enright, Jessica
Erlebach, Thomas
Faliszewski, Piotr
Fang, Wang
Fearnley, John
Feghali, Carl
Feldmann, Andreas Emil
Fervari, Raul
Feuerstein, Esteban
Fici, Gabriele
Fieux, Etienne
Fijalkow, Nathanaël
Fortin, Marie
Freer, Cameron
Furber, Robert
Földvári, Attila
Fürst, Maximilian
Gajarský, Jakub
Galesi, Nicola
Gan, Jiarui
Ganian, Robert
Gao, Yihan
Gavazzo, Francesco
Gavinsky, Dmitry
Georgieva, Lilia
Ghilezan, Silvia
Gimbert, Hugo
Golovach, Petr
Gonzalez, Carolina
Gonçalves, Daniel
Goodman-Strauss, Chaim
van Gool, Sam
Gorain, Barun
Graça, Daniel
Gupta, Sid
Gutierrez, Julian
Göbel, Andreas
Gözüpek, Didem
Haase, Christoph
Hadzihasanovic, Amar
Hanh, Michel
Harju, Tero
Hatzel, Meike
He, Meng
Heindel, Tobias
Hellerstein, Lisa
Hermann, Miki
Hershberger, John
Hirvensalo, Mika
Hoang, Duc A.
Hoffmann, Stefan

Holub, Stepan
Horn, Florian
Huang, Shenwei
Hughes, Andrew
Idziak, Pawel
Ikenmeyer, Christian
Iljazović, Zvonko
Jaffke, Lars
Jain, Pallavi
Jones, Mark
Kanté, Mamadou Moustapha
Kari, Jarkko
Katoh, Takashi
Kazakov, Yevgeny
Kelmendi, Edon
Khaniki, Erfan
Kiefer, Stefan
Kissinger, Aleks
Klauck, Hartmut
Klivans, Adam
Knop, Alexander
Kociumaka, Tomasz
Kolay, Sudeshna
Kortsarz, Guy
Koucky, Michal
Kowalik, Lukasz
Krnc, Matjaž
Kulik, Ariel
Kuperberg, Denis
Kwon, O-Joung
Kötzing, Timo
Künnemann, Marvin
Laekhanukit, Bundit
Lampis, Michael
Langetepe, Elmar
Lee, Barton
Levin, Asaf
Li, Qin
Limaye, Nutan
Limouzy, Vincent
Loebl, Martin
Loff, Bruno
Lohrey, Markus
Lubiw, Anna
Löding, Christof
Madathil, Jayakrishnan
Madeira, Alexandre
Magnien, Clemence
Mailler, Cécile
Marino, Andrea
Martin, Barnaby
Marx, Dániel
Matheja, Christoph
Mayr, Peter
Mazowiecki, Filip
McCarty, Rose
Melgratti, Hernan
Melnikov, Alexander
Mendes de Oliveira, Rafael
Mestre, Julian
Michaliszyn, Jakub
Mikhailin, Ivan
Mikulas, Szabolcs
Milius, Stefan
Misra, Pranabendu
Mkrtchyan, Vahan
Mnich, Matthias
Mohr, Elena
Morak, Michael
Moreira, Nelma
Mouatadid, Lalla
Mouawad, Amer
Mudrinski, Nebojša
Mukhopadhyay, Partha
Muskalla, Sebastian
Müller, Moritz
Natarajan Ramamoorthy, Sivaramakrishnan
Nekrashevych, Volodymyr
Neves, Renato
Ng, Kang Feng
Nilsson, Bengt J.
Niskanen, Reino
Nisse, Nicolas
Nitay, Dolav
Nogueira, Loana Tito
Nutov, Zeev
Ochem, Pascal
Ochremiak, Joanna
Ordyniak, Sebastian
Orlandelli, Eugenio
Otachi, Yota
Ozols, Māris
Pankratov, Denis
Paoletti, Nicola
Parreau, Aline
Paschos, Vangelis
Pasechnik, Dmitrii
Paul, Erik
Peleg, David
Penelle, Vincent
Perez, Anthony
Perez, Guillermo
Pieterse, Astrid
Piribauer, Jakob
Pirogov, Anton
Posobin, Gleb
Potapov, Igor
Protti, Fábio
Raffinot, Mathieu
Ramamoorthi, Vijayaragunathan
Ramanujam, R.
Rampersad, Narad
Ramyaa, Ramyaa
Rao B V, Raghavendra
Rao, Michael
Rattan, Gaurav
Ravi, S. S.
Rawitz, Dror
Reitzner, Daniel
Renken, Malte
Richomme, Gwenaël
Rivas, Exequiel
Riveros, Cristian
Roggenbach, Markus
Romashchenko, Andrei
Rot, Jurriaan
Roth, Marc
S, Krishna
S. Thinniyam, Ramanathan
Saar, Guy
Saikawa, Takafumi
Saivasan, Prakash
Sajenko, Andrej
Sakai, Yoshifumi
Sampaio Rocha, Leonardo
Sangnier, Arnaud
Sarma, Jayalal
Sarpatwar, Kanthi
Saurabh, Nitin
Scheideler, Christian
Schewior, Kevin

Schmid, Markus L.
Schmidt, Christiane
Schnoebelen, Philippe
Schoeters, Jason
Schroeder, Matthias
Schwartz, Roy
Schwoon, Stefan
Scquizzato, Michele
Semanišin, Gabriel
Semukhin, Pavel
Shalom, Mordechai
Sharma, Roohani
Shirley, Morgan
Shkatov, Dmitry
Siebertz, Sebastian
Simon, Sunil
Simonov, Kirill
Sinaimeri, Blerina
Sivaraman, Vaidy
Skrzypczak, Michał
Slivovsky, Friedrich
Sokolov, Dmitry
Solomon, Shay
Sonar, Chinmay
Souto, André
Souza, Uéverton
Sprunger, David
Sreejith, A V
Srinivasan, Srikanth
Strehler, Martin
Strozecki, Yann
Subramanyan, Pramod
Suchy, Ondrej
Takaoka, Asahi
Talbot, Jean-Marc
Tamaki, Suguru
Tendera, Lidia
Thaler, Justin
Toruńczyk, Szymon
Torán, Jacobo
Tsukada, Takeshi
Turetsky, Daniel
Uchizawa, Kei
Veltri, Niccolò
Verbitsky, Oleg
Vihrovs, Jevgēnijs
Vijayaraghavan, Aravindan
Vocca, Paola
Voigt, Marco
Volk, Ben Lee
Vyas, Nikhil
Wang, Yishu
Wasa, Kunihiro
Wiederrecht, Sebastian
Worrell, James
Wrochna, Marcin
Xie, Ning
Yuster, Raphael
Zamaraev, Viktor
Zantema, Hans
Zanuttini, Bruno
Zeume, Thomas
Zhuk, Dmitriy
Ziemianski, Krzysztof
Zoros, Dimitris

Trustworthy Graph Algorithms

Mohammad Abdulaziz
TU München, Germany

Kurt Mehlhorn
MPI for Informatics, Saarbrücken, Germany

Tobias Nipkow
TU München, Germany

Abstract

The goal of the LEDA project was to build an easy-to-use and extendable library of correct and efficient data structures, graph algorithms and geometric algorithms. We report on the use of formal program verification to achieve an even higher level of trustworthiness. Specifically, we report on an ongoing and largely finished verification of the blossom-shrinking algorithm for maximum cardinality matching.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Software and its engineering → Formal software verification; Software and its engineering → Software libraries and repositories

Keywords and phrases graph algorithms, formal correct proofs, Isabelle, LEDA, certifying algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.1

Category Invited Talk

1 Introduction

This talk is a follow-up on two previous invited MFCS-talks given by the second author:

- *LEDA: A Library of Efficient Data Types and Algorithms* in MFCS 1989 [31], and
- *From Algorithms to Working Programs: On the Use of Program Checking in LEDA* in MFCS 1998 [33].

After a review of these papers, we discuss the further steps taken to reach even higher trustworthiness of our implementations.

- Formal correctness proofs of checker programs [5, 39], and
- Formal verification of complex graph algorithms [1].

The second item is the technical core of the paper: it reports on the ongoing and largely finished verification of the blossom-shrinking algorithm for maximum cardinality matching in Isabelle/HOL by the first author.

Personal Note by the Second Author. As this paper spans 30 years of work, the reader might get the impression that I followed a plan. This is not the case. As a science, in this case computer science, progresses, there are logical next steps. I took these steps. I did not know 30 years ago, where the journey would lead me.

2 Level One of Trustworthiness: The LEDA Library of Efficient Data Types and Algorithms

In 1989, Stefan Näher and the second author set out to build an easy-to-use and extendable library of correct and efficient data structures, graph algorithms and geometric algorithms. The project was announced in an invited talk at MFCS 1989 [31] and the library is available from Algorithmic Solutions GmbH [27]. LEDA, the library of efficient data types and

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 1; pp. 1:1–1:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

```
template <class NT>
void DIJKSTRA_T(const graph& G, node s, const edge_array<NT>& cost,
                   node_array<NT>& dist, node_array<edge>& pred)
{
  node_pq<NT>  PQ(G);   // a priority queue for the nodes of G
  node v; edge e;
  dist[s] = 0;          // distance from s to s is zero
  PQ.insert(s,0);       // insert s with value 0 into PQ
  forall_nodes(v,G) pred[v] = nil; // no incoming tree edge yet
  while (!PQ.empty())   // as long as PQ is non-empty
  { node u = PQ.del_min();   // let u be the node with minimum dist in PQ
    NT du = dist[u];         // and du its distance
    forall_adj_edges(e,u)    // iterate over all edges e out of u
    { v = G.opposite(u,e);   // makes it work for ugraphs
      NT c = du + cost[e];   // distance to v via u
      if (pred[v] == nil && v != s )            // v already reached?
        PQ.insert(v,c);                         // first path to v
      else if (c < dist[v]) PQ.decrease_p(v,c); // better path
           else continue;
      dist[v] = c;           // store distance value
      pred[v] = e;           // and incoming tree edge
    }
   }
  }
```

Figure 1 The LEDA implementation of Dijkstra's algorithm: Note that the executable code above is similar to a typical pseudo-code presentation of the algorithm.

algorithms, offers a flexible data type graph with loops for iterating over edges and nodes and arrays indexed by nodes and edges. It also offers the data types required for graph algorithms such as queues, stacks, and priority queues. It thus created a framework in which graph algorithms can be formulated easily and naturally, see Figure 1 for an example. The design goal was to create a system in which the difference between the pseudo-code used to explain an algorithm and what constitutes an executable program is as small as possible. The expectation was that this would ease the burden of the implementer and make it easier to get implementations correct.

3 Level Two of Trustworthiness: Certifying Algorithms

Nevertheless, some implementations in the initial releases were incorrect, in particular, the planarity test[1]; it declared some planar graphs non-planar. At around 1995, we adopted the concept of certifying algorithms [33, 30] for the library and reimplemented all algorithms [34]. A certifying algorithm computes for each input a easy-to-check certificate (witness) that demonstrates to the user that the output of the program for this particular input is correct; see Figure 2. For example, the certifying planarity test returns a Kuratowski subgraph if it

[1] Most of the implementations of the geometric algorithms were also incorrect in their first release as we had naïvely used floating point arithmetic to implement real arithmetic and the rounding errors invalidated the implementations of the geometric primitives. This lead to the development of the exact computation paradigm for geometric computing by us and others [20, 45, 14, 44, 32]. In this paper, we restrict to graph algorithms.

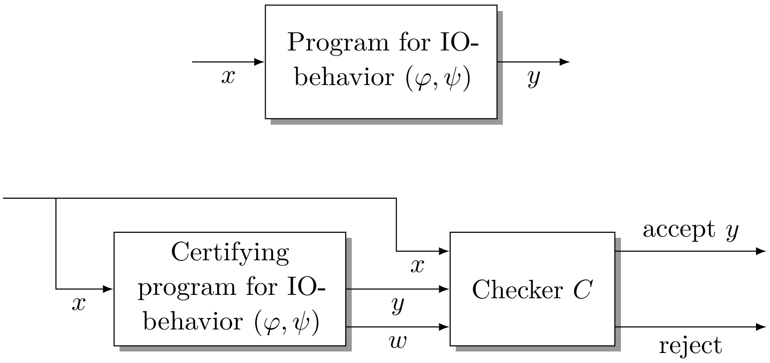

x
Program for IO-behavior (φ, ψ)
y
x
Certifying program for IO-behavior (φ, ψ)
x
y
w
Checker C
accept y
reject

Figure 2 The top figure shows the I/O behavior of a conventional program for IO-behavior (φ, ψ); here φ is the precondition and ψ is the postcondition. The user feeds an input x satisfying $\varphi(x)$ to the program and the program returns an output y satisfying $\psi(x, y)$. A certifying algorithm for IO-behavior (φ, ψ) computes y and a witness w. The checker C accepts the triple (x, y, w) if and only if w is a valid witness for the postcondition $\psi(x, y)$, i.e., it proves $\psi(x, y)$. (reprinted from [5])

declares the input graph non-planar and a (combinatorial) planar embedding if it declares the input graph planar, and the maximum cardinality matching algorithm computes a matching and an odd-set-cover that proves its optimality; see Figures 3 and 4. The state of the art of certifying algorithms is described in [30]. We also implemented checker programs that check the witness for correctness and argued that the checker programs are so simple that their correctness is evident. From a pragmatic point of view, the goals of the project were reached by 2010. The library was easy-to-use and extendable, the implementations were efficient, and no error was discovered in any of the graph algorithms for several years despite intensive use by a commercial and academic user community.

Note that, most likely, errors would not have gone undiscovered because of the use of certifying algorithms and checker programs. Only if a module produced an incorrect output and hence an invalid certificate and the checker program missed to uncover the invalidity of the certificate would an error go unnoticed. Of course, the possibility is there and the phrase “most likely” in the preceding sentence has no mathematical meaning.

Alternative libraries such as Boost and LEMON [43, 28] are available now and some of their implementations are slightly more efficient than ours. However, none of the new libraries pays the same attention to correctness. For example, all libraries allow floating point numbers as weights and capacities in network algorithms, but only LEDA ensures that the intricacies of floating point arithmetic do not invalidate the implementations; see [6] and [34, Section 7.2].

4 Level Three of Trustworthiness: Formal Verification of Checkers

We stated above that the checker programs are so simple that their correctness is evident. Shouldn’t they then be amenable to formal verification? Harald Ganzinger and the second author attempted to do so at around 2000 and failed. About 10 years later (2011 – 2014) Eyad Alkassar from the Verisoft Project [42], Sascha Böhme and Lars Noschinski from Tobias Nipkow’s group at TU München, and Christine Rizkallah and the second author succeeded in formally verifying some of the checker programs [5, 39]. In order to be able to talk about formal verification of checker programs, we need to take a more formal look at certifying algorithms.

A *matching* in a graph G is a subset M of the edges of G such that no two share an endpoint.

An odd-set cover OSC of G is a labeling of the nodes of G with non-negative integers such that every edge of G (which is not a self-loop) is either incident to a node labeled 1 or connects two nodes labeled with the same i, $i \geq 2$.

Let n_i be the number of nodes labeled i and consider any matching N. For i, $i \geq 2$, let N_i be the edges in N that connect two nodes labeled i. Let N_1 be the remaining edges in N. Then $|N_i| \leq \lfloor n_i/2 \rfloor$ and $|N_1| \leq n_1$ and hence

$$|N| \leq n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$$

for any matching N and any odd-set cover OSC. It can be shown that for a maximum cardinality matching M there is always an odd-set cover OSC with

$$|M| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor,$$

thus proving the optimality of M. In such a cover all n_i with $i \geq 2$ are odd, hence the name.

list<edge> MAX_CARD_MATCHING(*graph G*, *node_array<int>& OSC*)

computes a maximum cardinality matching M in G and returns it as a list of edges.
The algorithm ([12], [15]) has running time $O(nm \cdot \alpha(n,m))$.
An odd-set cover that proves the maximality of M is returned in OSC.

bool CHECK_MAX_CARD_MATCHING(*graph G*, *list<edge> M*, *node_array<int> OSC*)

checks whether M is a maximum cardinality matching in G and OSC is a proof of optimality. Aborts if this is not the case.

Figure 3 The LEDA manual page for maximum cardinality matchings (reprinted from [33]).

We consider algorithms which take an input from a set X and produce an output in a set Y and a witness in a set W. The input $x \in X$ is supposed to satisfy a precondition $\varphi(x)$, and the input together with the output $y \in Y$ is supposed to satisfy a postcondition $\psi(x,y)$. A *witness predicate* for a specification with precondition φ and postcondition ψ is a predicate $\mathcal{W} \subseteq X \times Y \times W$, where W is a set of witnesses with the following *witness property*:

$$\varphi(x) \wedge \mathcal{W}(x,y,w) \longrightarrow \psi(x,y). \tag{1}$$

The checker program C receives a triple[2] (x,y,w) and is supposed to check whether it fulfills the witness property. If $\neg\varphi(x)$, C may do anything (run forever or halt with an arbitrary output). If $\varphi(x)$, C must halt and either accept or reject. It is required to accept if $\mathcal{W}(x,y,w)$ holds and is required to reject otherwise. This results in the following proof obligations.

Checker Correctness: We need to prove that C checks the witness predicate assuming that the precondition holds, i.e., on input (x,y,w):

(i) If $\varphi(x)$, C halts.

(ii) If $\varphi(x)$ and $\mathcal{W}(x,y,w)$, C accepts (x,y,w), and if $\varphi(x)$ and $\neg\mathcal{W}(x,y,w)$, C rejects the triple.

Witness Property: We need to prove implication (1).

[2] We ignore the minor complication that X, Y, and W are abstract sets and programs handle concrete representations.

```
static bool return_false(string s)
{ cerr << "CHECK_MAX_CARD_MATCHING: " << s << "\n"; return false; }

bool CHECK_MAX_CARD_MATCHING(const graph& G, const list<edge>& M,
                                    const node_array<int>& OSC)
{ int n = Max(2,G.number_of_nodes());
  int K = 1;
  array<int> count(n);
  for (int i = 0; i < n; i++) count[i] = 0;
  node v; edge e;

  forall_nodes(v,G)
  { if ( OSC[v] < 0 || OSC[v] >= n )
     return_false("negative label or label larger than n - 1");
    count[OSC[v]]++;
    if (OSC[v] > K) K = OSC[v];
  }

  int S = count[1];
  for (int i = 2; i <= K; i++) S += count[i]/2;
  if ( S != M.length() )
    return_false("OSC does not prove optimality");

  forall_edges(e,G)
  { node v = G.source(e); node w = G.target(e);
    if ( v == w || OSC[v] == 1 || OSC[w] == 1 ||
            ( OSC[v] == OSC[w] && OSC[v] >= 2) ) continue;
    return_false("OSC is not a cover");
  }
  return true;
}
```

Figure 4 The checker for maximum cardinality matchings (reprinted from [33]).

In case of the maximum cardinality matching problem, the witness property states that an odd-set cover OSC as defined in Figure 3 with $|M| = n_1 + \sum_{i\geq 2}\lfloor n_i/2 \rfloor$ proves that the matching M has maximum cardinality. Checker correctness amounts to the statement that the program shown in Figure 4 is correct.

We proved the witness property using Isabelle/HOL [37]; see Section 5.1 for more information on Isabelle/HOL. For the checker correctness, we used VCC [9] and later Simpl [41] and AutoCorres [16]. The latter approach has the advantage that the entire verification can be performed within Isabelle. Simpl is a generic imperative programming language embedded into Isabelle/HOL, which was designed as an intermediate language for program verification. We implemented checkers both in Simpl and C. Checkers written in Simpl were verified directly within Isabelle. For the checkers written in C, we first translated from C to Isabelle using the C-to-Isabelle parser that was developed as part of the seL4 project [21], and then used the AutoCorres tool developed at NICTA that simplifies reasoning about C in Isabelle/HOL. Christine spent several months at NICTA to learn how to use the tool. We verified the checkers for connectivity, maximum cardinality matching, and non-planarity. In particular, for the non-planarity checker it was essential that Lars Noschinski in parallel formalized basic graph theory in Isabelle [38].

A disclaimer is in order here. We did not verify the C++ program shown in Figure 4. Rather we verified a manual translation of this program into Simple or C, respectively. For this translation, we assumed a very basic representation of graphs. The nodes are numbered from 0 to $n-1$, the edges are numbered from 0 to $m-1$ with the edges incident to any vertex numbered consecutively and arrays of the appropriate dimension are used for cross-referencing and for encoding adjacency lists.

The verification attempt for the maximum cardinality checker shown in Figure 4 discovered a flaw. Note that the program does not check whether the edges in M actually belong to G. When we wrote the checker, we apparently took this for granted. The verification attempt revealed the flaw.

We also considered going further and briefly tried to verify the LEDA maximum cardinality matching algorithm [34, Section 7.7]. The program has 330 lines of code and the description of the algorithm, its implementation and its correctness proof spans over 20 pages. We found the task too daunting and, extrapolating from the effort required for the verification of the checkers, estimated the effort as several man-years.

5 Level Four of Trustworthiness: Formal Verification of Complex Algorithms

A decade later, we perform the formal verification of the blossom-shrinking algorithm for maximum cardinality. We give a short account of the verification which will be described in detail in our forthcoming publication [1]. On a high-level Edmond's blossom-shrinking algorithm [12] works as follows. The algorithm repeatedly searches for an augmenting path with respect to the current matching. Initially, the current matching is empty. Whenever an augmenting path is found, augmentation of the path increases the size of the matching by one. If no augmenting path exists with respect to the current matching, the current matching has maximum cardinality.

The search for an augmenting path is via growing alternating trees rooted at free vertices, i.e. vertices not incident to an edge of the matching. The search is initialised by making each free vertex a root of an alternating tree; the matched nodes are in no tree initially. In an alternating tree, vertices at even depth are entered by a matching edge, vertices at odd depth are entered by a non-matching edge, and all leaves have even depth. In each step of the growth process, one considers a vertex, say u_1, of even depth that is incident to an edge $\{u_1, u_2\}$ not considered before. If u_2 is not in a tree yet, then one adds u_2 (at odd level) and its mate (at even level) under the current matching to the tree. If u_2 is already in a tree and has odd level then one does nothing as one simply has discovered another odd length path to u_2. If u_2 is already in a tree and has even level then one has either discovered an augmenting path (if u_2 is in a different tree than u_1) or a blossom (if u_2 and u_1 are in the same tree). In the latter case, consider the tree paths from u_2 and u_1 back to their common root and let u_3 be the lowest common ancestor of u_2 and u_1. The edge $\{u_1, u_2\}$ plus the tree paths from u_1 and u_2 to u_3 form an odd length cycle. One collapses all nodes on the cycle into a single node and repeats the search for an augmenting path in the quotient (= shrunken) graph. If an augmenting path is found in the quotient graph, it is lifted (refined) to an augmenting path in the original graph. If no augmenting path exists in the quotient graph, no augmenting path exists in the original graph. In this section, we describe in detail the algorithm outlined above, and the process of formalising and verifying it in Isabelle/HOL.

5.1 Isabelle/HOL

Isabelle/HOL [40] is a theorem prover for classical Higher-Order Logic. Roughly speaking, Higher-Order Logic can be seen as a combination of functional programming with logic. Isabelle's syntax is a variation of Standard ML combined with (almost) standard mathematical notation. Application of function f to arguments $x_1 \ \dots \ x_n$ is written as $f \ x_1 \ \dots \ x_n$ instead of the standard notation $f(x_1, \ \dots \ , x_n)$. We explain non-standard syntax in the paper where it occurs.

Isabelle is designed for trustworthiness: following the LCF approach [35], a small kernel implements the inference rules of the logic, and, using encapsulation features of ML, it guarantees that all theorems are actually proved by this small kernel. Around the kernel, there is a large set of tools that implement proof tactics and high-level concepts like algebraic data types and recursive functions. Bugs in these tools cannot lead to inconsistent theorems being proved since they all rely on the kernel only, but only to error messages when the kernel refuses a proof. Isabelle/HOL comes with a rich set of already formalized theories, among which are natural numbers and integers as well as sets and finite sets.

5.2 Preliminaries

An edge is a set of vertices with size 2. A graph $\mathcal{G}$ is a set of edges. A set of edges $\mathcal{M}$ is a matching iff $\forall e, e' \in \mathcal{M}.\ e \cap e' = \emptyset$. In Isabelle/HOL that is represented as follows:

```
matching M ⟷ (∀e1 ∈ M. ∀e2 ∈ M. e1 ≠ e2 ⟶ e1 ∩ e2 = {})
```

In may cases, a matching is a subset of a graph, in which case we call it a matching w.r.t. the graph. For a graph $\mathcal{G}$, $\mathcal{M}$ is a maximum matching w.r.t. $\mathcal{G}$ iff for any matching $\mathcal{M}'$ w.r.t. $\mathcal{G}$, we have that $|\mathcal{M}'| \leq |\mathcal{M}|$.

5.3 Formalising Berge's Lemma

A list of vertices $u_1 u_2 \dots u_n$ is a path w.r.t. a graph $\mathcal{G}$ iff every $\{u_i, u_{i+1}\} \in \mathcal{G}$. A path $u_1 u_2 \dots u_n$ is a simple path iff for every $1 \leq i \neq j \leq n$, $u_i \neq u_j$. A list of vertices $u_1 u_2 \dots u_n$ is an alternating path w.r.t. a set of edges E iff for some E' (i) $E' = E$ or $E' = \{e \mid e \notin E\}$, (ii) $\{u_i, u_{i+1}\} \in E'$ holds for all even numbers i, where $1 \leq i < n$, and (iii) $\{u_i, u_{i+1}\} \notin E'$ holds for all odd numbers i, where $1 \leq i \leq n$. We call a list of vertices $u_1 u_2 \dots u_n$ an augmenting path w.r.t. a matching $\mathcal{M}$ iff $u_1 u_2 \dots u_n$ is an alternating path w.r.t. $\mathcal{M}$ and $u_1, u_n \notin \bigcup \mathcal{M}$. It is often the case that an augmenting path γ w.r.t. to a matching $\mathcal{M}$ is also a simple path w.r.t. a graph $\mathcal{G}$, in which case we call the path an augmenting path w.r.t. to the pair $\langle \mathcal{G}, \mathcal{M} \rangle$. Also, for two sets s and t, $s \oplus t$ denotes the symmetric difference of the two sets. We overload $\oplus$ to arguments which are lists in the obvious fashion.

▶ **Theorem 1** (Berge's Lemma). *For a graph $\mathcal{G}$, a matching $\mathcal{M}$ is maximum w.r.t. $\mathcal{G}$ iff there is not an augmenting path γ w.r.t. $\langle \mathcal{G}, \mathcal{M} \rangle$.*

Our proof of Berge's lemma is shorter than the standard proof. The standard proof consists of three steps. First, for any two matchings $\mathcal{M}$ and $\mathcal{M}'$, every connected component of the graph $\mathcal{M} \oplus \mathcal{M}'$ is either (i) a singleton vertex, (ii) a path, or (iii) a cycle. Second, for a set of edges $C \subseteq \mathcal{M} \oplus \mathcal{M}'$ s.t. $|C \cap \mathcal{M}| < |C \cap \mathcal{M}'|$, the edges from C form a path. Third, such a set C of edges exists, if $|\mathcal{M}| < |\mathcal{M}'|$. We observe that it is easier to directly show that such a C exists and that all its edges can be arranged in a path, without having to prove the first step about all connected components. We found this different proof during the process of formalising the theorem, and finding this shorter proof was primarily motivated by making the formalisation shorter and more feasible. The discovery of simpler proofs or more general theorem statements is one potential positive outcome of verifying algorithms, and mathematics in general, in interactive theorem provers [3, 2, 10].

Algorithm 1 FIND_MAX_MATCHING($\mathcal{G}, \mathcal{M}$).

```
$\gamma :=$ AUG_PATH_SEARCH($\mathcal{G}, \mathcal{M}$)
if $\gamma$ is some augmenting path
  return FIND_MAX_MATCHING($\mathcal{G}, \mathcal{M} \oplus \gamma$)
else
  return $\mathcal{M}$
```

Now consider Algorithm 1. Berge's lemma implies the validity of that algorithm as a method to compute maximum matchings in graphs. The validity of Algorithm 1 is stated in the following corollary.

▶ **Corollary 1.** *Assume that* AUG_PATH_SEARCH($\mathcal{G}, \mathcal{M}$) *is an augmenting path w.r.t.* $\langle \mathcal{G}, \mathcal{M} \rangle$*, for any graph* $\mathcal{G}$ *and matching* $\mathcal{M}$*, iff* $\mathcal{G}$ *has an augmenting path w.r.t.* $\langle \mathcal{G}, \mathcal{M} \rangle$*. Then, for any graph* $\mathcal{G}$*,* FIND_MAX_MATCHING($\mathcal{G}, \emptyset$) *is a maximum matching w.r.t.* $\mathcal{G}$*.*

As shown in Corollary 1, Algorithm 1 depends on the function AUG_PATH_SEARCH which is a sound and a complete procedure to compute augmenting paths in graphs.

In Isabelle/HOL, the first step is to formalise the path concepts from above. Paths and alternating paths are defined recursively in a straightforward fashion. An augmenting path is defined as follows:

```
augmenting_path M p ≡ (length p ≥ 2) ∧ alt_path M p
                       ∧ hd p ∉ Vs M ∧ last p ∉ Vs M
```

The formalised statement of Berge's lemma is as follows:

```
theorem Berge:
  assumes
    finite M and matching M and M ⊆ E
    and
    (∀e∈E. ∃u v. e = {u,v} ∧ u ≠ v) and finite (Vs E)
  shows (∃p. augmenting_path M p ∧ path E p ∧ distinct p) ⟷
          (∃M' ⊆ E. matching M' ∧ card M < card M')
```

Note that in the formalisation when the paths need to be simple, such as in Berge's lemma above, we have the additional assumption that all vertices are pairwise distinct, denoted by the Isabelle/HOL predicate *distinct*. Just to clarify Isabelle's syntax: the lemma above has two sets of assumptions, one on the matching and the other on the graph. The matching has to be a finite set, which is a matching w.r.t. the given graph. The graph has to have edges which only have two vertices, and its set of vertices has to be finite.

In Isabelle/HOL Algorithm 1 is formalised within the following *locale*.

```
locale find_max_match =
  fixes aug_path_search::'a set set ⇒ 'a set set ⇒ ('a list) option and
    E
  assumes
    aug_path_search_complete:
    matching M ∧ M ⊆ E ∧ finite M ∧
      (∃p. path E p ∧ distinct p ∧ augmenting_path M p)
```

```
         ⟹ (∃p. aug_path_search E M = Some p)
    and
    aug_path_search_sound:
    matching M ∧ M ⊆ E ∧ finite M ∧ aug_path_search E M = Some p ⟹
             path E p ∧ distinct p ∧ augmenting_path M p
    and
    graph: ∀e∈E. ∃u v. e = {u, v} ∧ u ≠ v finite (Vs E)
```

A locale is a named context: definitions and theorems proved within locale `find_max_match` can refer to the parameters and assumptions declared there. In this case, we need the locale to identify the parameter `aug_path_search` of the locale, corresponding to the function Aug_Path_Search, which is used in Algorithm 1. The function `aug_path_search` should take as input a graph and a matching. It should return an `('a list) option` typed value. Generally speaking, the value of an `'a option` valued term could be in one of two forms: either `Some x`, or `None`, where `x` is of type `'a`. In the case of `aug_path_search`, it should return either `Some p`, where `p` is a path in case an augmenting path is found, or `None`, otherwise. There is also the function `the`, which given a term of type `'a option`, returns `x`, if the given term is `Some x`, and which is undefined otherwise. Within that locale, the definition of Algorithm 1 and its verification theorem are as follows. Note that the verification theorem has four conclusions: the algorithm returns a subset of the graph, that subset is a matching, that matching is finite and the cardinality of any other matching is bounded by the size of the returned matching.

```
  find_max_matching M =
     (if (∃p. aug_path_search E M = Some p) then
        (find_max_matching (M ⊕ (set (edges_of_path (the (aug_path_search E M))))))
      else M)

lemma find_max_matching_works:
  shows (find_max_matching {}) ⊆ E
    matching (find_max_matching {})
    finite (find_max_matching {})
    ∀M. matching M ∧ M ⊆ E ∧ finite M ⟶ card M ≤ card (find_max_matching {})
```

Functions defined within a locale are parameterised on the constants which are declared in the locale's definition. When a function is used outside a locale, these parameters must be specified. So, if `find_max_matching` is used outside the locale above, it should take a function which computes augmenting paths as a parameter. Similarly, theorems proven within a locale implicitly have the assumptions of the locale. So if we use the lemma `find_max_matching_works`, we would have to prove that the functional argument to `find_max_matching` satisfies the assumptions of the locale, i.e. that argument is a sound and complete procedure for computing augmenting paths. The way theorems from locales are used will be clearer in the next section when we refer to the function `find_max_matching` and use the lemma `find_max_matching_works` outside of the locale `find_max_match`. The use of locales for performing gradual refinement of algorithms allows to focus on the specific aspects of the algorithm relevant to a refinement stage, with the rest of the algorithm abstracted away.

5.4 Verifying that Blossom Contraction Works

In Corollary 1, which specifies the soundness of Find_Max_Matching, we have not explicitly specified the function Aug_Path_Search. Indeed, we have only specified what its output has to conform to. We now refine that specification and describe Aug_Path_Search algorithmically.

Firstly, for a function f and a set s, let $f(\!|s|\!)$ denote the image of f on s. Also, for a set of edges E, and a function f, the quotient E/f is the set $\{f(\!|e|\!) \mid e \in E\}$. We now introduce the concepts of a *blossom*. A list of vertices $u_1 u_2 \dots u_n$ is called a cycle if $3 < n$ and $u_n = u_1$, and we call it an odd cycle if n is even. A pair $\langle u_1 u_2 \dots u_{i-1}, u_i u_{i+1} \dots u_n \rangle$ is a blossom w.r.t. a matching $\mathcal{M}$ iff (i) $u_i u_{i+1} \dots u_n$ is an odd cycle, (ii) $u_1 u_2 \dots u_n$ is an alternating path w.r.t. $\mathcal{M}$, and (iii) $u_1 \notin \bigcup \mathcal{M}$. We also refer to $u_1 u_2 \dots u_i$ as the stem of the blossom. In many situations we have a pair $\langle u_1 u_2 \dots u_{i-1}, u_i u_{i+1} \dots u_n \rangle$ which is a blossom w.r.t. a matching $\mathcal{M}$ where $u_1 u_2 \dots u_{i-1} u_i u_{i+1} \dots u_{n-1}$ is also a simple path w.r.t. a graph $\mathcal{G}$ and $\{u_{n-1}, u_n\} \in \mathcal{G}$. In this case we call it a blossom w.r.t. $\langle \mathcal{G}, \mathcal{M} \rangle$.

Based on the above definitions, we prove that contracting (i.e. shrinking) the odd cycle of a blossom preserves the existence of an augmenting path, which is the second main result needed to prove the validity of the blossom-shrinking algorithm, after Berge's lemma.

▶ **Theorem 2.** *Consider a graph $\mathcal{G}$ and a vertex $u \notin \bigcup \mathcal{G}$. Let for a set s, the function P_s be defined as $P_s(x) = \mathsf{if}\ x \in s\ \mathsf{then}\ u\ \mathsf{else}\ x$. Then, for a blossom $\langle \gamma, C \rangle$ w.r.t. $\langle \mathcal{G}, \mathcal{M} \rangle$, if s is the set of vertices in C, then we have an augmenting path w.r.t. $\langle \mathcal{G}, \mathcal{M} \rangle$ iff there is an augmenting path w.r.t. $\langle \mathcal{G}/P_s, \mathcal{M}/P_s \rangle$.*

Theorem 2 is used in most expositions of the blossom-shrinking algorithm. In our proof for the forward direction (if an augmenting path exists w.r.t. $\langle \mathcal{G}, \mathcal{M} \rangle$, then there is an augmenting path w.r.t. $\langle \mathcal{G}/P_s, \mathcal{M}/P_s \rangle$, i.e. w.r.t. the quotients), we follow a standard textbook approach [22]. In our proof for the backward direction (an augmenting path w.r.t. the quotients can be lifted to an augmenting path w.r.t. the original graph) we define an (almost) executable function refine that does the lifting.[3] We took the choice of explicitly defining that function with using it in the final algorithm in mind. This is similar to the approach used in the informal proof of soundness of the variant of the blossom-shrinking algorithm used in LEDA [34].

Now, using Theorem 2, one can show that Algorithm 2 is a sound and complete procedure for computing augmenting paths.

Algorithm 2 AUG_PATH_SEARCH($\mathcal{G}, \mathcal{M}$).

```
if COMPUTE_BLOSSOM($\mathcal{G}, \mathcal{M}$) is a blossom $\langle \gamma, C \rangle$ w.r.t. $\langle \mathcal{G}, \mathcal{M} \rangle$
  return refine(AUG_PATH_SEARCH($\mathcal{G}/P_C, \mathcal{M}/P_C$))
else if COMPUTE_BLOSSOM($\mathcal{G}, \mathcal{M}$) is an augmenting path w.r.t. $\langle \mathcal{G}, \mathcal{M} \rangle$
  return COMPUTE_BLOSSOM($\mathcal{G}, \mathcal{M}$)
else
  return no augmenting path found
```

The soundness and completeness of this algorithm assumes that COMPUTE_BLOSSOM can successfully compute a blossom or an augmenting path in a graph iff either one exists. This is formally stated as follows.

▶ **Corollary 2.** *Assume that, for a graph $\mathcal{G}$ and a matching $\mathcal{M}$ w.r.t. $\mathcal{G}$, there is a blossom or an augmenting path w.r.t. $\langle \mathcal{G}, \mathcal{M} \rangle$ iff* COMPUTE_BLOSSOM($\mathcal{G}, \mathcal{M}$) *is a blossom or an augmenting path w.r.t. $\langle \mathcal{G}, \mathcal{M} \rangle$. Then for any graph $\mathcal{G}$ and matching $\mathcal{M}$,* AUG_PATH_SEARCH($\mathcal{G}, \mathcal{M}$) *is an augmenting path w.r.t. $\langle \mathcal{G}, \mathcal{M} \rangle$ iff there is an augmenting path w.r.t. $\langle \mathcal{G}, \mathcal{M} \rangle$.*

[3] The function refine, as defined later, is executable except for a choice operation.

To formalise that in Isabelle/HOL, an odd cycle and a blossom are defined as follows:

```
odd_cycle p ≡ (length p ≥ 3) ∧ odd (length (edges_of_path p)) ∧ hd p = last p

blossom M stem C ≡ alt_path M (stem @ C) ∧
  distinct (stem @ (butlast C)) ∧ odd_cycle C ∧ hd (stem @ C) ∉ Vs M ∧
  even (length (edges_of_path (stem @ [hd C])))
```

In the above definition `@` stands for list concatenation and `edges_of_path` is a function which, given a path, returns the list of edges constituting the path.

To define the function refine that refines a quotient augmenting path to a concrete one or to formalise the theorems showing that contracting blossoms preserves augmenting paths we first declare the following locale:

```
locale quot =
  fixes P s u
  assumes ∀v∈s. P v = v and u∉s and (∀v. v∉s ⟶ P v = u)
```

That locale fixes a function `P`, a set of vertices `s` and a vertex `u`. The function `P` maps all vertices from `s` to the given vertex `u`.

Now, we formalise the function refine which lifts an augmenting path in a quotient graph to an augmenting path in the concrete graph. The function refine takes an augmenting path p in the quotient graph and returns it unchanged if it does not contain the vertex u and deletes u and splits p into two paths p_1 and p_2 otherwise. In the latter case, p_1 and p_2 are passed to `replace_cycle`. This function first defines two auxiliary paths `stem2p2` and `p12stem` using the function `stem2vert_path`. Let us have a closer look at the path `stem2p2`. `stem2vert_path` with last argument `hd p2` uses `choose_con_vert` to find a neighbor of `hd p2` on the cycle C. It splits the cycle at this neighbor and then returns the path leading to the base of the blossom starting with a matching edge. Finally, `replace_cycle` glues together p_1, p_2 and either `stem2p2` and `p12stem` to obtain an augmenting path in the concrete graph.

```
choose_con_vert vs E v ≡ (SOME v'. v' ∈ vs ∧ {v, v'} ∈ E)

stem2vert_path C E M v ≡
let find_pfx' = (λC. find_pfx ((=) (choose_con_vert (set C) E v)) C) in
  if (last (edges_of_path (find_pfx' C)) ∈ M) then
    (find_pfx' C)
  else
    (find_pfx' (rev C))

replace_cycle C E M p1 p2 ≡
 let stem2p2 = stem2vert_path C E M (hd p2);
     p12stem = stem2vert_path C E M (last p1) in
 if p1 = [] then
   stem2p2 @ p2
 else
   (if p2 = [] then
      p12stem @ (rev p1)
    else
     (if {u, hd p2} ∉ quotG M then
        p1 @ stem2p2 @ p2
      else
        (rev p2) @ p12stem @ (rev p1)))
```

```
refine C E M p ≡
 if (u ∈ set p) then
   (replace_cycle C E M (fst (pref_suf [] u p)) (snd (pref_suf [] u p)))
 else p
```

In Isabelle/HOL the two directions of the equivalence in Theorem 2 are formalised as follows:

```
theorem quot_apath_to_apath:
  assumes
    odd_cycle C and alt_path M C and distinct (tl C) and path E C
    and
    augmenting_path (quotG M) p' and distinct p' and path (quotG E) p'
    and
    matching M and M ⊆ E
    and
    s = (Vs E) - set C
    and
    ∀e∈E. ∃u v. e = {u, v} ∧ u ≠ v
  shows augmenting_path M (refine C E M p') ∧ path E (refine C E M p') ∧
          distinct (refine C E M p')

theorem aug_path_works_in_contraction:
  assumes
    path E (stem @ C) and blossom M stem C
    and
    augmenting_path M p and path E p and distinct p
    and
    matching M and M ⊆ E and finite M
    and
    s = (Vs E) - set C and u ∉  Vs E
    and
    ∀e∈E. ∃u v. e = {u, v} ∧ u ≠ v and finite (Vs E)
  shows ∃p'. augmenting_path (quotG M) p' ∧ path (quotG E) p' ∧ distinct p'
```

A main challenge with formalising Theorem 2 in Isabelle/HOL is the lack of automation for handling symmetries in its proof.

To formalise Algorithm 2 we use a locale to assume the existence of the function which computes augmenting paths or blossoms, iff either one exist. That function is called `blos_search` in the locale declaration. Its return type and the assumptions on it are as follows:

```
datatype 'a blossom_res =
    Path (aug_path: "'a list") | Blossom (stem_vs: "'a list") (cycle_vs: "'a list")

bloss_algo_complete:
  (((∃p. path E p ∧ distinct p ∧ augmenting_path M p)
        ∨ (matching M ∧ (∃stem C. path E (stem @ C) ∧ blossom M stem C))))
     ⟹ (∃blos_comp. blos_search E M = Some blos_comp)

bloss_algo_sound:
  (∀e∈E. ∃u v. e = {u, v} ∧ u ≠ v) ∧ blos_search E M = Some (Path p)
      ⟹ (path E p ∧ distinct p ∧ augmenting_path M p)
  blos_search E M = Some (Blossom stem C)
      ⟹ (path E (stem @ C) ∧ (matching M ⟶ blossom M stem C))
```

The locale also fixes a function *create_vert* which creates new vertex names to which vertices from the odd cycle are mapped during contraction. Within that locale, we define Algorithm 2 and prove its soundness and completeness theorems, which are as follows:

```
quotG E ≡ (quot_graph P E) - {{u}}

find_aug_path E M =
   (case blos_search E M of Some blossom_res ⇒
      case blossom_res of Path p ⇒ Some p
      | Blossom stem cyc ⇒
          let u = create_vert (Vs E);
              s = Vs E - (set cyc);
              quotG = quot.quotG (quot_fun s u) u;
              refine = quot.refine (quot_fun s u) u cyc E M
          in (case find_aug_path (quotG E) (quotG M) of Some p' ⇒ Some (refine p')
              | _ ⇒ None)
     | _ ⇒ None)
```

```
lemma find_aug_path_sound:
  assumes
    matching M and M ⊆ E and finite M
    and
    ∀e∈E. ∃u v. e = {u, v} ∧ u ≠ v and finite (Vs E)
    and
    find_aug_path E M = Some p
  shows augmenting_path M p ∧ path E p ∧ distinct p
```

```
lemma find_aug_path_complete:
  assumes
    augmenting_path M p and path E p and distinct p
    and
    matching M and M ⊆ E and finite M
    and
    ∀e∈E. ∃u v. e = {u, v} ∧ u ≠ v and finite (Vs E)"
  shows ∃p'. find_aug_path E M = Some p'
```

Note that in *find_aug_path*, we instantiate both arguments *P* an *s* of the locale *quot* to obtain the quotienting function *quotG* and the function for refining augmenting path *refine*.

Lastly, what follows shows the validity of instantiating the functional argument of *find_max_matching* with *find_aug_path*, which gives us the following soundness theorem of the resulting algorithm.

```
lemma find_max_matching_works:
  assumes
    finite (Vs E) and ∀e∈E. ∃u v. e = {u, v} ∧ u ≠ v
  shows
    find_max_match.find_max_matching find_aug_path E {} ⊆ E
    matching (find_max_match.find_max_matching find_aug_path E {})
    finite (find_max_match.find_max_matching find_aug_path E {})
    ∀M. matching M ∧ M ⊆ E ∧ finite M
        ⟶ card M ≤ card (find_max_match.find_max_matching find_aug_path E {})
```

5.5 Computing Blossoms and Augmenting Paths

Until now, we have only assumed the existence of the function COMPUTE_BLOSSOM, which can compute augmenting paths or blossoms, if any exist in the graph. We now refine that to an algorithm which, given two alternating paths resulting from the ascent of alternating trees, returns either an augmenting path or a blossom.

We first introduce some notions and notation. For a list l, let $|l|$ be the length of l. For a list l and a natural number n, let $\mathsf{drop}\ n\ l$ denote the list l, but with the first n elements dropped. For a list l, let $h :: l$ denote adding an element h to the front of a list l. For a non-empty list l, let $\mathsf{first}\ l$ and $\mathsf{last}\ l$ denote the first and last elements of l, respectively. Also, for a list l, let $\mathsf{rev}\ l$ denote its reverse. For two lists l_1 and l_2, let $l_1 \frown l_2$ denote their concatenation. Also, let $\mathsf{longest\_disj\_pref}\ l_1\ l_2$ denote the pair of lists $\langle l'_1, l'_2\rangle$, where l'_1 and l'_2 are the longest disjoint prefixes of $l1$ and l_2, respectively, s.t. $\mathsf{last}\ l'_1 = \mathsf{last}\ l'_2$. Note: $\mathsf{longest\_disj\_pref}\ l_1\ l_2$ is only well-defined if there is are l'_1, l'_2, and l s.t. $l_1 = l'_1 \frown l$ and $l_2 = l'_2 \frown l$, and if both l'_1 and l'_2 are disjoint except at their endpoints.

We now are able to state the following two lemmas concerning the construction of a blossom or an augmenting path given paths resulting from alternating trees search.

▶ **Lemma 1.** *If γ_1 and γ_2 are both (i) simple paths w.r.t. $\mathcal{G}$, (ii) alternating paths w.r.t. $\mathcal{M}$, and (iii) of odd length, and if we have that (iv)* $\mathsf{last}\ \gamma_1 = \mathsf{last}\ \gamma_2$*, (v)* $\mathsf{last}\ \gamma_1 \notin \bigcup \mathcal{M}$*, (vi)* $\{\mathsf{first}\ \gamma_1, \mathsf{first}\ \gamma_2\} \in \mathcal{G}$*, (vii)* $\{\mathsf{first}\ \gamma_1, \mathsf{first}\ \gamma_2\} \notin \mathcal{M}$*, and (viii)* $\mathsf{longest\_disj\_pref}\ \gamma_1\ \gamma_2$ *is well-defined and* $\langle \gamma'_1, \gamma'_2\rangle = \mathsf{longest\_disj\_pref}\ \gamma_1\ \gamma_2$*, then* $\langle \mathsf{rev}(\mathsf{drop}\ (|\gamma'_1| - 1)\ \gamma_1), (\mathsf{rev}\ \gamma'_1) \frown \gamma'_2\rangle$ *is a blossom w.r.t. $\langle \mathcal{G}, \mathcal{M}\rangle$.*

▶ **Lemma 2.** *If γ_1 and γ_2 are both (i) simple paths w.r.t. $\mathcal{G}$, (ii) alternating paths w.r.t. $\mathcal{M}$, (iii) of odd length, and (iv) disjoint, and if we have that (v)* $\mathsf{last}\ \gamma_1 \notin \bigcup \mathcal{M}$*, (vi)* $\mathsf{last}\ \gamma_2 \notin \bigcup \mathcal{M}$*, (vii)* $\mathsf{last}\ \gamma_1 \neq \mathsf{last}\ \gamma_2$*, (viii)* $\{\mathsf{first}\ \gamma_1, \mathsf{first}\ \gamma_2\} \in \mathcal{G}$*, and (ix)* $\{\mathsf{first}\ \gamma_1, \mathsf{first}\ \gamma_2\} \notin \mathcal{M}$*, then* $(\mathsf{rev}\ \gamma_1) \frown \gamma_2$ *is an augmenting path w.r.t. $\langle \mathcal{G}, \mathcal{M}\rangle$.*

Based on the above lemmas we refine the algorithm COMPUTE_BLOSSOM as shown in Algorithm 3.

Algorithm 3 COMPUTE_BLOSSOM($\mathcal{G}, \mathcal{M}$).

```
if $\exists e \in \mathcal{G}. e \cap \bigcup \mathcal{M} = \emptyset$
  return Augmenting path choose $\{e \mid e \in \mathcal{G} \wedge e \cap \bigcup \mathcal{M} = \emptyset\}$
else if compute_alt_path$(\mathcal{G}, \mathcal{M}) = \langle \gamma_1, \gamma_2\rangle$
  if last $\gamma_1 \neq$ last $\gamma_2$
    return Augmenting path (rev $\gamma_1) \frown \gamma_2$
  else
    $\langle \gamma'_1, \gamma'_2\rangle =$ longest_disj_pref $\gamma_1\ \gamma_2$
    return Blossom $\langle$rev(drop $(|\gamma'_1| - 1)\ \gamma_1)$, (rev $\gamma'_1) \frown \gamma'_2\rangle$
else
  return No blossom or augmenting path found
```

The following corollary shows the conditions under which COMPUTE_BLOSSOM works.

▶ **Corollary 3.** *Assume the function* $\mathsf{compute\_alt\_path}(\mathcal{G}, \mathcal{M})$ *returns two lists of vertices $\langle \gamma_1, \gamma_2\rangle$ s.t. both lists are (i) simple paths w.r.t. $\mathcal{G}$, (ii) alternating paths w.r.t. $\mathcal{M}$, and (iii) of odd length, and also (iv)* $\mathsf{last}\ \gamma_1 \notin \bigcup \mathcal{M}$*, (v)* $\mathsf{last}\ \gamma_2 \notin \bigcup \mathcal{M}$*, (vi)* $\{\mathsf{first}\ \gamma_1, \mathsf{first}\ \gamma_2\} \in \mathcal{G}$*, and (vii)* $\{\mathsf{first}\ \gamma_1, \mathsf{first}\ \gamma_2\} \notin \mathcal{M}$*, iff two lists of vertices with those properties exist. Then there is a blossom or an augmenting path w.r.t. $\langle \mathcal{G}, \mathcal{M}\rangle$ iff* COMPUTE_BLOSSOM($\mathcal{G}, \mathcal{M}$) *is a blossom or an augmenting path w.r.t. $\langle \mathcal{G}, \mathcal{M}\rangle$.*

In Isabelle/HOL, to formalise the function COMPUTE_BLOSSOM, we firstly defined a function, *longest_disj_pfx*, which finds the longest common prefix in a straightforward fashion with a quadratic wort-case runtime. The formalised versions of Lemma 1 and 2, which show that the output of *longest_disj_pfx* can be used to construct a blossom or an augmenting path are as follows:

```
lemma common_pfxs_form_blossom:
  assumes
    (Some pfx1, Some pfx2) = longest_disj_pfx p1 p2"
    and
    p1 = pfx1 @ p and p2 = pfx2 @ p"
    and
    alt_path M p1 and alt_path M p2 and last p1 ∉ Vs M and {hd p1, hd p2} ∈ M"
    and
    hd p1 ≠ hd p2"
    and
    even (length p1) and even (length p2)
    and
    distinct p1 and distinct p2
    and
    matching M
  shows blossom M (rev (drop (length pfx1) p1)) (rev pfx1 @ pfx2)
```

```
lemma construct_aug_path:
  assumes
    set p1 ∩ set p2 = {}
    and
    p1 ≠ [] and p2 ≠ []
    and
    alt_path M p1 and alt_path M p2 and last p1 ∉ Vs M and last p2 ∉ Vs M
    and
    {hd p1, hd p2} ∈ M and
    and
    even (length p1) and even (length p2)
  shows augmenting_path M ((rev p1) @ p2)
```

The function COMPUTE_BLOSSOM is formalised as follows:

```
"compute_blossom G M ≡
 (if (∃e. e ∈ unmatched_edges G M) then
       let
         singleton_path =
            (SOME p. ∃v1 v2. p = [v1 ,v2] ∧ {v1, v2} ∈ unmatched_edges G M)
       in
         Some (Path singleton_path)
   else
    case compute_alt_path G M
      of Some (p1,p2) ⇒
        (if (set p1 ∩ set p2 = {}) then
           Some (Path ((rev p1) @ p2))
         else
           (let
              (pfx1, pfx2) = longest_disj_pfx p1 p2;
              stem = (rev (drop (length (the pfx1)) p1));
              cycle = (rev (the pfx1) @ (the pfx2))
```

```
          in
           (Some (Blossom stem cycle))))
    | _ ⇒ None)"
```

We use a locale again to formalise that function. That locale parameterises it on a function that searches for alternating paths and poses the soundness and completeness assumptions for that alternating path search function. This function is equivalent to the unspecified function compute_alt_path in Corollary 3 and locale's assumptions on it are formalised statements of the seven assumptions on compute_alt_path in Corollary 3.

5.6 Computing Alternating Paths

Lastly, we refine the function compute_alt_path to an algorithmic specification. The algorithmic specification of that function performs the alternating tree search, see Algorithm 4. If the function positively terminates, i.e. finding two vertices with even labels, returns two alternating paths by ascending the two alternating trees to which the two vertices belong. This tree ascent is performed by the function follow . That function takes a functional argument f and a vertex, and returns the singleton list $[u]$ if $f(u) = \text{None}$, and $u :: (\text{follow } f\ (f(u)))$ otherwise.

Algorithm 4 compute_alt_path($\mathcal{G}, \mathcal{M}$).

```
ex = $\emptyset$ // Set of examined edges
foreach $u \in \bigcup \mathcal{G}$
    label $u$ = None
    parent $u$ = None
$U = \bigcup \mathcal{G} \setminus \bigcup \mathcal{M}$ // Set of unmatched vertices
foreach $u \in U$
    label $u = \langle u, \text{even} \rangle$
while $(\mathcal{G} \setminus \text{ex}) \cap \{e \mid \exists u \in e, r \in \bigcup \mathcal{G}.\text{label } u = \langle r, \text{even} \rangle\} \neq \emptyset$
    // Choose a new edge and labelled it examined
    $\{u_1, u_2\} = \text{choose } (\mathcal{G} \setminus \text{ex}) \cap \{\{u_1, u_2\} \mid \exists r.\text{label } u_1 = \langle r, \text{even} \rangle\}$
    $\text{ex} = \text{ex} \cup \{\{u_1, u_2\}\}$
    if label $u_2$ = None
        // Grow the discovered set of edges from $r$ by two
        $u_3 = \text{choose } \{u_3 \mid \{u_2, u_3\} \in \mathcal{M}\}$
        $\text{ex} = \text{ex} \cup \{\{u_2, u_3\}\}$
        label $u_2 = \langle r, \text{odd} \rangle$; label $u_3 = \langle r, \text{even} \rangle$; parent $u_2 = u_1$; parent $u_3 = u_2$
    else if $\exists s \in \bigcup \mathcal{G}.\text{label } u_2 = \langle s, \text{even} \rangle$
        // Return two paths from current edge's tips to unmatched vertex(es)
        return $\langle \text{follow parent } u_1, \text{follow parent } u_2 \rangle$
return No paths found
```

The soundness and completeness of Algorithm 4 is stated as follows.

▶ **Theorem 3.** *The function* compute_alt_path($\mathcal{G}, \mathcal{M}$) *returns two lists of vertices* $\langle \gamma_1, \gamma_2 \rangle$ *s.t. both lists are (i) simple paths w.r.t.* $\mathcal{G}$*, (ii) alternating paths w.r.t.* $\mathcal{M}$*, and (iii) of odd length, and also (iv)* last $\gamma_1 \notin \bigcup \mathcal{M}$*, (v)* last $\gamma_2 \notin \bigcup \mathcal{M}$*, (vi)* $\{\text{first } \gamma_1, \text{first } \gamma_2\} \in \mathcal{G}$*, and (vii)* $\{\text{first } \gamma_1, \text{first } \gamma_2\} \notin \mathcal{M}$*, iff two lists of vertices with those properties exist.*

The primary difficulty with proving this theorem is identifying the loop invariants, which are as follows:

(i) For any vertex u, if for some r, label $u = \langle r, \mathsf{even}\rangle$, then the vertices in the list follow parent u have labels that alternate between $\langle r, \mathsf{even}\rangle$ and $\langle r, \mathsf{odd}\rangle$.

(ii) For any vertex u_1, if for some r and some l, we have label $u_1 = \langle r, l\rangle$, then the vertex list $u_1 u_2 \ldots u_n$ returned by follow parent u_1 has the following property: if label $u_i = \langle r, \mathsf{even}\rangle$ and label $u_{i+1} = \langle r, \mathsf{odd}\rangle$, for some r, then $\{u_i, u_{i+1}\} \in \mathcal{M}$, otherwise, $\{u_i, u_{i+1}\} \notin \mathcal{M}$.

(iii) The relation induced by the function parent is well-founded.

(iv) For any $\{u_1, u_2\} \in \mathcal{M}$, label $u_1 = \text{None}$ iff label $u_2 = \text{None}$.

(v) For any u_1, if label $u_1 = \text{None}$ then parent $u_2 \neq u_1$, for all u_2.

(vi) For any u, if label $u \neq \text{None}$, then last (follow parent u) $\notin \bigcup \mathcal{M}$.

(vii) For any u, if label $u \neq \text{None}$, then label (last (follow parent u)) $= \langle r, \mathsf{even}\rangle$, for some r.

(viii) For any $\{u_1, u_2\} \in \mathcal{M}$, if label $u_1 \neq \text{None}$, then $\{u_1, u_2\} \in \mathsf{ex}$.

(ix) For any u, follow parent u is a simple path w.r.t. $\mathcal{G}$.

(x) Suppose we have two vertex lists γ_1 and γ_2, s.t. both lists are (i) simple paths w.r.t. $\mathcal{G}$, (ii) alternating paths w.r.t. $\mathcal{M}$, and (iii) of odd length, and also (iv) last $\gamma_1 \notin \bigcup \mathcal{M}$, (v) last $\gamma_2 \notin \bigcup \mathcal{M}$, (vi) $\{\mathsf{first}\ \gamma_1, \mathsf{first}\ \gamma_2\} \in \mathcal{G}$, and (vii) $\{\mathsf{first}\ \gamma_1, \mathsf{first}\ \gamma_2\} \notin \mathcal{M}$. Then there is at least an edge from the path rev $\gamma_1 \frown \gamma_2$ which is a member of neither $\mathcal{M}$ nor ex.[4]

To formalise Algorithm 4 in Isabelle/HOL, we first define the function which follows a vertex's parent as follows:

```
follow v = (case (parent v) of Some v' ⇒ v # (follow v') | _  ⇒ [v])
```

Again, we use a locale to formalise that function, and that locale fixes the function *parent*. Note that the above function is not well-defined for all possible arguments. In particular, it is only well-defined if the relation between pairs of vertices induced by the function *parent* is a well-founded relation. This assumption on *parent* is a part of the locale's definition.

Then, we then formalise compute_alt_path as follows:

```
compute_alt_path ex par flabel =
   (if (∃v1 v2. {v1, v2} ∈ G - ex ∧ (∃r. flabel v1 = Some (r, Even))) then
      let
        (v1,v2) = (SOME (v1,v2). {v1, v2} ∈ G - ex ∧
                                 (∃r. flabel v1 = Some (r, Even)));
        ex' = insert {v1, v2} ex;
        r = (SOME r. flabel v1 = Some (r, Even))
      in
        (if flabel v2 = None ∧ (∃v3. {v2, v3} ∈ M) then
          let
            v3 = (SOME v3. {v2, v3} ∈ M);
            par' = par(v2 := Some v1, v3 := Some v2);
            flabel' = flabel(v2 := Some (r, Odd), v3 := Some (r, Even));
            ex'' = insert {v2, v3} ex';
            return = compute_alt_path ex'' par' flabel'
          in
            return
        else if ∃r. flabel v2 = Some (r, Even) then
           let
             r' = (SOME r'. flabel v2 = Some (r', Even));
             return = Some (parent.follow par v1, parent.follow par v2)
```

[4] The hypothesis of this invariant is equivalent to the existence of an augmenting path or a blossom w.r.t. $\langle \mathcal{G}, \mathcal{M}\rangle$.

```
            in
              return
         else
           let
              return = None
           in
             return)
     else
       let
         return = None
       in
         return)
```

Note that we do not use a while combinator to represent the while loop: instead we formalise it recursively, passing the context along recursive calls. In particular, we define it as a recursive function which takes as arguments the variables representing the state of the while loop, namely, the set of examined edges `ex`, the parent function `par`, and the labelling function `flabel`.

5.7 Discussion

The algorithm in LEDA differs from the description above in one aspect. If no augmenting path is found, an odd-set cover is constructed proving optimality. Also the correctness proof uses the odd-set cover instead of the fact that an augmenting path exists in the original graph if and only if one exists in the quotient graph.

For an efficient implementation, the shrinking process and the lifting of augmenting paths are essential. The shrinking process is implemented using a union-find data structure and the lifting is supported by having each node in a contracted cycle point to the edge that closes the cycle in a blossom [34].

6 Level Five of Trustworthiness: Extraction of Efficient Executable Code

In this section we examine the process of obtaining trustworthy executable and efficient code from algorithms verified in theorem provers. First we discuss the problem in general and then we examine our formalization of the blossom-shrinking algorithm.

Most theorem provers are connected to a programming language of some sort. Frequently, as in the case of Isabelle/HOL, that programming language is a subset of the logic and close to a functional programming language. The theorem prover will usually support the extraction of actual code in some programming language. Isabelle/HOL supports Standard ML, Haskell, OCaml and Scala.

To show that code extraction "works", here are some random non-trivial examples of verifications that have resulted in reasonably efficient code: Compilers for C [29] and for ML [23], a SPIN-like model checker [13], network flow algorithms [26] and the Berlekamp-Zassenhaus factorization algorithm [11].

We will now discuss some approaches to obtaining code from function definitions in a theorem prover. In the ACL2 theorem prover all functions are defined in a purely functional subset of Lisp and are thus directly executable. In other systems, code generation involves an explicit translation step. The trustworthiness of this step varies. Probably the most trustworthy code generator is that of HOL4, because its backend is a verified compiler for

CakeML [23], a dialect of ML. The step from HOL to CakeML is not verified once and for all, but every time it is run it produces a theorem that can be examined and that states the correctness of this run [36]. The standard code generator in Isabelle/HOL is unverified (although its underlying theory has been proved correct on paper [18]). There is ongoing work to replace it with a verified code generator that produces CakeML [19].

So far we have only considered purely functional code but efficient algorithms often make use of imperative features. Some theorem provers support imperative languages directly, e.g. Java [4]. We will now discuss how to generate imperative code from purely functional one. Clearly the code generator must turn the purely functional definitions into more imperative ones. The standard approach [7, 36] is to let the code generator recognize monadic definitions (a purely functional way to express imperative computations) and implement those imperatively. This is possible because many functional programming languages do in fact offer imperative features as well.

Just as important as the support for code extraction is the support for verified stepwise refinement of data types and algorithms by the user. Data refinement means the replacement of abstract data types by concrete efficient ones, e.g. sets by search trees. Algorithm refinement means the stepwise replacement of abstract high-level definitions that may not even be executable by efficient implementations. Both forms of refinement are supported well in Isabelle/HOL [17, 24, 25].

We conclude this section with a look at code generation from our formalization of the blossom-shrinking algorithm. It turns out that our formalization is almost executable as is. The only non-executable construct we used is *SOME* `x. P` that denotes some arbitrary `x` that satisfies the predicate `P`. Of course one can hide arbitrarily complicated computations in such a contruct but we have used it only for simple nondeterministic choices and it will be easy to replace. For example, one can obtain an executable version of function `choose_con_vert` (see Section 5.4) by defining a function that searches the vertex list `vs` for the first `v'` such that `{v, v'}` $\in$ `E`. This is an example of algorithm refinement. To arrive at efficient code for the blossom-shrinking algorithm as a whole we will need to apply both data and algorithm refinement down to the imperative level. At least the efficient implementations referred to above, just before Section 5.1, are intrinsically imperative.

Finally let us note that instead of code generation it is also possible to verify existing code in a theorem prover. This was briefly mentioned in Section 4 and Charguéraud [8] has followed this approach quite successfully.

7 The Future

The state of the art in the verification of complex algorithms has improved enormously over the last decade. Yet there is still a lot to do on the path to a verified library such as LEDA. Apart from the shere amount of material that would have to be verified there is the challenge of obtaining trustworthy code that is of comparable efficiency. This requires trustworthy code generation for a language such C or C++, including the memory management. This is a non-trivial task, but some of the pieces of the puzzle, like a verified compiler, are in place already.

References

1 Mohammad Abdulaziz, Kurt Mehlhorn, and Tobias Nipkow. A Correctness Proof of Edmonds' Blossom Shrinking Algorithm for Maximum Matchings in Graphs. forthcoming.

2 Mohammad Abdulaziz, Michael Norrish, and Charles Gretton. Formally Verified Algorithms for Upper-Bounding State Space Diameters. *J. Autom. Reasoning*, 61(1-4):485–520, 2018. doi:10.1007/s10817-018-9450-z.

3 Mohammad Abdulaziz and Lawrence C. Paulson. An Isabelle/HOL Formalisation of Green's Theorem. *Archive of Formal Proofs*, 2018, 2018. URL: https://www.isa-afp.org/entries/Green.html.

4 Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reiner Hähnle, Peter H. Schmitt, and Mattias Ulbrich, editors. *Deductive Software Verification - The KeY Book - From Theory to Practice*, volume 10001 of *Lecture Notes in Computer Science*. Springer, 2016. doi:10.1007/978-3-319-49812-6.

5 E. Alkassar, S. Böhme, K. Mehlhorn, and Ch. Rizkallah. A Framework for the Verification of Certifying Computations. *Journal of Automated Reasoning (JAR)*, 52(3):241–273, 2014. A preliminary version appeared under the title "Verification of Certifying Computations" in CAV 2011, LCNS Vol 6806, pages 67 – 82. arXiv:1301.7462.

6 E. Althaus and K. Mehlhorn. Maximum Network Flow with Floating Point Arithmetic. *Info. Proc. Lett.*, 66:109–113, 1998.

7 Lukas Bulwahn, Alexander Krauss, Florian Haftmann, Levent Erkök, and John Matthews. Imperative Functional Programming with Isabelle/HOL. In *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008*, pages 134–149, 2008. doi:10.1007/978-3-540-71067-7_14.

8 Arthur Charguéraud. Characteristic formulae for the verification of imperative programs. In *Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, September 19-21, 2011*, pages 418–430, 2011. doi:10.1145/2034773.2034828.

9 Ernie Cohen, Markus Dahlweid, Mark Hillebrand, Dirk Leinenbach, Michał Moskal, Thomas Santen, Wolfram Schulte, and Stephan Tobies. VCC: A practical system for verifying concurrent C. In *Theorem Proving in Higher Order Logics (TPHOLs 2009)*, volume 5674 of *LNCS*, pages 23–42. Springer, 2009. doi:10.1007/978-3-642-03359-9_2.

10 Sander R Dahmen, Johannes Hölzl, and Robert Y Lewis. Formalizing the Solution to the Cap Set Problem. *arXiv*, 2019. arXiv:1907.01449.

11 Jose Divasón, Sebastiaan J. C. Joosten, René Thiemann, and Akihisa Yamada. A Verified Implementation of the Berlekamp-Zassenhaus Factorization Algorithm. *J. Autom. Reasoning*, 2019. published online.

12 J. Edmonds. Maximum Matching and a polyhedron with 0,1 - vertices. *Journal of Research of the National Bureau of Standards*, 69B:125–130, 1965.

13 Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A Fully Verified Executable LTL Model Checker. In *Computer Aided Verification - 25th International Conference, CAV 2013*, pages 463–478, 2013. doi:10.1007/978-3-642-39799-8_31.

14 S. Fortune. Robustness Issues in Geometric Algorithms. In *Proceedings of the 1st Workshop on Applied Computational Geometry: Towards Geometric Engineering (WACG'96)*, volume 1148 of *LNCS Series*, pages 9–13, 1996.

15 H. N. Gabow. An efficient implementation of Edmonds' algorithm for maximum matching on graphs. *J. ACM*, 23:221–234, 1976.

16 David Greenaway, June Andronick, and Gerwin Klein. Bridging the Gap: Automatic Verified Abstraction of C. In *Interactive Theorem Proving*, volume 7406 of *LNCS*, pages 99–115, 2012.

17 Florian Haftmann, Alexander Krauss, Ond*v*rej Kun*v*car, and Tobias Nipkow. Data Refinement in Isabelle/HOL. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving (ITP 2013)*, volume 7998 of *LNCS Series*, pages 100–115, 2013.

18 Florian Haftmann and Tobias Nipkow. Code Generation via Higher-Order Rewrite Systems. In M. Blume, N. Kobayashi, and G. Vidal, editors, *Functional and Logic Programming (FLOPS 2010)*, volume 6009 of *LNCS*, pages 103–117. Springer, 2010.

19 Lars Hupel and Tobias Nipkow. A Verified Compiler from Isabelle/HOL to CakeML. In A. Ahmed, editor, *European Symposium on Programming (ESOP 2018)*, volume 10801 of *LNCS*, pages 999–1026. Springer, 2018.

20 L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom Examples of Robustness Problems in Geometric Computations. *Computational Geometry: Theory and Applications (CGTA)*, 40:61–78, 2008. a preliminary version appeared in ESA 2004, LNCS 3221, pages 702 – 713.

21 Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an operating-system kernel. *CACM*, 53(6):107–115, 2010.

22 B. Korte and J.Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2012.

23 Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. CakeML: a verified implementation of ML. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14*, pages 179–192. ACM, 2014.

24 Peter Lammich. Automatic Data Refinement. In *Interactive Theorem Proving - 4th International Conference, ITP 2013*, pages 84–99, 2013. doi:10.1007/978-3-642-39634-2_9.

25 Peter Lammich. Refinement to Imperative HOL. *J. Autom. Reasoning*, 62(4):481–503, 2019. doi:10.1007/s10817-017-9437-1.

26 Peter Lammich and S. Reza Sefidgar. Formalizing Network Flow Algorithms: A Refinement Approach in Isabelle/HOL. *J. Autom. Reasoning*, 62(2):261–280, 2019. doi:10.1007/s10817-017-9442-4.

27 LEDA (Library of Efficient Data Types and Algorithms). www.algorithmic-solutions.com.

28 LEMON graph library. COIN-OR project.

29 Xavier Leroy. A Formally Verified Compiler Back-end. *Journal of Automated Reasoning*, 43:363–446, 2009.

30 R.M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011.

31 K. Mehlhorn and S. Näher. LEDA: A library of efficient data types and algorithms. In *MFCS'89*, volume 379 of *LNCS Series*, pages 88–106, 1989.

32 K. Mehlhorn and S. Näher. The implementation of geometric algorithms. In *Proceedings of the 13th IFIP World Computer Congress*, volume 1, pages 223–231. Elsevier Science B.V. North-Holland, Amsterdam, 1994.

33 K. Mehlhorn and S. Näher. From Algorithms to Working Programs: On the Use of Program Checking in LEDA. In *MFCS'98*, volume 1450 of *LNCS Series*, pages 84–93, 1998.

34 K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

35 Robin Milner. Logic for computable functions: description of a machine implementation. Technical report, Stanford University, 1972.

36 Magnus O. Myreen and Scott Owens. Proof-producing translation of higher-order logic into pure and stateful ML. *J. Funct. Program.*, 24(2-3):284–315, 2014. doi:10.1017/S0956796813000282.

37 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL—A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS Series*. Springer, 2002.

38 Lars Noschinski. A Graph Library for Isabelle. *Mathematics in Computer Science*, 9:22–39, 2015.

39 Lars Noschinski, Christine Rizkallah, and Kurt Mehlhorn. Verification of Certifying Computations through AutoCorres and Simpl. In *NASA Formal Methods*, volume 8430 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2014.

40 Lawrence C Paulson. *Isabelle: A generic theorem prover*, volume 828. Springer, 1994.

41 Norbert Schirmer. *Verification of Sequential Imperative Programs in Isabelle/HOL*. PhD thesis, Technische Universität München, 2006.

42 Verisoft. `http://www.verisoft.de/`, 2007.

43 Andrew Lumsdaine von Jeremy G. Siek, Lie-Quan Lee. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Professional, 2001.

44 C.-K. Yap. Towards Exact Geometric Computation. *CGTA: Computational Geometry: Theory and Applications*, 7, 1997.

45 C.-K. Yap. Robust geometric computation. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41. CRC Press LLC, Boca Raton, FL, 2nd edition, 2003.

Guarded Kleene Algebra with Tests: Verification of Uninterpreted Programs in Nearly Linear Time

Alexandra Silva
University College London, UK
alexandra.silva@ucl.ac.uk

Abstract

Guarded Kleene Algebra with Tests (GKAT) is a variation on Kleene Algebra with Tests (KAT) that arises by restricting the union (+) and iteration (*) operations from KAT to predicate-guarded versions. We develop the (co)algebraic theory of GKAT and show how it can be efficiently used to reason about imperative programs. In contrast to KAT, whose equational theory is PSPACE-complete, we show that the equational theory of GKAT is (almost) linear time. We also provide a full Kleene theorem and prove completeness for an analogue of Salomaa's axiomatization of Kleene Algebra. We will also discuss how this result has practical implications in the verification of programs, with examples from network and probabilistic programming. This is joint work with Nate Foster, Justin Hsu, Tobias Kappe, Dexter Kozen, and Steffen Smolka.

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

Keywords and phrases Kleene algebra, verification, decision procedures

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.2

Category Invited Talk

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 2; pp. 2:1–2:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Picking Random Vertices

Daniel Lokshtanov
University of California, San Diego, USA
daniello@ii.uib.no

Abstract

We survey some recent graph algorithms that are based on picking a vertex at random and declaring it to be a part of the solution. This simple idea has been deployed to obtain state-of-the-art parameterized, exact exponential time, and approximation algorithms for a number of problems, such as Feedback Vertex Set and 3-Hitting Set. We will also discuss a recent 2-approximation algorithm for Feedback Vertex Set in Tournaments that is based on picking a vertex at random and declaring it to *not* be part of the solution.

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

Keywords and phrases Randomized algorithms, Parameterized algorithms, Exact algorithms, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.2

Category Invited Talk

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 2; pp. 2:1–2:1
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Popular Matchings: Good, Bad, and Mixed

Telikepalli Kavitha
Tata Institute of Fundamental Research, Mumbai, India
kavitha@tifr.res.in

Abstract

We consider the landscape of popular matchings in a bipartite graph G where every vertex has strict preferences over its neighbors. This is a very well-studied model in two-sided matching markets. A matching M is *popular* if it does not lose a head-to-head election against any matching, where each vertex casts a vote for the matching where it gets a better assignment. Roughly speaking, a popular matching is one such that there is no matching where more vertices are happier. The notion of popularity is more relaxed than *stability*: a classical notion studied for the last several decades. Popular matchings always exist in G since stable matchings always exist in a bipartite graph and every stable matching is popular.

Algorithmically speaking, the landscape of popular matching seems to have only a few bright spots. Every stable matching is a *min-size* popular matching and there are also simple linear time algorithms for computing a *max-size* popular matching and for the *popular edge* problem. All these algorithms reduce the popular matching problem to an appropriate question in stable matchings and solve the corresponding stable matching problem.

We now know NP-hardness results for many popular matching problems. These include the min-cost/max-weight popular matching problem and the problem of deciding if G admits a popular matching that is neither a min-size nor a max-size popular matching. For non-bipartite graphs, it is NP-hard to even decide if a popular matching exists or not.

A *mixed* matching is a probability distribution or a lottery over matchings. A popular mixed matching is one that never loses a head-to-head election against any mixed matching. As an allocation mechanism, a popular mixed matching has several nice properties. Moreover, finding a max-weight or min-cost popular mixed matching in G is easy (by solving a linear program). Interestingly, there is always an optimal popular mixed matching Π with a simple structure: $\Pi = \{(M_0, \frac{1}{2}), (M_1, \frac{1}{2})\}$ where M_0 and M_1 are matchings in G. Popular mixed matchings always exist in non-bipartite graphs as well and can be computed in polynomial time.

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

Keywords and phrases Matchings under preferences, Algorithms, NP-Hardness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.4

Category Invited Talk

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 4; pp. 4:1–4:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Petri Net Reachability Problem

Jérôme Leroux
Univ.Bordeaux, CNRS, Bordeaux-INP, France
leroux@labri.fr

Abstract

Petri nets, also known as vector addition systems, are a long established model of concurrency with extensive applications in modelling and analysis of hardware, software and database systems, as well as chemical, biological and business processes. The central algorithmic problem for Petri nets is reachability: whether from the given initial configuration there exists a sequence of valid execution steps that reaches the given final configuration. The complexity of the problem has remained unsettled since the 1960s, and it is one of the most prominent open questions in the theory of verification. In this presentation, we overview decidability and complexity results over the last fifty years about the Petri net reachability problem.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation

Keywords and phrases Petri net, Reachability problem, Formal verification, Concurrency

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.5

Category Invited Talk

Funding *Jérôme Leroux*: The author was supported by the grant ANR-17-CE40-0028 of the French National Research Agency ANR (project BRAVAS).

1 Outline

The presentation focuses on the reachability problem for vector addition systems with states given as multi-dimensional weighted automata. Main results about 1) reachability in small dimensions, 2) boundedness problems, and 3) decidability and complexity results for the reachability problem in any dimension will be overviewed.

- For small dimensions, the complexity of the reachability problem depends on the dimension and on the way weights are encoded (in unary or in binary). In dimension one, the reachability problem can be easily shown to be NL-complete when weights are written in unary thanks to a hill-cutting argument (the same argument that applies on pushdown automata). When updates are given in binary, this argument can only provide a complexity in between NP and PSPACE. Nevertheless, by using some additional arguments, the problem was proved to be NP-complete in [5]. The complexity of the reachability problem is also known in dimension two. Thanks to a precise analysis of the algorithm introduced in [12], the problem was proved to be PSPACE-complete in [1] for binary updates. This last result was extended later in [4] to show that the problem is NL-complete for unary updates. In dimension three the complexity of the reachability problem is nowadays open whatever the encoding of the weights.
- The Karp and Miller algorithm introduced in [6] is central for deciding the reachability problem in general dimension. It provides a way for computing the maximal value of a bounded counter. Notice that even if this value can be Ackermannian [16], deciding the boundedness of a counter is known to be EXPSPACE-complete [3] by extending the Rackoff's proof [17].

- The reachability problem was first proved to be decidable by Mayr [14, 15] in 1981. This proof was improved later by Kosaraju [7] and Lambert [8] by introducing an algorithm that decompose the set of executions. We call this decomposition the KLM decomposition (following the initials of the contributed authors). In [10] we proved that the KLM decomposition can be interpreted as ideal decompositions for a natural well-quasi order on the set of executions. In that paper we also provided a cubic-Ackermaninan complexity upper-bound of the reachability problem; the very first complexity upper-bound for the reachability problem. This bound was recently improved in [11], by proving that there exists a KLM decomposition algorithm that works in time primitive recursive in fixed dimension, and at most Ackermannian in general. Concerning lower-bounds, recently in [2], the complexity of the problem was shown to be TOWER-hard, improving the best-known EXPSPACE complexity lower-bound given by Lipton [13] in 1976. Nowadays, the exact complexity of the reachability problem is still open between TOWER and ACKERMANN. In order to close that problem, either we need to improve the recent TOWER lower bound, or we need to design an algorithm improving the ACKERMANN upper bound. The very simple algorithm introduced in [9], based on Presburger inductive invariant seems to be a good candidate for that late direction.

References

1 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in Two-Dimensional Vector Addition Systems with States Is PSPACE-Complete. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 32–43. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.14.

2 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019.*, pages 24–33. ACM, 2019. doi:10.1145/3313276.3316369.

3 Stéphane Demri. On selective unboundedness of VASS. *J. Comput. Syst. Sci.*, 79(5):689–713, 2013. doi:10.1016/j.jcss.2013.01.014.

4 Matthias Englert, Ranko Lazic, and Patrick Totzke. Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-Complete. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 477–484. ACM, 2016. doi:10.1145/2933575.2933577.

5 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in Succinct and Parametric One-Counter Automata. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, volume 5710 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2009. doi:10.1007/978-3-642-04081-8_25.

6 Richard M. Karp and Raymond E. Miller. Parallel Program Schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

7 S. Rao Kosaraju. Decidability of Reachability in Vector Addition Systems (Preliminary Version). In *STOC*, pages 267–281. ACM, 1982. doi:10.1145/800070.802201.

8 Jean-Luc Lambert. A Structure to Decide Reachability in Petri Nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992. doi:10.1016/0304-3975(92)90173-D.

9 Jérôme Leroux. Vector Addition Systems Reachability Problem (A Simpler Solution). In *Turing-100*, volume 10 of *EPiC Series in Computing*, pages 214–228. EasyChair, 2012.

10 Jérôme Leroux and Sylvain Schmitz. Demystifying Reachability in Vector Addition Systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 56–67. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.16.

11 Jérôme Leroux and Sylvain Schmitz. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. *CoRR*, abs/1903.08575, 2019. To appear at LICS'19. arXiv:1903.08575.

12 Jérôme Leroux and Grégoire Sutre. On Flatness for 2-Dimensional Vector Addition Systems with States. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2004. doi:10.1007/978-3-540-28644-8_26.

13 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976. URL: http://cpsc.yale.edu/sites/default/files/files/tr63.pdf.

14 Ernst W. Mayr. An Algorithm for the General Petri Net Reachability Problem. In *STOC*, pages 238–246. ACM, 1981. doi:10.1145/800076.802477.

15 Ernst W. Mayr. An Algorithm for the General Petri Net Reachability Problem. *SIAM J. Comput.*, 13(3):441–460, 1984. doi:10.1137/0213029.

16 Ernst W. Mayr and Albert R. Meyer. The Complexity of the Finite Containment Problem for Petri Nets. *J. ACM*, 28(3):561–576, 1981. doi:10.1145/322261.322271.

17 Charles Rackoff. The Covering and Boundedness Problems for Vector Addition Systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

An Improved Online Algorithm for the Traveling Repairperson Problem on a Line

Marcin Bienkowski
Institute of Computer Science, University of Wrocław, Poland
marcin.bienkowski@cs.uni.wroc.pl

Hsiang-Hsuan Liu
Institute of Computer Science, University of Wrocław, Poland
alison.hhliu@cs.uni.wroc.pl

Abstract

In the online variant of the traveling repairperson problem (TRP), requests arrive in time at points of a metric space $\mathcal{X}$ and must be eventually visited by a server. The server starts at a designated point of $\mathcal{X}$ and travels at most at unit speed. Each request has a given weight and once the server visits its position, the request is considered serviced; we call such time *completion time* of the request. The goal is to minimize the weighted sum of completion times of all requests.

In this paper, we give a 5.429-competitive deterministic algorithm for line metrics improving over 5.829-competitive solution by Krumke et al. (TCS 2003). Our result is obtained by modifying the schedule by serving requests that are close to the origin first. To compute the competitive ratio of our approach, we use a charging scheme, and later evaluate its properties using a factor-revealing linear program which upper-bounds the competitive ratio.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Scheduling algorithms

Keywords and phrases traveling repairperson problem, competitive analysis, minimizing completion time, factor-revealing LP

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.6

Funding Supported by Polish National Science Centre grant 2016/22/E/ST6/00499.

1 Introduction

The traveling repairperson problem (TRP) is a variant of the traveling salesperson problem (TSP), where the goal is to minimize the *total latency* instead of a more standard objective of minimizing the total length of a route. In the TRP, there are m requested points of a given metric space $\mathcal{X}$ and they must be eventually visited by a server. Request r_j is a triple (p_j, a_j, w_j), where $p_j \in \mathcal{X}$ denotes request position, $a_j \geq 0$ its release time and w_j its weight.

The server starts at a designated point of $\mathcal{X}$ called *origin* and travels at most at unit speed. That is, for any two times $t < t'$ the distance between positions of the server at times t and t' is at most $t' - t$. Each request r_j must be eventually serviced by moving the server to point p_j. The request cannot be serviced before its release time a_j; we call the time when it is eventually serviced its *completion time* and we denote it C_j. The goal is to find a route for the server (a *schedule*) that minimizes the cost, defined as the weighted sum of completion times, i.e., $\sum_{j=1}^{m} w_j \cdot C_j$.

The TRP has a natural online variant. There, an online algorithm ALG, at time t, knows only requests that arrived before or at time t. The number of requests m is also not known by an algorithm a priori. In the online setting, the goal is to minimize the competitive ratio [11], defined as the maximum over all inputs of the ALG-to-OPT cost ratio, where OPT denotes the optimal offline algorithm.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 6; pp. 6:1–6:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Previous work

The online variant has been first investigated by Feuerstein and Stougie [13]. They considered the case where $\mathcal{X}$ is a real line and, adapting an algorithm for the so-called cow-path problem [7], presented a 9-competitive solution. They also gave a lower bound (that holds also already for a line) of $1+\sqrt{2}$. The result has been subsequently improved by Krumke et al. [18], who gave a deterministic algorithm Interval attaining competitive ratio of $(1+\sqrt{2})^2 < 5.829$ and a randomized 3.874-competitive solution. Their algorithm works for an arbitrary metric space.

A natural extension to the TRP is a so-called *dial-a-ride problem*, where each request is an object with a source and a destination and the goal is to transport the object [9, 13, 18]. There, the server may have a fixed capacity allowing it to store at most k objects, or this capacity may be infinite. The 5.829-competitive deterministic algorithm by Krumke et al. [18] extends also to this variant and no better algorithms are known even for specific metric spaces.

Some papers considered an extension of the TRP to $k \geq 1$ servers. Bonifaci and Stougie showed how to adapt the algorithm Interval by Krumke et al. [18] to this setting without an increase of the competitive ratio [10]. However, the best known lower bound for multiple servers is 2 (i.e., smaller than the lower bound for the one-server case) [10]. Furthermore, for the particular case of line metrics, the ratio converges to 1 with growing k, and is between $1 + 1/(2k-1)$ [14] and $1 + O((\log k)/k)$ [10].

Another strand of papers considered different objectives, such as minimizing the total makespan [3, 4, 5, 6, 8, 9] or maximum flow time [16, 17, 19], with a special focus on line metrics. Finally, the offline variant (also known as *minimum latency problem*) has been extensively studied both from the computational hardness and approximation algorithms perspectives, see, e.g., [1, 2, 12, 15, 20, 21].

1.2 Our contribution

In this paper, we focus on the TRP on line metrics and give a 5.429-competitive algorithm Reroute. This improves the long standing record of 5.829 achieved by the algorithm Interval by Krumke et al. [18].

Similarly to the algorithm Interval, Reroute partitions an input into phases of geometrically increasing lengths, and in each phase greedily tries to service the set of pending requests of maximum weight. However, our algorithm Reroute tries to modify the route, so to ensure that (i) either it services requests only in the initial part of the phase, (ii) or in the later part of the phase, it services only requests that are far away from the origin. As such requests cannot be serviced early by any algorithm (also by Opt), this allows us to charge the cost of Reroute against the cost of Opt in a more efficient way.

For the analysis, we construct a charging scheme that maps the total weight serviced by Reroute in particular time intervals to the total weight serviced in appropriate intervals by Opt. This yields a set of linear inequalities that need to hold for any input instance. On this basis, we create a maximization LP (linear program), whose objective value is an upper bound on the competitive ratio. Finally, to bound the value of such factor-revealing LP, we explicitly construct a solution to its dual.

2 Algorithms Interval and ReturnFirst

As our algorithm is built on the phase-based approach of the algorithm Interval proposed by Krumke et al. [18], we start with a description of the latter.

Let $f = \min_j \{\max\{p_j, a_j\}\}$ be the earliest time at which Opt may service a request. (Note that an online algorithm can learn f before or at time f.) Without loss of generality, we may assume that there are no requests that arrive at time 0 at the origin, and hence $f > 0$.

We partition time into phases. Phase 0 starts at time 0 and ends at time f. Phase $i \geq 1$ starts at time $f \cdot \alpha^{i-1}$ and ends at time $f \cdot \alpha^i$, where $\alpha = 1 + \sqrt{2}$. At the beginning of any phase $i \geq 1$, Interval computes and executes a schedule that

- starts at the final server position from the last phase;
- stops at distance at most $f \cdot \alpha^{i-1}$ from the origin;
- has length at most $f \cdot (\alpha^i - \alpha^{i-1})$;
- among schedules satisfying the previous three conditions maximizes the total weight of serviced requests (which are pending when the phase starts).

To simplify the notation, in the rest of the paper, we assume that $f = 1$.[1] We start with a slight modification of the algorithm Interval, called ReturnFirst (RetF). At the beginning of any phase $i \geq 1$, RetF computes and executes a schedule that

- starts at the final server position from the last phase;
- in the first part of the schedule (called *return part*), the server returns to the origin;
- the second part of the schedule (called *serving part*) starts at the origin, is of length at most α^{i-1}, and among such schedules maximizes the total weight of serviced requests (which are pending when the phase starts).

▶ **Observation 1.** *At the beginning of each phase $i \geq 1$, RetF has its server at a distance at most α^{i-2} from the origin. Furthermore, the total length of both parts of the schedule is at most the phase length.*

Proof. The first property is clearly satisfied at the beginning of phase 1, which is started with the server at the the origin. In the subsequent phases, this property follows inductively: as in phase $i - 1$ the serving part of the schedule starts at the origin and has length at most α^{i-2}, at be beginning of phase i the server distance to the origin is at most α^{i-2}.

The second property follows as the total length of the planned schedule in phase i is at most $\alpha^{i-2} + \alpha^{i-1} = \alpha^i - \alpha^{i-1}$, which is the length of the phase. Note that this property holds as long as $\alpha \geq 1 + \sqrt{2}$. ◀

While RetF may produce schedules that are worse than those of Interval, using similar arguments to those of [18], one can show that RetF is α^2-competitive. In particular, the following bound holds both for Interval and RetF; we present its proof for completeness.

In our arguments, we use Alg_i and Opt_i to denote the total weight of requests serviced by an online algorithm and Opt, respectively, in a phase i. Observe that for RetF and Interval, $\text{Alg}_0 = 0$.

▶ **Lemma 2** ([18]). *Let L be the index of the last phase in which Opt services any request. Then, RetF services all requests within the first $L + 1$ phases, and for any phase $j \in \{1, \ldots, L+1\}$, it holds that $\sum_{i=j}^{L+1} \text{Alg}_i \leq \sum_{i=j-1}^{L} \text{Opt}_i$.*

[1] All terms occurring in the proof, both related to distances and to time, have a multiplicative factor f, which cancels out when the competitive ratio is computed.

Proof. Consider the schedule S_k of $\textsc{Opt}$ in phases $0, 1, \ldots, k$, where $k \in \{0, \ldots, L\}$. Schedule S_k starts at the origin, its length is equal to α^k and the total weight serviced by S_k is $w(S_k) = \sum_{i=0}^{k-1} \textsc{Opt}_i$. In phase $k+1$, when $\textsc{RetF}$ chooses the route for the serving part, S_k is among feasible options. If $\textsc{RetF}$ chose such route, then each request serviced by schedule S_k in $\textsc{Opt}$'s solution is serviced by $\textsc{RetF}$ (in phase $k+1$ or already in earlier phases). Thus, the total weight serviced by $\textsc{RetF}$ in phases $1, \ldots, k+1$ would be at least $w(S_k)$. As $\textsc{RetF}$ chooses the schedule for the serving part which is at least as good as S_k, it holds that

$$\textstyle\sum_{i=1}^{k+1} \textsc{Alg}_i \geq w(S_k) = \sum_{i=0}^{k} \textsc{Opt}_i \,. \tag{1}$$

If we set $k = L$, then the above inequality implies that $\textsc{RetF}$ services all requests already in the first $L+1$ phases, i.e.,

$$\textstyle\sum_{i=1}^{L+1} \textsc{Alg}_i = \sum_{i=0}^{L} \textsc{Opt}_i \,. \tag{2}$$

The lemma for $j = 1$ follows by (2), and the lemma for $j \geq 2$ follows by subtracting (1) from (2) and setting $k = j - 2$. ◀

3 Modifying the schedule

We now take a closer look at the structure of schedules produced by $\textsc{RetF}$ in particular phases. We show that the schedule produced in a given phase can be modified, so that it services the same set of requests as $\textsc{RetF}$, but either it ends substantially earlier than the phase end or from some time it services only far requests.

▶ **Lemma 3.** *Fix any $c \in [0, 1]$ and let $\ell = \alpha^{i-1} + c \cdot \alpha^{i-2}$. Fix a schedule S for phase i produced by the algorithm $\textsc{RetF}$. On the basis of S and c, it is possible to construct schedule $\tilde{S}$ that services the same set of requests as S and*

- *Either the length of $\tilde{S}$ is at most ℓ,*
- *or after executing the prefix of $\tilde{S}$ of length ℓ, the server distance from the origin is at least $(1/3) \cdot (\alpha - 1 + 5c) \cdot \alpha^{i-2}$, and afterwards the server travels away from the origin (with unit speed).*

Proof. In the following proof, server positions are real numbers, with zero denoting the origin. Let p denote the server position at the beginning of phase i. Note that we may assume that $p \geq 0$ without loss of generality. As in the proof of Observation 1, it can be inductively shown that in the produced schedule $\tilde{S}$, the server ends phase $i-1$ at most at distance α^{i-2} from the origin. Thus, $p \leq \alpha^{i-2}$.

Let $[b, u]$ be the interval containing all points visited by the serving part of S. As this part starts at zero, $b \leq 0 \leq u$. Let $x \in \{b, u\}$ be the interval endpoint closer to the origin and y be the further one (with ties broken arbitrarily). Observe that the shortest possible schedule that starts at zero and services all requests from interval $[b, u]$ has length $2 \cdot |x| + |y|$. As the serving part of S services this interval and its length is at most α^{i-1}, it holds that

$$2 \cdot |x| + |y| \leq \alpha^{i-1} \text{ and thus } |x| \leq \alpha^{i-1}/3 \,. \tag{3}$$

We use (3) extensively in our bounds below.

We define two possible schedules S_{xy} and S_{yx} for phase i; we show that at least one of them satisfies the requirements of the lemma.

- S_{xy} starts at p, goes to x (possibly going through zero if p and x are on the opposite sides of the origin), and then proceeds through 0 to y. Its length is $|p - x| + |x| + |y|$.

- S_{yx} starts at p, goes to y (possibly going through zero if p and y are on the opposite sides of the origin), and then proceeds through 0 to x. Its length is $|p-y|+|y|+|x|$.

We consider four cases depending on values of x, y, p and c. Note that all these values are known by RetF at the beginning of phase i.

Case 1. If $x<0$ and $p+3\cdot|x| \geq (\alpha-c)\cdot\alpha^{i-2}$, we set $\tilde{S}=S_{yx}$.
We show that the length of $\tilde{S}$ is at most ℓ. As $x<0$, it holds that $y\geq 0$. If $y\leq p$, the length of $\tilde{S}$ is $(p-y)+y+|x| = p+|x| \leq \alpha^{i-2}+\alpha^{i-1}/3 < \alpha^{i-1} \leq \ell$. Otherwise $y>p$, and then the length of $\tilde{S}$ is

$$\begin{aligned}(y-p)+y+|x| &= 2\cdot y - p + |x| \leq 2\cdot(\alpha^{i-1}-2\cdot|x|) - p + |x| \\ &= 2\cdot\alpha^{i-1} - (p+3\cdot|x|) \leq 2\cdot\alpha^{i-1} - (\alpha-c)\cdot\alpha^{i-2} = \ell\,.\end{aligned}$$

Case 2. If $x<0$ and $p+3\cdot|x| < (\alpha-c)\cdot\alpha^{i-2}$, we set $\tilde{S}=S_{xy}$.
If the length of $\tilde{S}$ is at most ℓ, then the lemma follows. Otherwise, we analyze the prefix of $\tilde{S}$ of length ℓ. It contains the server movement from p to x and then to 0: this holds because the total length of this movement is $p+2\cdot|x| < (\alpha-c)\cdot\alpha^{i-2} \leq \alpha^{i-1} \leq \ell$. Thus, after having executed the prefix of $\tilde{S}$ of length ℓ, the server is traveling away from the origin towards y and its position is equal to

$$\begin{aligned}\ell-(p+2\cdot|x|) &= \alpha^{i-1}+c\cdot\alpha^{i-2}-(2/3)\cdot(p+3\cdot|x|)-p/3 \\ &> \alpha^{i-1}+c\cdot\alpha^{i-2}-(2/3)\cdot(\alpha-c)\cdot\alpha^{i-2}-\alpha^{i-2}/3 \\ &= (1/3)\cdot(\alpha-1+5c)\cdot\alpha^{i-2}\,.\end{aligned}$$

Case 3. If $x\geq p\geq 0$, we set $\tilde{S}=S_{xy}$.
The length of $\tilde{S}$ is then $(x-p)+x+|y| = 2\cdot x+|y|-p \leq \alpha^{i-1}-p \leq \ell$.

Case 4. If $p>x\geq 0$, we set $\tilde{S}=S_{xy}$.
The reasoning here is similar to the one from Case 2. If the length of $\tilde{S}$ is at most ℓ, then the lemma follows. Otherwise, we analyze the prefix of $\tilde{S}$ of length ℓ. It contains the server movement from p to 0 through x: this holds because the length of this movement is equal to $p\leq\alpha^{i-2}<\ell$. Thus, after having executed the prefix of $\tilde{S}$ of length ℓ, the server is traveling away from the origin towards y and its distance from the origin is equal to

$$\ell-p \geq \alpha^{i-1}+c\cdot\alpha^{i-2}-\alpha^{i-2} = (\alpha+c-1)\cdot\alpha^{i-2} > (1/3)\cdot(\alpha-1+5c)\cdot\alpha^{i-2}\,.$$

The last inequality follows as $\alpha > 1+c$. ◀

4 Algorithm Reroute

Our algorithm Reroute(β) is parameterized with a constant $\beta\in[2/\alpha,1]$ and follows the phase framework of RetF. At the beginning of any phase $j\geq 1$, Reroute(β) computes the schedule S in the same way RetF would do, modifies it according to Lemma 3 using $c=\beta\cdot\alpha^2-2\cdot\alpha\in[0,1]$ obtaining schedule $\tilde{S}$, and then executes $\tilde{S}$ within phase j.

For any real $\xi\geq[\beta,1]$, we define

$$\tau_\beta(\xi) = \frac{5\cdot\beta\cdot\alpha^2-9\alpha-1}{3\alpha} + (\xi-\beta)\cdot\alpha\,. \tag{4}$$

The following lemma shows that requests that are serviced late by Reroute(β) in a given phase are far away from the origin, and hence cannot be serviced too early by Opt.

▶ **Lemma 4.** *Fix $\beta \in [2/\alpha, 1]$. For any phase i and any value $\xi \geq \beta$, in time interval $(\xi \cdot \alpha^i, \alpha^i]$, $\textsc{Reroute}(\beta)$ services only requests whose distance to the origin is at least $\tau_\beta(\xi) \cdot \alpha^{i-1}$.*

Proof. By the definition of $\textsc{Reroute}(\beta)$, its schedule $\tilde{S}$ for phase i is constructed as described in Lemma 3 with $c = \beta \cdot \alpha^2 - 2\alpha \in [0, 1]$. Let $\ell = \alpha^{i-1} + c \cdot \alpha^{i-2} = \beta \cdot \alpha^i - \alpha^{i-1}$. Recall that phase i starts at time α^{i-1}. By Lemma 3, two cases are possible.

- If the length of $\tilde{S}$ is at most ℓ, then the execution of $\tilde{S}$ ends at or before time $\alpha^{i-1}+\ell = \beta\cdot\alpha^i$. Then, the lemma follows trivially as $\textsc{Reroute}(\beta)$ does not service anything in the remaining part of phase i, i.e., in the time interval $(\beta \cdot \alpha^i, \alpha^i]$.
- Otherwise, the length of $\tilde{S}$ is larger than ℓ. Then, at time $\alpha^{i-1} + \ell = \beta \cdot \alpha^i$, the server still executes $\tilde{S}$, is at distance at least $(1/3) \cdot (\alpha - 1 + 5c) \cdot \alpha^{i-2}$ from the origin and it travels away from the origin with unit speed. Within time interval $[\beta \cdot \alpha^i, \xi \cdot \alpha^i]$, the server either finishes executing $\tilde{S}$ or it increases its distance to the origin by $\xi \cdot \alpha^i - \beta \cdot \alpha^i$. In the former case, the lemma follows trivially, and in the latter case, the server distance to the origin at time $\xi \cdot \alpha^i$ is at least

$$\begin{aligned}\frac{\alpha - 1 + 5c}{3} \cdot \alpha^{i-2} + (\xi - \beta) \cdot \alpha^i &= \frac{5 \cdot \beta \cdot \alpha^2 - 9\alpha - 1}{3} \cdot \alpha^{i-2} + (\xi - \beta) \cdot \alpha^i \\ &= \tau_\beta(\xi) \cdot \alpha^{i-1} .\end{aligned}$$

  From time $\xi \cdot \alpha^i$, the server continues to travel away from the origin, and thus the lemma follows. ◀

4.1 Relating Reroute to Opt

We analyze the performance of algorithm $\textsc{Reroute}(\beta)$ for a fixed parameter β, such that $2/\alpha \leq \beta \leq 1$. Moreover, we choose β, so that $\tau_\beta(\beta) \geq 1/\alpha$. In our analysis we use a parameter ξ, such that $\beta \leq \xi \leq 1$ and $\tau_\beta(\beta) \leq \tau_\beta(\xi) \leq 1$. Concrete values of β and ξ will be fixed later.

Let L be the index of the last phase in which $\textsc{Opt}$ services a request. On the basis of β and ξ, we partition both $\textsc{Alg}_i$ (the total weight of requests serviced in phase $i \in \{1, \dots, L+1\}$ by $\textsc{Reroute}$) and $\textsc{Opt}_i$ (the total weight of requests serviced in phase $i \in \{0, \dots, L\}$ by $\textsc{Opt}$) into three parts:

- A_i^a: the weight serviced by $\textsc{Reroute}(\beta)$ in time interval $(\alpha^{i-1}, \beta \cdot \alpha^i]$,
- A_i^b: the weight serviced by $\textsc{Reroute}(\beta)$ in time interval $(\beta \cdot \alpha^i, \xi \cdot \alpha^i]$,
- A_i^c: the weight serviced by $\textsc{Reroute}(\beta)$ in time interval $(\xi \cdot \alpha^i, \alpha^i]$,
- O_i^a: the weight serviced by $\textsc{Opt}$ in time interval $[\alpha^{i-1}, \tau_\beta(\beta) \cdot \alpha^i)$,
- O_i^b: the weight serviced by $\textsc{Opt}$ in time interval $[\tau_\beta(\beta) \cdot \alpha^i, \tau_\beta(\xi) \cdot \alpha^i)$,
- O_i^c: the weight serviced by $\textsc{Opt}$ in time interval $[\tau_\beta(\xi) \cdot \alpha^i, \alpha^i)$.

Note that the validity of this partitioning requires that $1/\alpha \leq \beta \leq \xi \leq 1$ and $1/\alpha \leq \tau_\beta(\beta) \leq \tau_\beta(\xi) \leq 1$. We slightly modify the definition of A_0^a and O_L^c to include also the initial and final time points, i.e., to be the weight serviced by $\textsc{Reroute}(\beta)$ in $[\alpha^0, \beta \cdot \alpha^1]$ and the weight serviced by $\textsc{Opt}$ in $[\tau_\beta(\xi) \cdot \alpha^L, \alpha^L]$, respectively. Clearly, $\textsc{Alg}_i = \mathrm{A}_i^a + \mathrm{A}_i^b + \mathrm{A}_i^c$ and $\textsc{Opt}_i = \mathrm{O}_i^a + \mathrm{O}_i^b + \mathrm{O}_i^c$.

As $\textsc{Reroute}$ services the same set of requests as $\textsc{RetF}$, the guarantee of Lemma 2 applies to the schedule produced by $\textsc{Reroute}(\beta)$ as well. In particular, $\textsc{Reroute}(\beta)$ finishes servicing all requests till the end of phase $L + 1$ (it does not service anything in phase 0) and for any phase $j \in \{1, \dots, L + 1\}$, it holds that

$$\sum_{i=j}^{L+1} \left(\mathrm{A}_i^a + \mathrm{A}_i^b + \mathrm{A}_i^c\right) \leq \sum_{i=j-1}^{L} \left(\mathrm{O}_i^a + \mathrm{O}_i^b + \mathrm{O}_i^c\right) . \tag{5}$$

Fix any phase $j \in \{1, \ldots, L+1\}$ and consider all requests contributing to the sum $\sum_{i=j}^{L+1}(\mathrm{A}_i^b + \mathrm{A}_i^c)$. By Lemma 4, each such request has to be serviced by Opt at time $\tau_\beta(\beta) \cdot \alpha^{j-1}$ or later (because its distance to the origin is at least $\tau_\beta(\beta) \cdot \alpha^{j-1}$). Thus,

$$\sum_{i=j}^{L+1} \left(\mathrm{A}_i^b + \mathrm{A}_i^c\right) \leq \mathrm{O}_{j-1}^b + \mathrm{O}_{j-1}^c + \sum_{i=j}^{L} \left(\mathrm{O}_i^a + \mathrm{O}_i^b + \mathrm{O}_i^c\right) . \tag{6}$$

Similarly, consider all requests contributing to the sum $\mathrm{A}_j^c + \sum_{i=j+1}^{L+1}(\mathrm{A}_i^b + \mathrm{A}_i^c)$. Again, by Lemma 4, each such request has to be serviced by Opt at time $\tau_\beta(\xi) \cdot \alpha^{j-1}$ or later (because its distance to the origin is at least $\tau_\beta(\xi) \cdot \alpha^{j-1}$). Hence,

$$\mathrm{A}_j^c + \sum_{i=j+1}^{L+1} \left(\mathrm{A}_i^b + \mathrm{A}_i^c\right) \leq \mathrm{O}_{j-1}^c + \sum_{i=j}^{L} \left(\mathrm{O}_i^a + \mathrm{O}_i^b + \mathrm{O}_i^c\right) . \tag{7}$$

Finally, observe that the total cost of Reroute(β) can be upper-bounded by $\sum_{i=1}^{L+1}(\beta \cdot \alpha^i \cdot \mathrm{A}_i^a + \xi \cdot \alpha^i \cdot \mathrm{A}_i^b + \alpha^i \cdot \mathrm{A}_i^c)$ and the cost of Opt can be lower-bounded by $\sum_{i=0}^{L}(\alpha^{i-1} \cdot \mathrm{O}_i^a + \tau_\beta(\beta) \cdot \alpha^i \cdot \mathrm{O}_i^b + \tau_\beta(\xi) \cdot \alpha^i \cdot \mathrm{O}_i^c)$.

4.2 Factor-revealing LP

Let us now consider what happens when an adversary constructs an instance $\mathcal{I}$ for the algorithm Reroute. When Opt is run on $\mathcal{I}$, this defines L as the index of the last phase when Opt services a request and this also defines non-negative values of variables $\mathrm{O}_i^a, \mathrm{O}_i^b, \mathrm{O}_i^c$ for $i \in \{0, \ldots, L\}$. When Reroute(β) is run on $\mathcal{I}$, this defines non-negative values of variables $\mathrm{A}_i^a, \mathrm{A}_i^b, \mathrm{A}_i^c$ for $i \in \{1, \ldots, L+1\}$. As shown above, these variables satisfy inequalities (5), (6) and (7). Moreover, the goal of the adversary is to maximize the ratio between the cost of Reroute(β) and the cost of Opt.

This maximization problem may become only easier for the adversary if instead of creating an actual input sequence, the adversary simply chooses L and the non-negative values of variables $\mathrm{A}_i^a, \mathrm{A}_i^b, \mathrm{A}_i^c$, $\mathrm{O}_i^a, \mathrm{O}_i^b, \mathrm{O}_i^c$ for $i \in \{0, \ldots, L\}$, satisfying inequalities (5), (6) and (7), so to maximize the objective value of Reroute-to-Opt cost ratio.

The values of all variables can be multiplied by a fixed value without changing the objective value. Thus, instead of maximizing the cost ratio, the adversary may maximize total cost of Reroute(β) with an additional constraint ensuring that the total cost of Opt is at most 1.

This leads to the following factor-revealing linear program $\mathcal{P}(L, \beta, \xi)$, whose optimal value $P^*(L, \beta, \xi)$ is an upper bound on the competitive ratio of Reroute(β) for a given L. This relation holds for any choice of parameter $\xi \in [\beta, 1]$. The goal of $\mathcal{P}(L, \beta, \xi)$ is to maximize

$$\sum_{i=1}^{L+1} \left(\beta \cdot \alpha^i \cdot \mathrm{A}_i^a + \xi \cdot \alpha^i \cdot \mathrm{A}_i^b + \alpha^i \cdot \mathrm{A}_i^c\right)$$

subject to the following constraints

$$\sum_{i=j}^{L+1} \left(\mathrm{A}_i^a + \mathrm{A}_i^b + \mathrm{A}_i^c\right) \leq \sum_{i=j-1}^{L} \left(\mathrm{O}_i^a + \mathrm{O}_i^b + \mathrm{O}_i^c\right) \qquad \text{for all } j \in \{1,\dots,L+1\}$$

$$\sum_{i=j}^{L+1} \left(\mathrm{A}_i^b + \mathrm{A}_i^c\right) \leq \mathrm{O}_{j-1}^b + \mathrm{O}_{j-1}^c + \sum_{i=j}^{L} \left(\mathrm{O}_i^a + \mathrm{O}_i^b + \mathrm{O}_i^c\right) \qquad \text{for all } j \in \{1,\dots,L+1\}$$

$$\mathrm{A}_j^c + \sum_{i=j+1}^{L+1} \left(\mathrm{A}_i^b + \mathrm{A}_i^c\right) \leq \mathrm{O}_{j-1}^c + \sum_{i=j}^{L} \left(\mathrm{O}_i^a + \mathrm{O}_i^b + \mathrm{O}_i^c\right) \qquad \text{for all } j \in \{1,\dots,L+1\}$$

$$\sum_{i=0}^{L} \left(\alpha^{i-1} \cdot \mathrm{O}_i^a + \tau_\beta(\beta) \cdot \alpha^i \cdot \mathrm{O}_i^b + \tau_\beta(\xi) \cdot \alpha^i \cdot \mathrm{O}_i^c\right) \leq 1$$

and non-negativity of the variables. Note that the left hand side of the last inequality is a lower bound on the cost of Opt

It remains to upper-bound the value of $P^*(L, \beta, \xi)$. Such upper bound is given by the value of any feasible solution to the dual program $\mathcal{D}(L, \beta, \xi)$. The goal of $\mathcal{D}(L, \beta, \xi)$ is to minimize $\mathcal{R}$ subject to the following constraints

$$\sum_{i=1}^{j} q_i^a \geq \beta \cdot \alpha^j \qquad \text{for all } j \in \{1,\dots,L+1\} \tag{8}$$

$$\sum_{i=1}^{j-1} \left(q_i^a + q_i^b + q_i^c\right) + q_j^a + q_j^b \geq \xi \cdot \alpha^j \qquad \text{for all } j \in \{1,\dots,L+1\} \tag{9}$$

$$\sum_{i=1}^{j} \left(q_i^a + q_i^b + q_i^c\right) \geq \alpha^j \qquad \text{for all } j \in \{1,\dots,L+1\} \tag{10}$$

$$\alpha^{j-1} \cdot \mathcal{R} \geq \sum_{i=1}^{j} \left(q_i^a + q_i^b + q_i^c\right) + q_{j+1}^a \qquad \text{for all } j \in \{0,\dots,L\} \tag{11}$$

$$\tau_\beta(\beta) \cdot \alpha^j \cdot \mathcal{R} \geq \sum_{i=1}^{j} \left(q_i^a + q_i^b + q_i^c\right) + q_{j+1}^a + q_{j+1}^b \qquad \text{for all } j \in \{0,\dots,L\} \tag{12}$$

$$\tau_\beta(\xi) \cdot \alpha^j \cdot \mathcal{R} \geq \sum_{i=1}^{j+1} \left(q_i^a + q_i^b + q_i^c\right) \qquad \text{for all } j \in \{0,\dots,L\} \tag{13}$$

and non-negativity of the variables. Note that sets of inequalities (8), (9), (10), (11), (12) and (13) correspond to sets of variables A_i^a, A_i^b, A_i^c, O_i^a, O_i^b and O_i^c, respectively, in the primal program $\mathcal{P}(L, \beta, \xi)$.

▶ **Lemma 5.** *There exist values β and ξ, such that for any L, there exists a feasible solution to $\mathcal{D}(L, \beta, \xi)$ whose value is at most $2\sqrt{2} + 13/5 < 5.429$.*

Proof. We choose

$$\beta = (9\alpha + 4)/(5\alpha^2) = (3 + \sqrt{2})/5 \approx 0.883 \text{ and}$$
$$\xi = \beta + (1 - \beta)/\alpha = (4\sqrt{2} - 1)/5 \approx 0.931\,.$$

For these values of β and ξ, it holds that

$$\tau_\beta(\beta) = 1/\alpha = \sqrt{2} - 1 \approx 0.414 \text{ and}$$
$$\tau_\beta(\xi) = \tau_\beta(\beta) + (\xi - \beta) \cdot \alpha = 1/\alpha + (2 - \sqrt{2})/5 = (5 + \sqrt{2})/(5 + 5\sqrt{2}) \approx 0.531\,.$$

We set the dual variables as follows:

$$\begin{aligned}
&q_1^a = \beta \cdot \alpha\,, && \\
&q_1^b = (\xi - \beta) \cdot \alpha\,, && \\
&q_1^c = (1 - \xi) \cdot \alpha\,, && \\
&q_j^a = \beta \cdot (\alpha - 1) \cdot \alpha^{j-1} && \text{for } j \in \{2, \dots, L+1\}\,, \\
&q_j^b = 0 && \text{for } j \in \{2, \dots, L+1\}\,, \\
&q_j^c = (1 - \beta) \cdot (\alpha - 1) \cdot \alpha^{j-1} && \text{for } j \in \{2, \dots, L+1\}\,.
\end{aligned}$$

Our choice of β, ξ and dual variables satisfy (8), (9) and (10) conditions of $\mathcal{D}(L, \beta, \xi)$ with equality. Actually, (8), (10) and (9) for $j = 1$ hold with equality for any choice of β and ξ. For $j \in \{2, \dots, L+1\}$, the left hand side of (9) is equal to $\alpha^{j-1} + \beta \cdot (\alpha - 1) \cdot \alpha^{j-1} = (\beta \cdot \alpha + 1 - \beta) \cdot \alpha^{j-1}$, and the right hand side is equal to $\xi \cdot \alpha^j$; these values coincide for $\xi = \beta + (1 - \beta)/\alpha$.

Given the fixed values of the dual variables, we choose $\mathcal{R}$ as the minimum value satisfying inequalities (11), (12) and (13). Substituting the chosen values of the dual variables in these inequalities and using $\tau_\beta(\beta) = 1/\alpha$, yields

$$\begin{aligned}
&\mathcal{R} \geq (\beta \cdot \alpha)/\alpha^{-1} = \beta \cdot \alpha^2 && \text{by (11) for } j = 0\,, \\
&\mathcal{R} \geq (\alpha^j + \beta \cdot (\alpha - 1) \cdot \alpha^j)/\alpha^{j-1} = \beta \cdot \alpha^2 + \alpha - \beta \cdot \alpha && \text{by (11) for } j \geq 1\,, \\
&\mathcal{R} \geq (\beta \cdot \alpha + (\xi - \beta) \cdot \alpha)/\tau_\beta(\beta) = \xi \cdot \alpha^2 && \text{by (12) for } j = 0\,, \\
&\mathcal{R} \geq (1 + \beta \cdot (\alpha - 1)) \cdot \alpha^j/(\tau_\beta(\beta) \cdot \alpha^j) = \beta \cdot \alpha^2 + \alpha - \beta \cdot \alpha && \text{by (12) for } j \geq 1\,, \\
&\mathcal{R} \geq \alpha^{j+1}/(\tau_\beta(\xi) \cdot \alpha^j) = \alpha/\tau_\beta(\xi) && \text{by (13) for } j \geq 0\,.
\end{aligned}$$

Thus, using $\beta \leq \xi$ and $\xi = \beta + (1 - \beta)/\alpha$, we obtain

$$\begin{aligned}
\mathcal{R} &= \max\left\{\beta \cdot \alpha^2,\ \beta \cdot \alpha^2 + \alpha - \beta \cdot \alpha,\ \xi \cdot \alpha^2,\ \frac{\alpha}{\tau_\beta(\xi)}\right\} \\
&= \max\left\{\xi \cdot \alpha^2,\ \frac{\alpha}{\tau_\beta(\xi)}\right\} \\
&= \max\left\{2\sqrt{2} + 13/5,\ (15 + 10\sqrt{2})/(5 + \sqrt{2})\right\} \\
&= 2\sqrt{2} + 13/5 < 5.429\,,
\end{aligned}$$

which concludes the proof. ◀

▶ **Theorem 6.** *For $\beta = (3 + \sqrt{2})/5 \approx 0.883$, the competitive ratio of* Reroute(β) *for the traveling repairperson problem is at most $2\sqrt{2} + 13/5 < 5.429$.*

Proof. Fix any input sequence $\mathcal{I}$, run Opt on $\mathcal{I}$, and partition its execution into phases. Let L be the index of the last phase in which Opt services a request.

Let $\xi = \beta + (1 - \beta)/\alpha$. As discussed above, the competitive ratio of Reroute(β) is upper-bounded by the optimal value $P^*(L, \beta, \xi)$ of the maximization program $\mathcal{P}(L, \beta, \xi)$. By weak duality, the feasible solution to the dual minimization program $\mathcal{D}(L, \beta, \xi)$ of value $2\sqrt{2} + 13/5$ proposed in Lemma 5 is an upper bound on the optimal primal solution $P^*(L, \beta, \xi)$. Hence, $2\sqrt{2} + 13/5$ is an upper bound on the competitive ratio of Reroute(β). ◀

5 Final remarks

Our computer-based experiments show that further partitioning of phases into more than three intervals does not lead to an improvement of the competitive ratio.

Furthermore, it is possible to show that our solution to the dual program is asymptotically best possible and the ratio cannot be improved by simply choosing better parameters β and ξ. That is, with growing L, the optimal value of the dual $\mathcal{D}(L, \beta, \xi)$ converges to the value $2\sqrt{2} + 13/5$ given by Lemma 5, as demonstrated in the lemma below.

▶ **Lemma 7.** *Fix any L. For any values of β and ξ, satisfying $1/\alpha \le \beta \le \xi \le 1$ and $1/\alpha \le \tau_\beta(\beta) \le \tau_\beta(\xi) \le 1$, the value of any feasible solution to the dual program $\mathcal{D}(L, \beta, \xi)$ is at least $(2\sqrt{2} + 13/5)/(1 + \alpha^{-L})$.*

Proof. Fix any $j \in \{1, \dots, L\}$. By combining requirement (11) with (10), we obtain

$$\begin{aligned}(\mathcal{R} - \alpha) \cdot \alpha^{j-1} &= \mathcal{R} \cdot \alpha^{j-1} - \alpha^j \\ &\ge \sum_{i=1}^{j} \left(q_i^a + q_i^b + q_i^c\right) + q_{j+1}^a - \sum_{i=1}^{j} \left(q_i^a + q_i^b + q_i^c\right) \\ &= q_{j+1}^a \,.\end{aligned}$$

Summing this relation over all $j \in \{1, \dots, L\}$ and using (8) yields

$$(\mathcal{R} - \alpha) \cdot \frac{\alpha^L - 1}{\alpha - 1} \ge \sum_{j=1}^{L} q_{j+1}^a = -q_1^a + \sum_{j=1}^{L+1} q_j^a \ge \beta \cdot \alpha^{L+1} - q_1^a \,.$$

Now we observe that (11) for $j = 1$ implies $q_1^a \le \mathcal{R}/\alpha < \mathcal{R}/(\alpha - 1)$. By substituting this in the inequality above and multiplying both sides by $\alpha - 1$, we get

$$(\mathcal{R} - \alpha) \cdot (\alpha^L - 1) \ge \beta \cdot (\alpha - 1) \cdot \alpha^{L+1} - \mathcal{R} \,.$$

Finally, we divide both sides by α^L, obtaining

$$\mathcal{R} - \alpha \ge (\mathcal{R} - \alpha) \cdot (\alpha^L - 1)/\alpha^L \ge \beta \cdot (\alpha - 1) \cdot \alpha - \mathcal{R}/\alpha^L \,,$$

and therefore,

$$\mathcal{R} \cdot (1 + \alpha^{-L}) \ge \beta \cdot \alpha^2 + \alpha - \beta \cdot \alpha \,.$$

As in our construction we require $\tau_\beta(\beta) \ge 1/\alpha$, it holds that $\beta \ge (9\alpha + 4)/(5\alpha^2) = (3 + \sqrt{2})/5$. Combining this bound with the value of $\alpha = 1 + \sqrt{2}$ yields

$$\mathcal{R} \ge (\beta \cdot \alpha^2 + \alpha - \beta \cdot \alpha)/(1 + \alpha^{-L}) \ge (2\sqrt{2} + 13/5)/(1 + \alpha^{-L}) \,.$$ ◀

References

1 Foto N. Afrati, Stavros S. Cosmadakis, Christos H. Papadimitriou, George Papageorgiou, and Nadia Papakostantinou. The Complexity of the Travelling Repairman Problem. *Informat. Theor. Appl.*, 20(1):79–87, 1986. doi:10.1051/ita/1986200100791.

2 Aaron Archer and Anna Blasiak. Improved Approximation Algorithms for the Minimum Latency Problem via Prize-Collecting Strolls. In *Proc. 21st ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 429–447, 2010. doi:10.1137/1.9781611973075.36.

3 Norbert Ascheuer, Sven Oliver Krumke, and Jörg Rambau. Online Dial-a-Ride Problems: Minimizing the Completion Time. In *Proc. 17th Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 639–650, 2000. doi:10.1007/s10951-005-6811-3.

4 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Serving Requests with On-line Routing. In *Proc. 4th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 37–48, 1994. doi:10.1007/3-540-58218-5_4.

5 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Competitive Algorithms for the On-line Traveling Salesman. In *Proc. 4th Int. Workshop on Algorithms and Data Structures (WADS)*, pages 206–217, 1995. doi:10.1007/3-540-60220-8_63.

6 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the On-Line Travelling Salesman. *Algorithmica*, 29(4):560–581, 2001. doi:10.1007/s004530010071.

7 Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the Plane. *Information and Computation*, 106(2):234–252, 1993. doi:10.1006/inco.1993.1054.

8 Alexander Birx and Yann Disser. Tight Analysis of the Smartstart Algorithm for Online Dial-a-Ride on the Line. In *Proc. 36th Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 15:1–15:17, 2019. doi:10.4230/LIPIcs.STACS.2019.15.

9 Antje Bjelde, Yann Disser, Jan Hackfeld, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Kevin Schewior, Miriam Schlöter, and Leen Stougie. Tight Bounds for Online TSP on the Line. In *Proc. 28th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 994–1005, 2017. doi:10.1137/1.9781611974782.63.

10 Vincenzo Bonifaci and Leen Stougie. Online k-Server Routing Problems. *Theory of Computing Systems*, 45(3):470–485, 2009. doi:10.1007/s00224-008-9103-4.

11 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

12 Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, Trees, and Minimum Latency Tours. In *Proc. 44th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 36–45, 2003. doi:10.1109/SFCS.2003.1238179.

13 Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001. doi:10.1016/S0304-3975(00)00261-9.

14 Irene Fink, Sven Oliver Krumke, and Stephan Westphal. New lower bounds for online k-server routing problems. *Information Processing Letters*, 109(11):563–567, 2009. doi:10.1016/j.ipl.2009.01.024.

15 Michel X. Goemans and Jon M. Kleinberg. An improved approximation ratio for the minimum latency problem. *Math. Program.*, 82:111–124, 1998. doi:10.1007/BF01585867.

16 Dietrich Hauptmeier, Sven Oliver Krumke, and Jörg Rambau. The Online Dial-a-Ride Problem under Reasonable Load. In *Proc. 4th Int. Conf. on Algorithms and Complexity (CIAC)*, pages 125–136, 2000. doi:10.1007/3-540-46521-9_11.

17 Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On Minimizing the Maximum Flow Time in the Online Dial-a-Ride Problem. In *Proc. 3rd Workshop on Approximation and Online Algorithms (WAOA)*, pages 258–269, 2005. doi:10.1007/11671411_20.

18 Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, and Leen Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295:279–294, 2003. doi:10.1016/S0304-3975(02)00409-7.

19 Sven Oliver Krumke, Luigi Laura, Maarten Lipmann, Alberto Marchetti-Spaccamela, Willem de Paepe, Diana Poensgen, and Leen Stougie. Non-abusiveness Helps: An O(1)-Competitive Algorithm for Minimizing the Maximum Flow Time in the Online Traveling Salesman Problem. In *Proc. 5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 200–214, 2002. doi:10.1007/3-540-45753-4_18.

20 René Sitters. The Minimum Latency Problem Is NP-Hard for Weighted Trees. In *Proc. 9th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, pages 230–239, 2002. `doi:10.1007/3-540-47867-1_17`.

21 René Sitters. Polynomial time approximation schemes for the traveling repairman and other minimum latency problems. In *Proc. 25th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 604–616, 2014. `doi:10.1137/1.9781611973402.46`.

Query-Competitive Sorting with Uncertainty

Magnús M. Halldórsson
ICE-TCS, Department of Computer Science, Reykjavik University, Iceland
mmh@ru.is

Murilo Santos de Lima[1]
ICE-TCS, Department of Computer Science, Reykjavik University, Iceland
mslima@ic.unicamp.br

Abstract

We study the problem of sorting under incomplete information, when queries are used to resolve uncertainties. Each of n data items has an unknown value, which is known to lie in a given interval. We can pay a query cost to learn the actual value, and we may allow an error threshold in the sorting. The goal is to find a nearly-sorted permutation by performing a minimum-cost set of queries.

We show that an offline optimum query set can be found in polynomial time, and that both oblivious and adaptive problems have simple query-competitive algorithms. The query-competitiveness for the oblivious problem is n for uniform query costs, and unbounded for arbitrary costs; for the adaptive problem, the ratio is 2.

We then present a unified adaptive strategy for uniform query costs that yields: (i) a 3/2-query-competitive randomized algorithm; (ii) a 5/3-query-competitive deterministic algorithm if the dependency graph has no 2-components after some preprocessing, which has query-competitive ratio $3/2 + \mathrm{O}(1/k)$ if the components obtained have size at least k; (iii) an exact algorithm if the intervals constitute a laminar family. The first two results have matching lower bounds, and we have a lower bound of 7/5 for large components.

We also show that the advice complexity of the adaptive problem is $\lfloor n/2 \rfloor$ if no error threshold is allowed, and $\lceil n/3 \cdot \lg 3 \rceil$ for the general case.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Discrete mathematics; Theory of computation → Theory and algorithms for application domains

Keywords and phrases online algorithms, sorting, randomized algorithms, advice complexity, threshold tolerance graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.7

Funding Partially supported by Icelandic Research Fund grant 174484-051.

1 Introduction

Sorting is one of the most fundamental problems in computer science and an essential part of any system dealing with large amounts of data. High-performance algorithms such as QuickSort [19] have been known for decades, but the demand for fast sorting of huge amounts of data is such that improvements in sorting algorithms are still an active area of research; see, e.g., [26].

In a distributed application with dynamic data, it may not be feasible to maintain a precise copy of the information in each replica. In particular, to access a local cached information may be much cheaper, even though not as precise, than to query a master database or to run a distributed consensus algorithm. One approach is to maintain in the replicas, for each data item, an interval that bounds the actual value. These intervals can be updated much faster than to guarantee a strict consistency of the data. When higher

[1] Corresponding author.

precision is required, the system can query the master database for a more fine-grained interval or for the actual data value. Therefore a trade-off between data precision and system performance can be established. The TRAPP system, proposed by Olston and Widom [24], relies on this concept.

This idea has led to theoretical investigation on **uncertainty problems with queries** [7, 11, 12, 13, 14, 17, 22]. Such problems also appear in optimization scenarios in which an extra effort can be incurred in order to obtain more precise values of the input data, such as by investing in market research, which is expensive so its cost should be minimized. These works build upon more established frameworks of optimization with uncertainty, such as online [5], robust [3] and stochastic [4] optimization. In particular, the analysis of algorithms in terms of competitiveness against an adversary is inherited from the online optimization literature.

In this paper, we investigate the problem of sorting data items whose actual values are unknown, but for which we are given intervals on which the actual values lie. We can **query** an interval and then learn the actual value of the corresponding data item, but this incurs some cost. The goal, then, is to sort the items by performing a set of queries of minimum cost. Furthermore, the precision in the sorting may be relaxed, so that inversions may occur if the actual values are not too far apart.

We distinguish between two types of algorithms for uncertainty problems with queries. An **adaptive** algorithm may decide which queries to perform based on results from previous queries. An **oblivious** algorithm, however, must choose the whole set of queries to perform in advance; i.e., it must choose a set of queries that certainly allow the problem to be solved without any knowledge of the actual values. In this paper, both algorithms are compared with an offline optimum query set, i.e., a minimum-cost set of queries that proves the obtained solution to be correct.[2] An algorithm (either adaptive or oblivious) is **α-query-competitive** if it performs a total query cost of at most α times the cost of an offline optimum query set.

Another related problem is that of finding an **optimum query set**. Here we are given the actual data values, and want to identify a minimum-cost set of queries that would be sufficient to prove that the solution is correct. Solving this problem is useful, for example, to perform experimental evaluation of online algorithms, since we are actually finding the offline optimum solution for the uncertainty problem. This is also called the **verification version** of the corresponding uncertainty problem with queries [8, 11].

We are also interested in the **advice complexity** of the problems we study. In this setting, an online algorithm has access to an oracle that can give helpful information when making decisions. The advice complexity is the number of bits of advice that are sufficient and necessary for an online algorithm to solve the problem exactly. This is a research topic that has gained substantial attention; see [6] for a survey.

Our contribution. We begin by showing how to compute an optimum query set in polynomial time, and that both oblivious and adaptive problems have simple algorithms with matching deterministic lower bounds. The query-competitive ratio of the oblivious problem is n if we have uniform query costs, and unbounded for arbitrary costs; for the adaptive problem, the query-competitive ratio is 2. The optimal oblivious algorithm is trivial; for the

[2] This nomenclature differs to that used by Feder et al. [14]. They call an adaptive algorithm an **online** algorithm, and an oblivious algorithm an **offline** algorithm. We disagree with this nomenclature, since both types of algorithms are online in the standard sense of not knowing the data. Also, they compare an oblivious algorithm to an optimal oblivious strategy, and not to an offline optimum query set.

adaptive case, we have a simpler algorithm for uniform query costs, and a more sophisticated one for arbitrary query costs. If query costs are uniform and the error threshold is zero, then the simpler algorithm can be implemented as an oracle for any comparison-based sorting algorithm, preserving time complexity and stability.

Those results are rather simple and it seems like the query-competitiveness of the problem is settled. However, we present a unified adaptive strategy that attains different improvements for uniform query costs. First, we obtain a 3/2-query-competitive algorithm by using randomization. Second, if the error threshold is zero, and after some preprocessing the dependency graph has no 2-components, the strategy yields a deterministic 5/3-query-competitive algorithm; if the obtained graph has components of size at least k, then the same algorithm has query-competitive ratio $3/2 + \mathrm{O}(1/k)$. The first two results have a matching lower bound, and for large components we have a lower bound of 7/5. The problem can also be solved exactly if the intervals constitute a laminar family.

Finally, we show that the advice complexity for adaptive algorithms is exactly $\lfloor n/2 \rfloor$ bits if there is no error threshold, and exactly $\lceil n/3 \cdot \lg 3 \rceil$ bits for the general case.

Related work. The first work to investigate the minimum number of queries to solve a problem is by Kahan [20], who showed optimal oblivious strategies to find the minimum, maximum and median of n values in uncertainty intervals.

Olston and Widom [24] proposed the TRAPP system, a distributed database based on uncertainty intervals. The authors: (1) gave an optimal oblivious strategy for finding the minimum (and equivalently, the maximum) of a sequence of values within an error bound; (2) showed that it is NP-hard to find an optimum oblivious query set to compute the sum of a sequence of values within an error bound, with a reduction from the knapsack problem. The paper also discusses strategies for counting and finding the average of a sequence of values. Khanna and Tan [21] generalized these results for arbitrary query costs and different levels of precision.

Feder et al. [14] considered the uncertainty version of the problem of finding the k-th largest value on a sequence (i.e., the generalized median problem). The authors presented optimal oblivious and adaptive strategies for the problem, both running in polynomial time. Both strategies are optimal, and the ratio between the oblivious and the adaptive strategy (also called the **price of obliviousness**) is $\frac{2k-1}{k} < 2$ for uniform query costs, and k for arbitrary query costs.[3]

Bruce et al. [7] studied geometric problems where the points are given in uncertainty areas. The authors gave 3-query-competitive algorithms for finding the maximal points and the convex hull in a two-dimensional space. They also proposed the concept of **witness sets**, which has been used subsequently in various works on uncertainty problems with queries. Charalambous and Hoffman [8] showed that it is NP-hard to find an optimum query set for the maximal points problem.

Feder et al. [13] studied the uncertainty variant of the shortest path problem. They showed that to solve optimally the oblivious version of the problem is neither NP nor co-NP, unless NP = co-NP. Their paper also discusses the complexity of the problem for various particular cases.

Erlebach et al. [12] proved that the minimum spanning tree problem with uncertainty admits an adaptive 2-query-competitive algorithm, which is the best possible for a deterministic algorithm. Erlebach, Hoffmann and Kammer [11] studied a generalization called the

[3] The works cited up to this point do not evaluate the algorithms using the competitiveness framework.

cheapest set problem, for which there is an adaptive algorithm with at most $d \cdot \mathrm{opt} + d$ queries, where d is the maximum cardinality of a set. They also generalized the result in the previous work to obtain an adaptive 2-query-competitive algorithm for the problem of finding a minimum-weight base on a matroid.

Gupta, Sabharwal and Sen [18] studied various of the previous problems in the setting where a query may return a refined interval, instead of the exact value of the data item.

Megow, Meißner and Skutella [22] improved the result for the minimum spanning tree problem with a randomized adaptive algorithm, obtaining query-competitive ratio 1.7. (The problem has lower bound 1.5 for randomized algorithms.) They also considered non-uniform query costs and proved that their results can be extended to find a minimum-weight base on a matroid. Furthermore, they showed that an optimum query set and the actual value of the minimum spanning tree can be computed in polynomial time. Some experimental evaluation of those algorithms were presented in [15].

Ryzhov and Powell [25] investigated how to solve a linear program while minimizing the query cost when the coefficients of the objective function are uncertain. They presented a policy which is asymptotically optimal. Yamaguchi and Maehara [28] studied the variant with packing constraints and coefficients following a probability distribution, and showed how to apply this to stochastic problems such as matching, matroid and stable set problems.

Note that all the work cited so far deals with problems whose classical (offline) versions can be solved in polynomial time. Uncertainty versions with queries have been proposed for the knapsack problem [17] and the scheduling problem [2, 9]. Since those problems are NP-hard, we might include the query cost into the solution cost and look for a competitive algorithm if we are looking for a polynomial-time algorithm. Another option is to limit the maximum number of queries performed, and then to try to optimize the solution cost.

For a survey on the topic, see [10]. Other references for related problems are also cited in [11].

Another sorting problem with uncertainty was studied by Ajtain et al. [1]. In that problem, the values to be sorted are unknown, but their relative order can be tested by a comparison procedure. However, comparing values that are too close returns imprecise answers, so in principle we should compare all $\binom{n}{2}$ pairs to obtain a sorting with some error guarantee. The authors show how to solve the problem using only $\mathrm{O}(n^{3/2})$ comparisons.

Organization of the paper. In Section 2, we present the sorting problem with uncertainty and some basic facts, and in Section 3 we give algorithms to find an offline optimum query set and for the oblivious setting. We treat deterministic adaptive algorithms in Section 4. In Section 5, we show how to improve the adaptive result for uniform query costs by using a randomized algorithm, or by assuming some structure in the dependency graph. We investigate the advice complexity for adaptive algorithms in Section 6, and finally, in Section 7, we discuss possible future research directions.

2 Sorting with Uncertainty

In the sorting problem with uncertainty, there are n numbers $v_1, \ldots, v_n \in \mathbb{R}$ whose exact value is unknown. We are given n uncertainty intervals $I_1, \ldots, I_n$ with $v_i \in I_i = [\ell_i, r_i]$, a cost $w_i \in \mathbb{R}_+$ for querying interval I_i, and an error threshold $\delta \geq 0$. After querying I_i, we obtain the exact value of v_i; we can also say that we replace I_i with interval $I'_i = [v_i, v_i]$. The goal is to obtain a permutation $\pi : [n] \to [n]$ such that $v_i \leq v_j + \delta$ if $\pi(i) < \pi(j)$ by performing a minimum-cost set of queries.

We begin by defining the following dependency relation between intervals, which is essential to solve the problem.

▶ **Definition 2.1.** *Two intervals I_i and I_j such that $r_i - \ell_j > \delta$ and $r_j - \ell_i > \delta$ are* ***dependent****. Two intervals that are not dependent are* ***independent****.*

▶ **Lemma 2.2.** *The relative order between two intervals can be decided without querying either of them if and only if they are independent.*

Proof. Let I_i and I_j be such that $r_i - \ell_j \leq \delta$. Since $v_i \leq r_i$ and $v_j \geq \ell_j$, we have that $v_i \leq v_j + \delta$ and we can set $\pi(i) < \pi(j)$ without querying both I_i and I_j.

Conversely, let I_j and I_j be two dependent intervals. We cannot set $\pi(i) < \pi(j)$, because it may be the case that $v_i = r_i$ and $v_j = \ell_j$, thus $r_i - \ell_j > \delta$ implies that $v_i > v_j + \delta$. By a symmetric argument, we cannot set $\pi(j) < \pi(i)$, so we cannot decide the relative order between the intervals. ◀

The graphs defined by this dependency relation are exactly the **co-threshold tolerance** (co-TT) graphs [23]. $G = (V, E)$ is a **threshold tolerance** graph if there are functions $w : V \to \mathbb{R}$ and $t : V \to \mathbb{R}$ such that $uv \in E$ if and only if $w(u) + w(v) \geq \min(t(u), t(v))$. A co-TT graph is the complement of any threshold tolerance graph, or equivalently, $G = (V, E)$ is a co-TT graph if and only if there are functions $a : V \to \mathbb{R}$ and $b : V \to \mathbb{R}$ such that $uv \in E$ if and only if $a(u) < b(v)$ and $a(v) < b(u)$ [23]. The proof of Theorem 2.3 is omitted.

▶ **Theorem 2.3.** *The graphs defined by the dependency relation in Definition 2.1 are exactly the co-TT graphs.*

The following result will be useful.

▶ **Lemma 2.4** ([23]). *Every co-TT graph is chordal.*

When $\delta > 0$, it is useful to distinguish intervals of width smaller than δ, which we call **trivial** intervals. It is easy to check that two trivial intervals cannot be dependent, so when a trivial and a non-trivial interval are dependent, it is enough to query the non-trivial interval in order to decide their relative order. This does not mean, however, that trivial intervals should never be queried, and in particular adaptive algorithms may decide to do that.

It is also clear that the dependency graph is an interval graph when $\delta = 0$. This is also true when $\delta > 0$ and there are no trivial intervals, in which case we can simply replace each interval $I_i = [\ell_i, r_i]$ with $I_i^{(\delta)} = [\ell_i^{(\delta)}, r_i^{(\delta)}] := [\ell_i + \delta/2, r_i - \delta/2] \neq \emptyset$, and it is easy to check that I_i and I_j are dependent with error threshold δ if and only if $I_i^{(\delta)}$ and $I_j^{(\delta)}$ are dependent with error threshold 0. Note however that we cannot use this reduction to solve the sorting problem, since the precise values could fall outside of the given interval.

3 Warm-Up: Offline and Oblivious Algorithms

The first result we present concerns finding the optimum query set for a given set of intervals, assuming we know the actual values in each interval. I.e., given the intervals $I_1, \ldots, I_n$ and the actual values $v_1, \ldots, v_n$, find a minimum-cost set Q of intervals to query, such that Q is sufficient to prove an ordering for $I_1, \ldots, I_n$ without the knowledge of $v_1, \ldots, v_n$. Solving this problem is useful, for example, to perform experimental evaluation of algorithms, since we are actually finding the offline optimum solution for the online (either oblivious or adaptive) problem. The ideas we present here will also be useful when solving the online problem.

We show that the problem can be solved optimally in polynomial time. The key observations behind the algorithm are the following. In order to simplify notation, we write $I_i \supset I_j$ for intervals I_i and I_j if $\ell_i < \ell_j$ and $r_j < r_i$.

▶ **Proposition 3.1.** *Let I_i and I_j be intervals with actual values v_i and v_j. If $I_j \supset [v_i - \delta, v_i + \delta]$, then I_j is queried by every optimum solution.*

Proof. Even if we have queried I_i, we have to query I_j because we may have $v_j \in [\ell_j, v_i - \delta)$ or $v_j \in (v_i + \delta, r_j]$. ◀

▶ **Proposition 3.2.** *Let I_i and I_j be two dependent intervals, v_i the actual value in I_i and v_j the actual value in I_j. If $I_i \not\supset [v_j - \delta, v_j + \delta]$ and $I_j \not\supset [v_i - \delta, v_i + \delta]$, then it is enough to query either I_i or I_j to decide their relative order.*

Proof. If we query I_i, then $v_i \notin [\ell_j + \delta, r_j - \delta]$, so we can pick a reasonable order between I_i and I_j. The argument is symmetrical if we query I_j. ◀

The algorithm begins with a query set Q containing all intervals that satisfy the condition in Proposition 3.1. Due to Proposition 3.2, it is enough to complement Q with a minimum-cost vertex cover in the dependency graph defined by the remaining intervals, which can be found in polynomial time for chordal graphs [16].

▶ **Theorem 3.3.** *The problem of finding an optimum query set for the sorting problem with uncertainty can be solved optimally in polynomial time.*

Now we consider oblivious algorithms. In this case, all non-trivial intervals with some dependence must be queried, and clearly this is the best possible strategy. In the following theorem, we show that this implies a tight bound of n on the query-competitive ratio for the case with uniform costs, and that in the general case the query-competitive ratio is unbounded.

▶ **Theorem 3.4.** *If query costs are uniform, any oblivious algorithm for sorting with uncertainty has query-competitive ratio exactly n. For arbitrary costs, the query-competitive ratio is unbounded.*

Proof. For the upper bound with uniform costs, a naïve algorithm that queries all intervals and then sorts the numbers suffices.

For both lower bounds, we have $n - 1$ independent intervals with length greater than 2δ, plus an interval I_n which contains all the other ones. Both an algorithm and the optimum solution must query I_n in order to decide where v_n fits in the order. If the algorithm does not query some I_i with $i < n$, then the adversary can set $v_n \in (\ell_i + \delta, r_i - \delta) \neq \emptyset$ and the algorithm cannot decide the order. Thus, without the knowledge of v_n, the algorithm must query all I_i with $i < n$. However, it may be the case that $v_n \notin I_i$ for all $i < n$, and querying I_n suffices to decide the order. This gives a lower bound of n on the query-competitive ratio for uniform query costs. For the general case, w_n can be arbitrarily small and the query-competitive ratio is unbounded. ◀

4 Deterministic Adaptive Algorithms

Now let us consider deterministic adaptive algorithms. We begin with a lower bound.

▶ **Lemma 4.1.** *Any deterministic adaptive algorithm for the sorting problem with uncertainty has query-competitive ratio at least 2, even if query costs are uniform and the dependency graph has large components.*

Proof. Consider intervals I_1 and I_2 with uniform query cost, $\ell_1 < \ell_2 < r_1 < r_2$ and $r_1 - \ell_2 > 2\delta$. If the algorithm queries I_1, then the adversary chooses $v_1 \in (\ell_2 + \delta, r_1 - \delta)$. The algorithm must also query I_2 to decide the order, but then the adversary can choose $v_2 \in [r_1 - \delta, r_2]$ and one query would be sufficient. The argument is symmetrical if the algorithm queries I_2 first, with $v_2 \in (\ell_2 + \delta, r_1 - \delta)$ and $v_1 \in [\ell_1, \ell_2 + \delta]$. To obtain a large component, make several independent copies of this structure and connected them by a large interval containing all the others. ◀

First we give a simple deterministic 2-query-competitive adaptive algorithm for the case with uniform query costs. It is inspired by the algorithm of Erlebach et al. [12] for the minimum spanning tree problem with uncertainty, and it relies on the following concepts, which were introduced in [7]. Let $\mathcal{I} = \{I_1, \dots, I_n\}$ be a set of intervals for the sorting problem with uncertainty. We say that a set $W \subseteq \mathcal{I}$ of intervals is a **witness set** if at least one of the intervals in W must be queried to decide the order of $\mathcal{I}$, even if all intervals except those in W are queried. Due to Lemma 2.2, any pair of dependent intervals constitute a witness set. A set of intervals $\mathcal{I}' = \{I'_1, \dots, I'_n\}$ is a **refinement** of $\mathcal{I}$ if $\mathcal{I}'$ is obtained from $\mathcal{I}$ by performing a sequence of queries. Proposition 4.2 follows simply from $\mathcal{I}'$ having more information than $\mathcal{I}$.

▶ **Proposition 4.2.** *Let $\mathcal{I}'$ be a refinement of $\mathcal{I}$. If some set of intervals $W \subseteq \mathcal{I}' \cap \mathcal{I}$ is a witness set for $\mathcal{I}'$, then it is a witness set for $\mathcal{I}$.*

The algorithm, then, consists in the following. While there is some pair of dependent intervals, we query all intervals in this pair that have not been queried yet. When an interval I_i is queried, it is replaced by $[v_i, v_i]$. (Note that, even after querying I_i, it may still be dependent to a non-trivial interval.) Finally, intervals are sorted by breaking ties arbitrarily.[4]

For a better understanding of the algorithm, consider the examples in Figure 1, assuming $\delta = 0$. In Figure 1a, the optimum solution must query I_1 and I_3, since $v_1 \in I_3$ and $v_3 \in I_1$, and this is enough because I_2 will be independent after querying I_1. If the algorithm first queries I_1 and I_2, it must also query I_3. In Figure 1b it is enough to query I_1, but the algorithm will query a dependent pair, say, I_1 and I_2. Either way, the algorithm does not spend more than twice the optimum number of queries.

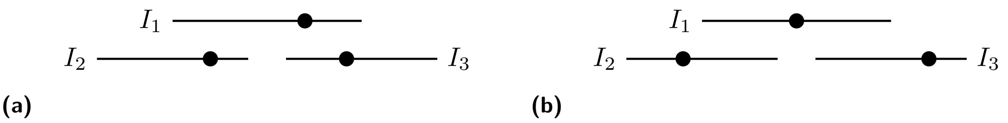


Figure 1 Example instances of the problem.

▶ **Theorem 4.3.** *The simple adaptive algorithm for sorting with uncertainty is 2-query-competitive for uniform query costs.*

Proof. Note that the optimum solution must query at least one interval in each witness set. For every pair $\{I_i, I_j\}$ of dependent intervals selected by the algorithm, we have that:

[4] If $\delta = 0$, then the algorithm can be implemented with stable sorting and in $O(n \lg n)$ time by running a standard stable sorting algorithm (e.g., MergeSort) and querying two intervals when MergeSort needs to know the relative order between them. It does not work, however, if $\delta > 0$, since the relation $v_i \le v_j + \delta$ is not transitive.

(1) if both I_i and I_j have not been queried yet, the algorithm queries the witness set $\{I_i, I_j\}$; (2) if I_i has already been queried then, by Proposition 3.1, $\{I_j\}$ is a witness set, which is queried by the algorithm. We can conclude that the algorithm only queries disjoint witness sets of size at most 2, and thus it queries at most twice the minimum number of intervals. ◀

For arbitrary query costs, the problem also admits a 2-query-competitive deterministic adaptive algorithm, although not as simple. The algorithm first queries a minimum-cost vertex cover S_1 on the dependency graph. Then, it queries all non-trivial intervals that are still dependent after querying S_1, which we denote by the set S_2.

▶ **Theorem 4.4.** *The adaptive algorithm for sorting with uncertainty with arbitrary query costs is 2-query-competitive.*

Proof. Let Q be an optimum query set. The set of intervals not contained in Q must be independent. By the duality between independent sets and vertex covers, Q must be a vertex cover. Thus $w(S_1) \leq w(Q)$, since S_1 has minimum cost. Furthermore, note that every interval in S_2 is a singleton witness set, since S_2 is a set of independent intervals. Thus $w(S_2) \leq w(Q)$ as well, and $w(S_1 \cup S_2) \leq 2 \cdot w(Q)$. ◀

5 Improved Adaptive Algorithms for Uniform Query Costs

We now explore refined analysis of query-competitive sorting. We present a unified strategy that yields different improvements to Theorem 4.3, depending on what assumptions we make.

The core observation is that the bad 2-interval instance in the proof of Lemma 4.1 is the only structure that prevents an algorithm from performing better than twice the optimum. The first strategy that comes to mind, then, is to use randomization: a simple randomized strategy attains query-competitive factor 3/2 on the instance of Lemma 4.1. Before extending the algorithm to arbitrary instances, we give a lower bound for any randomized algorithm.

▶ **Lemma 5.1.** *Any randomized adaptive algorithm has query-competitive ratio at least* 3/2 *against an adversary that is oblivious to the randomized tosses, even for uniform query costs.*

Proof. Use the same bad instance as Lemma 4.1, set probability 1/2 for each of the two possible inputs and apply Yao's minimax principle. ◀

The algorithm is based on the following property of the dependency graph, whose proof we omit.

▶ **Lemma 5.2.** *If I_x is an interval with minimum r_x, then the vertex x is simplicial, i.e., its neighborhood is a clique.*

The algorithm begins by querying intervals that are singleton witness sets according to a generalization of the condition in Proposition 3.1. Then, if a component of the remaining dependency graph is an edge, the randomized strategy is applied. Else, the algorithm considers a non-queried vertex x with minimum r_x, a neighbor y of x with minimum r_y, and another neighbor z of x (or of y if y is the only neighbor of x) with minimum r_z. The algorithm first queries I_y. If x and y are still adjacent, or if x and z are adjacent, then we query both I_x and I_z. We repeat this strategy until the dependency graph has no edges.

A pseudocode is presented in Algorithm 1; we parameterize the probability p in the randomized strategy since the algorithm will be reused afterwards. We also maintain a set $\mathcal{V}$ of the values resulting of queried intervals.

Algorithm 1 Improved adaptive algorithm for the sorting problem with queries.

```
   Input: $(I_1, \dots, I_n, p)$
1  $\mathcal{V} \leftarrow \emptyset$;
2  while there are $i, j$ with $I_i \supset [\ell_j - \delta, r_j + \delta]$ or $I_i \supset [v_j - \delta, v_j + \delta]$ with $v_j \in \mathcal{V}$ do
3      query $I_i$, add $v_i$ to $\mathcal{V}$;
4  while there is some dependency do
5      if some component is an edge $ij$ then
6          pick $i$ with probability $p$ (and $j$ with probability $1 - p$); assume $i$ is picked;
7          query $I_i$, add $v_i$ to $\mathcal{V}$;
8          if $I_j \supset [v_i - \delta, v_i + \delta]$ then
9              query $I_j$, add $v_j$ to $\mathcal{V}$;
10     else
11         let $I_x$ non-queried with $\min r_x$, and $y$ be a neighbor of $x$ with $\min r_y$;
12         let $z$ be another neighbor of $x$ (or of $y$ if $x$ is a leaf), with $\min r_z$;
13         query $I_y$, add $v_y$ to $\mathcal{V}$;
14         if $I_x \supset [v_y - \delta, v_y + \delta]$ or $I_x, I_z$ are dependent then
15             query $I_x$, add $v_x$ to $\mathcal{V}$;
16             query $I_z$, add $v_z$ to $\mathcal{V}$;
17     while there is $I_i \supset [v_j - \delta, v_j + \delta]$ for some $v_j \in \mathcal{V}$ do
18         query $I_i$, add $v_i$ to $\mathcal{V}$;
```

▶ **Theorem 5.3.** *Algorithm 1 has expected query-competitive ratio $3/2$ if $p = 1/2$.*

Proof. We form a partition $V_1, \dots, V_m$ of the set of input intervals with the following property. Let $a(V_i)$ be the number of intervals in V_i that are queried by the algorithm, and let $q(V_i) := |Q \cap V_i|$, where Q is an optimum query set. We show that $\mathbb{E}[a(V_i)/q(V_i)] \leq 3/2$ for every i, from which the theorem follows.

If the algorithm queries an interval I_i in Line 3 or Line 18, then $\{I_i\}$ is the next set in the partition. Due to Proposition 3.1, it is a singleton witness set, so $a(\{I_i\})/q(\{I_i\}) = 1$.

If the algorithm runs Lines 6–9 for edge ij, then $W = \{I_i, I_j\}$ is the next set in the partition. We consider the following cases.

1. If $I_i \supset [v_j - \delta, v_j + \delta]$ and $I_j \supset [v_i - \delta, v_i + \delta]$, then $q(W) = 2$ and $a(W) = 2$.
2. Otherwise, $q(W) \geq 1$ because this is a witness pair.
   a. If $I_i \supset [v_j - \delta, v_j + \delta]$ but $I_j \not\supset [v_i - \delta, v_i + \delta]$, then with probability $1/2$ the algorithm queries I_i and this is enough, and with probability $1/2$ it queries both, so $\mathbb{E}[a(W)] = 3/2$; the same holds for the symmetrical case.
   b. If $I_i \not\supset [v_j - \delta, v_j + \delta]$ and $I_j \not\supset [v_i - \delta, v_i + \delta]$, then Line 9 is not executed and $a(W) = 1$.

If the algorithm runs Lines 11–16 for x, y and z, then we have two cases.

1. If x and z are not neighbors, and x and y are not neighbors after Line 13, then $W = \{I_x, I_y\}$ is the next set in the partition. Since it is a witness set, $q(W) \geq 1$. But the algorithm will not query I_x because y is the only neighbor of x, so $a(W) = 1$.
2. Otherwise, $W = \{I_x, I_y, I_z\}$ is the next set in the partition. We have two subcases.
   a. If x and z are neighbors, then xyz is a clique by Lemma 5.2. So $q(W) \geq 2$, since otherwise a pair is unsolved.
   b. Otherwise, $I_x \supset [v_y - \delta, v_y + \delta]$ and $\{I_x\}$ is a singleton witness set. Since x and z are not neighbors, then y and z are neighbors and, by Lemma 2.2, $\{I_y, I_z\}$ is a witness set.

   Either way, $q(W) \geq 2$ and $a(W) = 3$.

We conclude that the expected query-competitive ratio is $3/2$. ◀

Our second strategy to obtain an improvement on Theorem 4.3 is, instead of using randomization, to assume that the graph does not have 2-components, i.e., components consisting of a single edge. This is not enough, however, since in Lemma 4.1 we have shown that we can have a large component. So our hypothesis is that $\delta = 0$ and, after executing the loop of Lines 2–3, the remaining dependency graph, which becomes a proper interval graph, has no 2-components. (Note that Theorem 5.3 is still true if we remove Lines 2–3 of the algorithm.) Let us prove a lower bound for this case.

▶ **Lemma 5.4.** *Any deterministic adaptive algorithm has query-competitive ratio at least* $5/3$*, even if* $\delta = 0$ *and the dependency graph is a proper interval graph with no* 2*-components.*

Proof. Consider five proper intervals I_a, I_b, I_c, I_d, I_e with $\ell_a < \ell_b < \ell_c < \ell_d < \ell_e$. The dependencies are defined by two triangles, abc and cde.

If the algorithm first queries I_c, then we set $v_c \in I_c \setminus (I_a \cup I_b \cup I_d \cup I_e)$, and we can make ab and de behave as the bad instance of Lemma 4.1.

If the algorithm first queries I_a, then we set $v_a \in I_b \cap I_c$, so the algorithm will be forced to query I_b and I_c, and we set $v_b, v_c \in (I_b \cup I_c) \setminus (I_a \cup I_d \cup I_e)$, so the optimum can avoid I_a. Then we can make de behave as the bad instance of Lemma 4.1. The argument is symmetric if the algorithm first queries I_e.

If the algorithm first queries I_b, then we set $v_b \in I_a \cap I_c$, so the algorithm will be forced to query I_a and I_c, and we set $v_a, v_c \in (I_a \cup I_c) \setminus (I_b \cup I_d \cup I_e)$, so the optimum can avoid I_b. Then we can make de behave as the bad instance of Lemma 4.1. The argument is symmetric if the algorithm first queries I_d. ◀

▶ **Theorem 5.5.** *Algorithm 1 (with* $p = 0$ *or* 1*) is* $5/3$*-query-competitive if* $\delta = 0$ *and the dependency graph has no* 2*-components after finishing the loop of Lines 2–3.*

Proof. The analysis is similar to that of Theorem 5.3. We will give a partition $V_1, \ldots, V_m$ of the set of intervals with the following property. Let $a(V_i)$ be the number of intervals in V_i that are queried by the algorithm, and let $q(V_i) := |Q \cap V_i|$, where Q is an optimum query set. We will have that $a(V_i)/q(V_i) \leq 5/3$ for every i, and then the theorem follows. The analysis for the cases of Lines 3, 11–16 and 18 are identical.

If the algorithm runs Lines 6–9 for edge ij, then let C be the component containing ij in the dependency graph after finishing the loop of Lines 2–3. We claim that i and j are the only vertices of C queried in Lines 6–9: Lines 11–16 force that intervals are queried from left to right; thus, since the dependency graph at this point is a proper interval graph, if an interval i' is queried in Line 18, then after that no interval j' with $r_{j'} < r_{i'}$ will have some dependency. Pick an arbitrary set W' of the partition consisting of vertices of C. We merge $\{I_i, I_j\}$ and W' into a single set W of the partition, and from the previous cases we have that $a(W)/q(W) \leq 5/3$. ◀

This proof indicates that the analysis can be improved if we require the graph to have *large* components after finishing the loop of Lines 2–3.

▶ **Theorem 5.6.** *Algorithm 1 (with* $p = 0$ *or* 1*) has query-competitive factor* $3/2 + \mathrm{O}(1/k)$ *if* $\delta = 0$ *and each component of the dependency graph has size at least* k *after finishing the loop of Lines 2–3.*

Proof. We only have to reconsider the case of Lines 6–9 in the proof of Theorem 5.5. If the algorithm runs Lines 6–9 for edge ij, then let C be the component containing ij in the dependency graph after finishing the loop of Lines 2–3. We merge $\{I_i, I_j\}$ and all partition sets containing vertices of C into a single set W of the partition. Since i and j are the only vertices of C queried in Lines 6–9 and C has size at least k, from the other cases we have that $a(W)/q(W) \leq 3/2 + \mathrm{O}(1/k)$. ◀

The analysis is tight since we can have a chain of k triangles plus 1 edge, such that we can force the algorithm to query all intervals, while the optimum can avoid one interval in each triangle and one interval in the extra edge. For large components, we still have a lower bound of $7/5$ for any deterministic algorithm.

▶ **Lemma 5.7.** *Any deterministic adaptive algorithm has query-competitive ratio at least* $7/5$*, even if* $\delta = 0$ *and the dependency graph is a proper interval graph with large components.*

Proof. (Lemma 5.7.) Consider the graph of Figure 2, which has $7k + 2$ vertices. For $i = 0, \ldots, k-1$, vertices $7i + 3, \ldots, 7i + 7$ consist in a copy of the instance of Lemma 5.4. For $i = 0, \ldots, k$, vertices $x_i = 7i + 1$ and $y_i = 7i + 2$ are dependent, x_i is dependent to $7(i-1) + 7$ if $i > 0$, and y_i is dependent to $7i + 3$ if $i < k$. We set $v_{x_i}, v_{y_i} \in I_{x_i} \cap I_{y_i}$, so both the algorithm and the optimum must query I_{x_i} and I_{y_i}, but querying them gives us no information about the remaining vertices. From Lemma 5.4, we can force any deterministic algorithm to query all vertices in the graph, while the optimum solution can query only 3 vertices of $7i + 3, \ldots, 7i + 7$. ◀

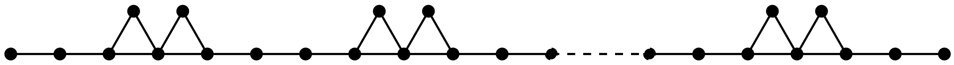


Figure 2 Instance which attains the lower bound for proper interval graphs with large components.

It remains an open question to close the gap between the lower bound of $7/5$ and the upper bound of $3/2 + \mathrm{O}(1/k)$. Finally, we note that the problem can be solved exactly for laminar families of intervals, since all queries will happen at Line 3 of the algorithm.

▶ **Theorem 5.8.** *Algorithm 1 obtains an optimum solution if* $\delta = 0$ *and the intervals constitute a laminar family.*

6 Advice Complexity for Adaptive Algorithms

In this section we investigate the advice complexity of solving the adaptive version of the problem. We assume arbitrary query costs, and for consistency that the oracle answers questions regarding a fixed optimum solution for the given instance.

First, we deal with the case when $\delta = 0$. Let n be the number of given intervals. We claim that $\lfloor n/2 \rfloor$ bits of advice are sufficient to solve the problem exactly, and that there are instances for which $\lfloor n/2 \rfloor$ bits are necessary.

▶ **Lemma 6.1.** *The advice complexity of the adaptive sorting problem with uncertainty is at least* $\lfloor n/2 \rfloor$*, where* n *is the number of intervals, even if* $\delta = 0$*.*

Proof. Assume n even and consider $n/2$ independent copies of the bad instance of Lemma 4.1. At least 1 bit of advice is necessary to decide the relative order between each pair. ◀

For an adaptive algorithm with a matching upper bound, we note that, if $\delta = 0$, then any triangle ijk contains a vertex j such that $I_j \subseteq I_i \cup I_k$ (just take i with minimum ℓ_i and k with maximum r_k). Thus, we can ask the oracle whether the optimum solution queries I_j; if not, then we must query all neighbors of j; otherwise, we query I_j, and since $I_j \subseteq I_i \cup I_k$, we will know at least one of I_i and I_k that also must be queried. If the dependency graph contains no triangles, then it is a forest, because any cycle in a chordal graph must contain a triangle. Therefore, we can pick a leaf i and ask the oracle whether the optimum solution queries its neighbor j; if not, then we query all neighbors of j; otherwise, we query I_j and we will know if I_i must or not be queried. Since we decide at least two intervals with one bit of advice, then $\lfloor n/2 \rfloor$ bits are sufficient. We present a pseudocode in Algorithm 2.

Algorithm 2 An adaptive algorithm that finds an optimum solution with $\lfloor n/2 \rfloor$ bits of advice when $\delta = 0$.

```
Input: $(I_1, \ldots, I_n)$
$\mathcal{V} \leftarrow \emptyset$;
while there is some dependency do
    if there is a triangle $K$ then
        let $i \in K$ with minimum $\ell_i$, $k \in K$ with maximum $r_k$, and $j \in K \setminus \{i, k\}$;
    else let $i$ be a leaf, and $j$ be the neighbor of $i$ ;
    ask the oracle whether the optimum solution queries $j$;
    if yes then query $I_j$, add $v_j$ to $\mathcal{V}$;
    else foreach neighbor $z$ of $j$ do
        query $I_z$, add $v_z$ to $\mathcal{V}$;
    while there is $I_i \supset [v_j - \delta, v_j + \delta]$ for some $v_j \in \mathcal{V}$ do
        query $I_i$, add $v_i$ to $\mathcal{V}$;
```

▶ **Theorem 6.2.** *The advice complexity of the adaptive sorting problem with uncertainty is $\lfloor n/2 \rfloor$ when $\delta = 0$, where n is the number of intervals.*

Now we consider the case when $\delta > 0$. Here, we can improve the lower bound to $\lceil n/3 \cdot \lg 3 \rceil$ and still have an algorithm with matching upper bound. Both are based on the fact that to encode k distinct values amortized $\lg k$ bits are sufficient and necessary [27].

▶ **Lemma 6.3.** *The advice complexity of the adaptive sorting problem with uncertainty is at least $\lceil n/3 \cdot \lg 3 \rceil$, where n is the number of intervals.*

Proof. Assume n multiple of 3 and consider $n/3$ independent triangles; it suffices to bound the number of bits of advice necessary to solve each triangle. Suppose by contradiction that there is an algorithm that solves any triangle with one bit of advice, and consider the following instances $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3$. In each $\mathcal{I}_i$, the k-th triangle has intervals I_1, I_2, I_3 such that $\ell_1 < \ell_2 < \ell_3 - \delta$, $r_1 + \delta < r_2 < r_3$, $\ell_2 \leq \ell_1 + \delta$, $r_2 \geq r_3 - \delta$ and $r_1 - \ell_3 > 2\delta$. We have $v_j = r_j$ for all j in $\mathcal{I}_1$; in $\mathcal{I}_2$, $v_1 = \ell_1$, $v_2 \in (\ell_3 + \delta, r_1 - \delta)$, $v_3 = r_3$; and $v_j = \ell_j$ for all j in $\mathcal{I}_3$. The only optimum solution for $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3$ is not to query I_1, I_2, I_3, respectively. (See Figure 3.)

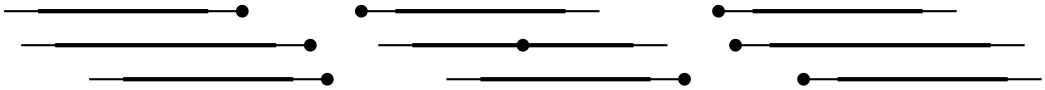


Figure 3 Instances for the lower bound on advice complexity when $\delta > 0$.

By the pigeonhole principle, the algorithm must have the same advice for at least two of those inputs. So, it suffices to prove that any deterministic algorithm fails in one instance of any subset with at least two of those instances. Since the intervals are structurally identical, any algorithm for a triangle performs no better than an algorithm in the following form, for fixed $x, y \in \{1, 2, 3\}$, $x \neq y$: query I_x, and if no helpful information is given, query I_y. The instances are constructed in such a way that, for instance $\mathcal{I}_i$, the algorithm does not get any helpful information by querying I_x with $i \neq x$, so it fails on instances $\mathcal{I}_x$ and $\mathcal{I}_y$. Since one bit is not sufficient, at least three different values must be encoded in the advice for each triangle, so $\lceil n/3 \cdot \lg 3 \rceil$ bits are necessary for the whole instance. ◀

The algorithm that attains the upper bound relies on Lemma 5.2. It considers the clique K consisting of vertex x with minimum r_x and its neighborhood. Then it asks the oracle for the index of a vertex y in K that is not queried in the optimum solution or, if there is no such vertex in K, then the oracle must return $y = x$. Either way, the algorithm queries all intervals in $K \setminus \{y\}$, and if $y = x$ then the algorithm will know if y must also be queried after querying everyone else. So it uses $\lg |K|$ bits of advice to decide at least $|K|$ intervals, and the bound follows since $\lg k/k$ has its maximum at $k = 3$ when k is integer. A pseudocode is presented in Algorithm 3.

Algorithm 3 An adaptive algorithm that finds an optimum solution with $\lceil n/3 \cdot \lg 3 \rceil$ bits of advice.

```
Input: $(I_1, \ldots, I_n)$
$\mathcal{V} \leftarrow \emptyset$;
while there is some dependency do
    let $x$ with minimum $r_x$, and $K$ be the clique consisting of $x$ and its neighborhood;
    ask the oracle for a vertex $y \in K$ not queried in the optimum solution, or $y = x$ if
      there is no such vertex;
    foreach $z \in K \setminus \{y\}$ do
        query $I_z$, add $v_z$ to $\mathcal{V}$;
    while there is $I_i \supset [v_j - \delta, v_j + \delta]$ for some $v_j \in \mathcal{V}$ do
        query $I_i$, add $v_i$ to $\mathcal{V}$;
```

▶ **Theorem 6.4.** *The advice complexity of the adaptive sorting problem with uncertainty is* $\lceil n/3 \cdot \lg 3 \rceil$*, where n is the number of intervals.*

7 Future Work Directions

One interesting question is how the sorting problem can take advantage of queries with different levels of precision, as in [21]. A variety of other classical optimization problems could also be studied in similar uncertainty variants with queries.

References

1 M. Ajtai, V. Feldman, A. Hassidim, and J. Nelson. Sorting and Selection with Imprecise Comparisons. *ACM Transactions on Algorithms*, 12(2):19:1–19:19, 2016. doi:10.1145/2701427.

2 L. Arantes, E. Bampis, A. V. Kononov, M. Letsios, G. Lucarelli, and P. Sens. Scheduling under uncertainty: A query-based approach. In *IJCAI 2018: 27th International Joint Conference on Artificial Intelligence*, pages 4646–4652, 2018. doi:10.24963/ijcai.2018/646.

3 H.-G. Beyer and B. Sendhoff. Robust optimization – a comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33-34):3190–3218, 2007. doi:10.1016/j.cma.2007.03.003.

4 J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming.* Springer Series in Operations Research and Financial Engineering. Springer, 2011.

5 A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

6 J. Boyar, L. M. Favrholdt, C. Kudahl, K. S. Larsen, and J. W. Mikkelsen. Online Algorithms with Advice: A Survey. *ACM Computing Surveys*, 50(2), 2017. doi:10.1145/3056461.

7 R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman. Efficient Update Strategies for Geometric Computing with Uncertainty. *Theory of Computing Systems*, 38(4):411–423, 2005. doi:10.1007/s00224-004-1180-4.

8 G. Charalambous and M. Hoffmann. Verification Problem of Maximal Points under Uncertainty. In T. Lecroq and L. Mouchard, editors, *IWOCA 2013: 24th International Workshop on Combinatorial Algorithms*, volume 8288 of *Lecture Notes in Computer Science*, pages 94–105. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-45278-9_9.

9 C. Dürr, T. Erlebach, N. Megow, and J. Meißner. Scheduling with Explorable Uncertainty. In A. R. Karlin, editor, *ITCS 2018: 9th Innovations in Theoretical Computer Science Conference*, volume 94 of *Leibniz International Proceedings in Informatics*, pages 30:1–30:14, 2018. doi:10.4230/LIPIcs.ITCS.2018.30.

10 T. Erlebach and M. Hoffmann. Query-competitive algorithms for computing with uncertainty. *Bulletin of EATCS*, 116:22–39, 2015. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/335.

11 T. Erlebach, M. Hoffmann, and F. Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. *Theoretical Computer Science*, 613:51–64, 2016. doi:10.1016/j.tcs.2015.11.025.

12 T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihal'Ák, and R. Raman. Computing minimum spanning trees with uncertainty. In *STACS'08: 25th International Symposium on Theoretical Aspects of Computer Science*, pages 277–288, 2008. URL: https://arxiv.org/abs/0802.2855.

13 T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62(1):1–18, 2007. doi:10.1016/j.jalgor.2004.07.005.

14 T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32(2):538–547, 2003. doi:10.1137/S0097539701395668.

15 J. Focke, N. Megow, and J. Meißner. Minimum Spanning Tree under Explorable Uncertainty in Theory and Experiments. In C. S. Iliopoulos, S. P. Pissis, S. J. Puglisi, and R. Raman, editors, *SEA 2017: 16th International Symposium on Experimental Algorithms*, volume 75 of *Leibniz International Proceedings in Informatics*, pages 22:1–22:14, 2017. doi:10.4230/LIPIcs.SEA.2017.22.

16 F. Gavril. Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph. *SIAM Journal on Computing*, 1(2):180–187, 1972. doi:10.1137/0201013.

17 M. Goerigk, M. Gupta, J. Ide, A. Schöbel, and S. Sen. The robust knapsack problem with queries. *Computers & Operations Research*, 55:12–22, 2015. doi:10.1016/j.cor.2014.09.010.

18 M. Gupta, Y. Sabharwal, and S. Sen. The Update Complexity of Selection and Related Problems. *Theory of Computing Systems*, 59(1):112–132, 2016. doi:10.1007/s00224-015-9664-y.

19 C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962. doi:10.1093/comjnl/5.1.10.

20 S. Kahan. A model for data in motion. In *STOC'91: 23rd Annual ACM Symposium on Theory of Computing*, pages 265–277, 1991. doi:10.1145/103418.103449.

21 S. Khanna and W.-C. Tan. On computing functions with uncertainty. In *PODS'01: 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 171–182, 2001. doi:10.1145/375551.375577.

22 N. Megow, J. Meißner, and M. Skutella. Randomization Helps Computing a Minimum Spanning Tree under Uncertainty. *SIAM Journal on Computing*, 46(4):1217–1240, 2017. doi:10.1137/16M1088375.

23 C. L. Monma, B. Reed, and W.T. Trotter Jr. Threshold Tolerance Graphs. *Journal of Graph Theory*, 12(3):343–362, 1988. doi:10.1002/jgt.3190120307.

24 C. Olston and J. Widom. Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data. In *VLDB 2000: 26th International Conference on Very Large Data Bases*, pages 144–155, 2000. URL: http://ilpubs.stanford.edu:8090/437/.

25 I. O. Ryzhov and W. B. Powell. Information Collection for Linear Programs with Uncertain Objective Coefficients. *SIAM Journal on Optimization*, 22(4):1344–1368, 2012. doi:10.1137/12086279X.

26 A. Salah, K. Li, and K. Li. Lazy-Merge: A Novel Implementation for Indexed Parallel k-Way In-Place Merging. *IEEE Transactions on Parallel and Distributed Systems*, 27(7):2049–2061, 2015. doi:10.1109/TPDS.2015.2475763.

27 C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

28 Y. Yamaguchi and T. Maehara. Stochastic Packing Integer Programs with Few Queries. In *SODA'18: Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 293–310, 2018. doi:10.1137/1.9781611975031.21.

Better Bounds for Online Line Chasing

Marcin Bienkowski
Institute of Computer Science, University of Wrocław, Poland
marcin.bienkowski@cs.uni.wroc.pl

Jarosław Byrka
Institute of Computer Science, University of Wrocław, Poland
jaroslaw.byrka@cs.uni.wroc.pl

Marek Chrobak
University of California at Riverside, CA, USA
marek@cs.ucr.edu

Christian Coester
University of Oxford, United Kingdom
christian.coester@cs.ox.ac.uk

Łukasz Jeż
Institute of Computer Science, University of Wrocław, Poland
lukasz.jez@cs.uni.wroc.pl

Elias Koutsoupias
University of Oxford, United Kingdom
elias@cs.ox.ac.uk

Abstract

We study online competitive algorithms for the *line chasing problem* in Euclidean spaces $\mathbb{R}^d$, where the input consists of an initial point P_0 and a sequence of lines $X_1, X_2, ..., X_m$, revealed one at a time. At each step t, when the line X_t is revealed, the algorithm must determine a point $P_t \in X_t$. An online algorithm is called c-competitive if for any input sequence the path $P_0, P_1, ..., P_m$ it computes has length at most c times the optimum path. The line chasing problem is a variant of a more general convex body chasing problem, where the sets X_t are arbitrary convex sets.

To date, the best competitive ratio for the line chasing problem was 28.1, even in the plane. We improve this bound by providing a simple 3-competitive algorithm for any dimension d. We complement this bound by a matching lower bound for algorithms that are memoryless in the sense of our algorithm, and a lower bound of 1.5358 for arbitrary algorithms. The latter bound also improves upon the previous lower bound of $\sqrt{2} \approx 1.412$ for convex body chasing in 2 dimensions.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases convex body chasing, line chasing, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.8

Funding Research supported by NSF grant CCF-1536026, by Polish National Science Centre grants 2016/22/E/ST6/00499 and 2015/18/E/ST6/0045, and by ERC Advanced Grant 321171 (ALGAME).

1 Introduction

Convex body chasing is a fundamental problem in online computation. It asks for an incrementally-computed path that traverses a given sequence of convex sets provided one at a time in an online fashion and is as short as possible. Formally, the input consists of an initial point $P_0 \in \mathbb{R}^d$ and a sequence $X_1, X_2, ..., X_m \subseteq \mathbb{R}^d$ of convex sets. The objective is to find a path $\mathbf{P} = (P_0, P_1, ..., P_m)$ with $P_t \in X_t$ for each $t = 1, 2, ..., m$ and minimum total length $\ell(\mathbf{P}) = \sum_{t=1}^{m} \ell_{P_{t-1}P_t}$. (Throughout the paper, by $\ell(P, Q)$ or ℓ_{PQ} we denote the Euclidean distance between points P and Q in $\mathbb{R}^d$.) This path $\mathbf{P}$ must be computed *online*, in the following sense: the sets X_t are revealed over time, one per time step. At step t, when set X_t is revealed, we need to immediately and irrevocably identify its visit point $P_t \in X_t$. Thus the choice of P_t does not depend on the future sets $X_{t+1}, ..., X_m$.

As can be easily seen, in this online scenario computing an optimal solution is not possible, and thus all we can hope for is to find a path whose length only approximates the optimum value. A widely accepted measure for the quality of this approximation is the competitive ratio. For a constant $c \geq 1$, we will say that an online algorithm $\mathcal{A}$ is *c-competitive* if it computes a path whose length is at most c times the optimum solution (computed offline). This constant c is called the *competitive ratio* of $\mathcal{A}$. Our objective is then to design an online algorithm whose competitive ratio is as close to 1 as possible.

The convex body chasing problem was originally introduced in 1993 by Friedman and Linial [11], who gave a constant-competitive algorithm for chasing convex bodies in $\mathbb{R}^2$ (the plane) and conjectured that it is possible to achieve constant competitiveness in any d-dimensional space $\mathbb{R}^d$. As shown in [11], this constant would have to depend on d; in fact it needs to be at least $\sqrt{d}$.

The Friedman-Linial conjecture has remained open for over two decades. In the last several years this topic has experienced a sudden increase in research activity, partly motivated by connections to machine learning (see [3, 7]), resulting in rapid progress. In 2016, Antoniadis *et al.* [1] gave a $2^{O(d)}$-competitive algorithm for chasing affine spaces of any dimension. In 2018, Bansal *et al.* [3] gave an algorithm with competitive ratio $2^{O(d \log d)}$ for nested families of convex sets, where the input set sequence satisfies $X_1 \supseteq X_2 \supseteq ... \supseteq X_m$. Soon later their bound was improved to $O(d \log d)$ by Argue *et al.* [2], and then to $O(\sqrt{d \log d})$ by Bubeck *et al.* [6]. Finally, Bubeck *et al.* [7] just recently announced a proof of the Friedman-Linial conjecture, providing an algorithm with competitive ratio $2^{O(d)}$ for arbitrary convex sets.

One other natural variant of convex body chasing that also attracted attention in the literature is *line chasing*, where all sets X_t are lines. Friedman and Linial [11] gave an online algorithm for line chasing in $\mathbb{R}^2$ with ratio 28.53. Their algorithm was simplified by Antoniadis *et al.* [1], who also slightly improved the ratio, to 28.1. Earlier, in 2014, Sitters [16] showed that a generalized work function algorithm has constant competitive ratio for line chasing, but he did not determine the value of the constant.

1.1 Our results

We study the line chasing problem discussed above. We give a 3-competitive algorithm for line chasing in $\mathbb{R}^d$, for any dimension $d \geq 2$, significantly improving the competitive ratios from [11, 1, 16]. Our algorithm is very simple and essentially memoryless, as it only needs to keep track of the last line in the request sequence. We start by providing the algorithm for line chasing in the plane, in Section 2, and later in Section 3 we extend it to an arbitrary dimension. In Section 4, we provide a matching lower bound of 3 for algorithms that are memoryless in the sense stated above and oblivious with respect to rotation, translation and uniform scaling of the metric space. We also provide a lower bound for arbitrary algorithms (see Section 5), showing that no online algorithm can achieve competitive ratio better than 1.5358. This improves the lower bound of $\sqrt{2} \approx 1.412$ for line chasing established in [11], which was previously also the best known lower bound for the more general problem of convex body chasing in the plane.

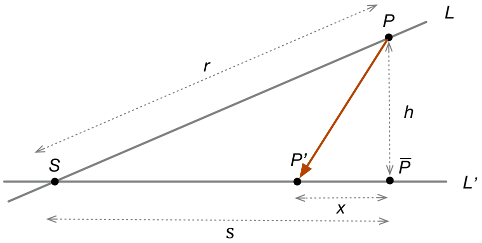


Figure 1 Algorithm Drift moves from P to P'.

1.2 Other related work

Set chasing problems are also known as *Metrical Service Systems* (see below) and belong to a very general class of problems for online optimization and competitive analysis called *Metrical Task Systems (MTS)* [5]. An instance of MTS specifies a metric space M, an initial point $P_0 \in M$, and a sequence of non-negative functions $\tau_1, \tau_2, ..., \tau_m$ over M called *tasks*. These tasks arrive online, one at a time. At each step t, the algorithm needs to choose a point $P_t \in M$ where it moves to "process" the current task τ_t. The goal is to minimize the total cost defined by $\sum_{t=1}^{m}(\mu(P_{t-1}, P_t) + \tau_t(P_t))$, where $\mu()$ is the metric in M. Thus in MTS, in addition to movement cost, at each step we also pay the cost of "processing" τ_t. For any metric space M with n points, if we allow arbitrary non-negative task functions then a competitive ratio of $2n - 1$ can be achieved and is optimal. This general bound is not particularly useful, because in many online optimization problems that can be modeled as an MTS, the metric space M has additional structure and only tasks of some special form are allowed, which makes it possible to design online algorithms with constant competitive ratios, independent of the size of M.

An MTS where $M = \mathbb{R}^d$ and all functions τ_t are convex is referred to as *convex function chasing*, and was studied in [1, 4, 13]. For the special case of convex functions on the real line, a 2-competitive algorithm was given in [4].

An MTS where each task function τ_t takes value 0 on a subset $X_t \subseteq M$ and ∞ elsewhere is called a *Metrical Service System (MSS)* [9]. In other words, in an MSS, in each step t the algorithm needs to move to a point in X_t. To achieve a competitive ratio independent of the size of M, it is generally required to restrict the sets X_t to be in some subset $\mathcal{X} \subsetneq \mathcal{P}(M)$. For instance, finite competitive ratios can be achieved when $\mathcal{X}$ is the set of sets of size at most k [10, 8, 15]. If $M = \mathbb{R}^d$ and $\mathcal{X}$ is the set of convex subsets, this is precisely the convex body chasing problem, and if $\mathcal{X}$ is the set of lines, it is the line chasing problem. One variant of MSS that has been particularly well studied is the famous *k-server problem* (see, for example, [14, 12]), in which one needs to schedule movement of k servers in response to requests arriving online in a metric space, where each request must be covered by one server. (In the MSS representation of the k-server problem, each set X_t consists of all k-tuples of points that include the request point at step t.)

2 A 3-Competitive Algorithm in the Plane

In this section, we present our online algorithm for line chasing in $\mathbb{R}^2$ with competitive ratio 3. The intuition is this: suppose that the last requested line is L and that the algorithm moved to point $P \in L$. Let L' be the new request line, S the intersection point of L and L', and $r = \ell_{SP}$. A naïve greedy algorithm would move to the point $\bar{P}$ on L' nearest to P (see

Figure 1) at cost $h = \ell_{P\bar{P}}$. If h is small, then $r - \ell_{S\bar{P}} = o(h)$, that is the distance between the greedy algorithm's point and S decreases only by a negligible amount. But the adversary can move to S, paying cost r, and then alternate requests on L and L'. On this sequence the overall cost of this algorithm would be $\omega(r)$, so it would not be constant-competitive. This example shows that if the angle between L and L' is small then the drift distance towards S needs to be roughly proportional to h. Our algorithm is designed so that this distance is roughly $h/\sqrt{2}$ if h is small (with the coefficient chosen to optimize the competitive ratio), and that it becomes 0 when L' is perpendicular to L.

Algorithm 1 Algorithm Drift.

Suppose that the last request is line L and that the algorithm is on point $P \in L$. Let the new request be L' and for any point $X \in L$, let $\bar{X}$ be the orthogonal projection of X onto L'. If L' does not intersect L, move to $P' = \bar{P}$. Otherwise, let $S = L \cap L'$ be the intersection point of L and L'. Let also $r = \ell_{SP}$, $h = \ell_{P\bar{P}}$, and $s = \ell_{S\bar{P}}$ (see Figure 1). Move to point $P' \in L'$ such that $\ell_{SP'} = s - x$, where $x = \frac{1}{\sqrt{2}}(h + s - r)$.

▶ **Theorem 1.** *Algorithm* Drift *is* 3*-competitive for the line chasing problem in* $\mathbb{R}^2$.

Proof. We establish an upper bound on the competitive ratio via amortized analysis, based on a potential function. The (always non-negative) value of this potential function, $\Phi(P, A)$, depends on locations $P, A \in L$ of the algorithm's and the adversary's point on the current line L. If L' is the new request line, and $P', A' \in L'$ are the new locations of the algorithm's and adversary's points, we want this function to satisfy

$$\ell_{PP'} + \Phi(P', A') - \Phi(P, A) \;\leq\; 3\,\ell_{AA'}. \tag{1}$$

Since initially the potential is 0 and is always non-negative, adding inequality (1) for all moves will establish 3-competitiveness of Algorithm Drift.

The potential function we use in our proof is $\Phi(P, A) = \sqrt{3}\,\ell_{AP}$. Substituting this formula, inequality (1) reduces to

$$\ell_{PP'} + \sqrt{3}\,(\,\ell_{A'P'} - \ell_{AP}\,) \;\leq\; 3\,\ell_{AA'}. \tag{2}$$

It thus remains to prove inequality (2). Let $g = \ell_{A\bar{A}}$, $z = \ell_{A'\bar{A}}$, and $v = \ell_{\bar{A}\bar{P}}$.

We first discuss the trivial case of non-intersecting L and L'. Keeping with the general notation, here we have $x = 0$ and thus $\ell_{PP'} = h$. Moreover, $g = \ell_{A\bar{A}} = h$ as well. For fixed z, we have $\ell_{AA'} = \sqrt{h^2 + z^2}$, i.e., the right hand side of (2) is fixed, whereas the left hand side is maximized if A' is on the other side of $\bar{A}$ than $\bar{P}$. The left hand side is thus at most

$$h + \sqrt{3}\,z \leq \sqrt{2}\,\sqrt{h^2 + 3z^2} \;\leq\; \sqrt{2}\,\sqrt{3\,(h^2 + z^2)} = \sqrt{6}\,\ell_{AA'} < 3\,\ell_{AA'},$$

where the first inequality follows from the power mean inequality (for powers 1 and 2), proving this easy case.

The situation when L' and L do intersect is illustrated in Figure 2. (The figure shows only the case when $\bar{A}$ is between S and P'.) Orient L' from left to right (with $\bar{P}$ being to the right of S), as shown in this figure. We want to express the distances in the above inequality in terms of s, h, v, and z (keeping in mind that x and r are functions of h and s):

$$\begin{aligned} \ell_{PP'} &= \sqrt{x^2 + h^2} \\ \ell_{AP} &= vr/s \;=\; (v\sqrt{s^2 + h^2})/s \\ \ell_{AA'} &= \sqrt{z^2 + g^2} \end{aligned}$$

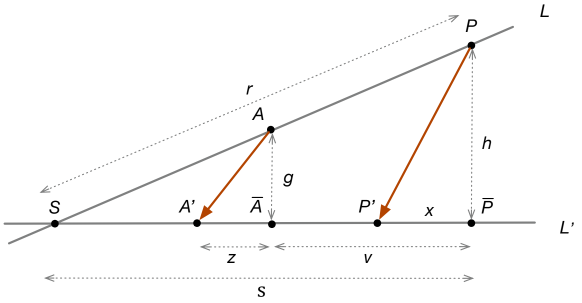


Figure 2 Notation for the analysis of Algorithm DRIFT.

The values of g and $\ell_{A'P'}$ depend on some cases, that we consider below.

Case 1. $\bar{A}$ is between S and P', as in Figure 2. Then $g = h(s-v)/s$. Our goal is first to find A' for which the bound in (2) is tightest. For a given z, among the two locations of A' at distance z from $\bar{A}$, the one on the left gives a larger value of the left-hand side of (2), while the right-hand side is the same for both. Thus we can assume that A' is to the left of $\bar{A}$, so $\ell_{A'P'} = z + v - x$. Then we can rewrite (2) as follows:

$$\tfrac{1}{\sqrt{3}}\,\ell_{PP'} - \ell_{AP} + v - x \;\le \sqrt{3}\,\sqrt{z^2+g^2} - z \tag{3}$$

By elementary calculus, the right-hand side is minimized for $z = \frac{1}{\sqrt{2}}\,g$, so we can assume that z has this value. Then inequality (3) reduces to

$$\tfrac{1}{\sqrt{3}}\,\ell_{PP'} - \ell_{AP} + v - x \;\le \sqrt{2}\,g. \tag{4}$$

After substituting $g = h(s-v)/s$ and $\ell_{AP} = vr/s$, inequality (4) reduces further to

$$s\,(\tfrac{1}{\sqrt{3}}\,\ell_{PP'} - x - \sqrt{2}h) \;\le v\,(r - s - \sqrt{2}\,h). \tag{5}$$

The expression in the parenthesis on the right-hand side of (5) is non-positive by triangle inequality, so the right-hand side is minimized when v is maximized, that is $v = s$, and then it reduces to

$$x^2 + h^2 \;\le 3(r - s + x)^2. \tag{6}$$

Recall that $x = \frac{1}{\sqrt{2}}(h + s - r)$. Since $r - h \le s \le r$, we have

$$\begin{aligned} x^2 + h^2 \;&=\; \tfrac{1}{2}(h+s-r)^2 + h^2 \\ &\le\; \tfrac{1}{2}h^2 + h^2 \\ &=\; \tfrac{3}{2}h^2 \;\le\; \tfrac{3}{2}[\,h + (\sqrt{2}-1)(r-s)\,]^2 \;=\; 3(r-s+x)^2, \end{aligned}$$

proving (6).

Case 2. $\bar{A}$ is before S. In this case we have $g = h(v-s)/s$. Just as in Case 1, we can assume that A' is to the left of $\bar{A}$, so that $\ell_{A'P'} = z + v - x$, and (2) reduces to

$$\tfrac{1}{\sqrt{3}}\,\ell_{PP'} - \ell_{AP} + v - x \;\le \sqrt{2}\,g. \tag{7}$$

After substituting $g = h(v-s)/s$ and $\ell_{AP} = vr/s$, inequality (4) reduces further to

$$s\,(\tfrac{1}{\sqrt{3}}\,\ell_{PP'} - x + \sqrt{2}h) \;\le v\,(r - s + \sqrt{2}\,h). \tag{8}$$

The expression in the parenthesis on the right-hand side of (8) is non-negative, so the right-hand side is minimized when $v = s$ (because in this case $v \geq s$), so (8) reduces to the same inequality (6) as in Case 1, completing the argument for Case 2.

Case 3. $\bar{A}$ is after $\bar{P}$. In this case we have $g = h(v+s)/s$. Symmetrically to Case 1, we can now assume that A' is to the right of $\bar{A}$, so that $\ell_{A'P'} = z + v + x$, and that $z = \frac{1}{\sqrt{2}}\, g$. Then, analogously to (4), we can rewrite (2) as follows:

$$\tfrac{1}{\sqrt{3}}\, \ell_{PP'} - \ell_{AP} + v + x \ \leq \sqrt{2}\, g \tag{9}$$

After substituting $g = h(v+s)/s$ and $\ell_{AP} = vr/s$, inequality (9) reduces further to

$$s\,(\tfrac{1}{\sqrt{3}}\, \ell_{PP'} + x - \sqrt{2}h) \ \leq v\,(r - s + \sqrt{2}\, h). \tag{10}$$

The expression in the parenthesis on the right-hand side of (10) is non-negative, so the right-hand side is minimized when $v = 0$, and then it reduces to

$$x^2 + h^2 \ \leq 3(\sqrt{2}h - x)^2. \tag{11}$$

To prove this, we proceed similarly as in Case 1:

$$x^2 + h^2 \ \leq \ \tfrac{3}{2}h^2 \ \leq \ \tfrac{3}{2}(h + r - s)^2 \ = \ 3(\sqrt{2}h - x)^2,$$

proving (11).

Case 4. $\bar{A}$ is between P' and $\bar{P}$. Then $g = h(s-v)/s$ (as in Case 1). Similar to Case 3, we can assume that A' is to the right of $\bar{A}$, so that now $\ell_{A'P'} = z - v + x$, and that $z = \frac{1}{\sqrt{2}}\, g$. Then, analogously to (4), we can rewrite (2) for this case as follows:

$$\tfrac{1}{\sqrt{3}}\, \ell_{PP'} - \ell_{AP} - v + x \ \leq \sqrt{2}\, g \tag{12}$$

After substituting $g = h(s-v)/s$ and $\ell_{AP} = vr/s$, inequality (12) reduces further to

$$s\,(\tfrac{1}{\sqrt{3}}\, \ell_{PP'} + x - \sqrt{2}h) \ \leq v\,(r + s - \sqrt{2}\, h). \tag{13}$$

We now have two sub-cases. If the expression in the parenthesis on the right-hand side of (13) is non-negative then the right-hand side is minimized when $v = 0$, so inequality (13) reduces to inequality (11) from Case 3. If this expression is negative (that is when $r + s < \sqrt{2}h$), then it is sufficient to prove (13) with v on the right-hand side replaced by s (because $v \leq s$). This reduces it to $\frac{1}{\sqrt{3}}\, \ell_{PP'} + x \leq r + s$. This last inequality follows from $\ell_{PP'} \leq r$ and $x \leq s$. ◀

3 An Algorithm for Arbitrary Dimension

In this section, we show how to extend Algorithm Drift to Euclidean spaces $\mathbb{R}^d$ for arbitrary dimension $d \geq 2$. This extension, that we call ExtDrift, is quite simple, and consists of projecting the whole space onto an appropriately chosen plane that contains the new request line. While such approach was suggested already by Friedman and Linial [11], their choice of plane may lose a constant factor in the competitive ratio. We project onto a different plane, which allows ExtDrift to also be 3-competitive.

Let P be the current ExtDrift position and L' the new request line. If $P \in L'$, ExtDrift makes no move. Otherwise, let U be the uniquely determined plane which contains both L' and P. ExtDrift makes the move prescribed by Drift in the plane U for P, L' and the projection of L onto U.

▶ **Theorem 2.** *Algorithm* ExtDrift *is 3-competitive for the line chasing problem in $\mathbb{R}^d$, for arbitrary dimension $d \geq 2$.*

Proof. We prove that (1) holds in arbitrary dimension. If $P \in L'$ then L and L' are co-planar, so the analysis from the previous section works directly.

So assume that $P \notin L'$. We first allow the adversary to perform a free move from its current position A to point $\ddot{A}$ defined as the orthogonal projection of A onto U, and then we analyze the move within U (that is, in a two-dimensional setting), as if the adversary started from point $\ddot{A}$.

We note that $\ell_{\ddot{A}X} \leq \ell_{AX}$ for any point $X \in U$, as $(\ell_{AX})^2 = (\ell_{\ddot{A}X})^2 + (\ell_{A\ddot{A}})^2$ by definition of $\ddot{A}$. It follows that:

- In the free adversary move from A to $\ddot{A}$ the potential function decreases (by taking $X = P$ in the above inequality) and both costs are 0. Further, in the move within U, with the adversary starting from $\ddot{A}$, Algorithm ExtDrift makes the same move as Drift, which implies that (1) is satisfied. Thus the complete move (combining the free adversary move and the move inside U) satisfies inequality (1) as well.
- The free move is only beneficial for the adversary: taking $X = A'$ shows that the cost of moving to A' from $\ddot{A}$ is no more costly for the adversary than moving to A' from A. ◀

4 Lower Bound for Memoryless Algorithms

We show that our algorithm achieves the optimal competitive ratio among a certain class of "memoryless" algorithms. For a metric space M, let $\mathcal{X} \subseteq \mathcal{P}(M)$ be the set of possible requests (i.e., lines in our case). In general, we can view an algorithm as a function $\mathcal{A}\colon M \times \mathcal{X}^* \to \mathcal{X}$ with $\mathcal{A}(P_0) = P_0$ and $\mathcal{A}(P_0, X_1, \ldots, X_t) \in X_t$ for each initial point $P_0 \in M$ and requests $X_1, \ldots, X_t \in \mathcal{X}$. We call an algorithm *memoryless* if $\mathcal{A}(P_0, X_1, \ldots, X_t)$ is a function of only the last position $\mathcal{A}(P_0, X_1, \ldots, X_{t-1})$, the last request X_{t-1} and the new request X_t.

However, memorylessness alone would not impose any limit on the power of line-chasing algorithms: By perturbing its positions very slightly, an algorithm could always encode the entire history in low significant bits of its current position. To get a meaningful notion of memorylessness, we therefore require an additional property, namely that the algorithm is oblivious with respect to rotation, translation or scaling of the metric space. More precisely, a *direct similarity* of $\mathbb{R}^d$ is a bijection $f\colon \mathbb{R}^d \to \mathbb{R}^d$ that is a composition of rotation, translation and scaling by some factor $r_f > 0$. In particular, for any $P, Q \in \mathbb{R}^d$, we have $\ell(f(P), f(Q)) = r_f \ell(P, Q)$. We call an algorithm $\mathcal{A}$ *rts-oblivious* if $\mathcal{A}(f(P_0), f(X_1), \ldots, f(X_t)) = f(\mathcal{A}(P_0, X_1, \ldots, X_t))$ for any $P_0 \in M$, $X_i \in \mathcal{X}$ and any direct similarity f. In general (when algorithms are allowed to use memory) there is no reason to behave differently when the input is transformed by such f, since it is just a renaming of points and scaling of distances by a uniform constant. For completeness, we provide a proof of this intuition via the following proposition:

▶ **Proposition 3.** *If there is a c-competitive algorithm for line-chasing, then there is a c-competitive rts-oblivious algorithm.*

Proof. For an initial position P_0 and request sequence $X_1, \ldots, X_t$, we assume without loss of generality that $P_0 \notin X_1$. For any such P_0 and X_1, there exists a unique direct similarity $g = g_{P_0 X_1}$ such that $g(P_0) = (0, 1)$ and $g(X_1) = \mathbb{R} \times \{0\}$. Given a c-competitive algorithm $\mathcal{A}$, we claim that the algorithm $\tilde{\mathcal{A}}$ given by

$$\tilde{\mathcal{A}}(P_0, X_1, \ldots, X_t) = g^{-1}(\mathcal{A}(g(P_0), g(X_1), \ldots, g(X_t)))$$

is rts-oblivious and c-competitive.

To see that $\tilde{\mathcal{A}}$ is rts-oblivious, consider an arbitrary direct similarity f. Notice that $g_{f(P_0)f(X_1)} = g \circ f^{-1}$. Thus,

$$\begin{aligned}\tilde{\mathcal{A}}(f(P_0), f(X_1), \dots, f(X_t)) &= (f \circ g^{-1})(\mathcal{A}(g(P_0), g(X_1), \dots, g(X_t))) \\ &= f(\tilde{\mathcal{A}}(P_0, X_1, \dots, X_t)),\end{aligned}$$

as required. To see that $\tilde{\mathcal{A}}$ is c-competitive, consider an initial position P_0 and request sequence $X_1, \dots, X_m$ along with an adversary's solution $A_0 = P_0, A_1 \in X_1, \dots, A_m \in X_m$. The cost of $\tilde{\mathcal{A}}$ can be bounded via

$$\begin{aligned}&\sum_{t=1}^{m} \ell(\tilde{\mathcal{A}}(P_0, X_1, \dots, X_{t-1}), \tilde{\mathcal{A}}(P_0, X_1, \dots, X_t)) \\ &\quad = \frac{1}{r_g} \sum_{t=1}^{m} \ell(\mathcal{A}(g(P_0), g(X_1), \dots, g(X_{t-1})), \mathcal{A}(g(P_0), g(X_1), \dots, g(X_t))) \\ &\quad \le \frac{c}{r_g} \sum_{t=1}^{m} \ell(g(A_{t-1}), g(A_t)) \\ &\quad = c \sum_{t=1}^{m} \ell(A_{t-1}, A_t),\end{aligned}$$

where the inequality uses that $\mathcal{A}$ is c-competitive against the solution $g(A_0), \dots, g(A_m)$ for the transformed input $g(P_0), g(X_1), \dots, g(X_m)$. ◀

Intuitively, an rts-oblivious algorithm does not know the absolute coordinates of its positions and requests, but only relative to each other and up to scaling. If it is memoryless, in the plane this boils down to only knowing the angle between the new and the old request line. We show now that our algorithms Drift and ExtDrift achieve the optimal competitive ratio among rts-oblivious memoryless algorithms.

▶ **Theorem 4.** *Any rts-oblivious memoryless algorithm for line-chasing has competitive ratio at least* 3.

Proof. We will construct an initial point P_0 and lines $L_0, \dots, L_m$ in $\mathbb{R}^2$ with the property that $P_0 \in L_0$ and L_t can be obtained by rotating L_{t-1} around some point $S_t \in L_{t-1}$ in clockwise direction by less than 90 degrees.

Let $P_0, \dots, P_m$ be the sequence of points visited by a given algorithm. We use notation similar to that in Figure 1: Write $\bar{P}_{t-1}$ for the orthogonal projection of P_{t-1} onto L_t and let $h_t = \ell(P_{t-1}, \bar{P}_{t-1})$ and $s_t = \ell(\bar{P}_{t-1}, S_t)$. The movement from P_{t-1} to P_t can always be viewed as first moving to $\bar{P}_{t-1}$ and then moving some distance $x_t \in \mathbb{R}$ in the direction towards intersection S_t, for a total cost $\sqrt{h_t^2 + x_t^2}$. Here, $x_t < 0$ would constitute movement away from S_t and $x_t > s_t$ would constitute movement beyond S_t.

Observe that for rts-oblivious memoryless algorithms, $\frac{x_t}{h_t}$ is a function of only $\frac{h_t}{s_t}$, i.e. $\beta(\frac{h_t}{s_t}) = \frac{x_t}{h_t}$ for some function $\beta\colon (0, \infty) \to \mathbb{R}$. Any rts-oblivious memoryless algorithm for line-chasing in the plane is uniquely determined by its associated function β as well as similar functions for the cases of counter-clockwise rotations of at most 90 degrees and parallel lines.[1] Let $\beta(0) := \limsup_{a \to 0} \beta(a) \in \mathbb{R} \cup \{-\infty, \infty\}$. Let us first show that algorithms with $\beta(0) = \infty$ or $\beta(0) \le 0$ have unbounded competitive ratio.

[1] If we require algorithms to be oblivious also with respect to reflection (which would still satisfy Proposition 3), they would be uniquely determined by β alone. Drift is the algorithm corresponding to $\beta(a) = \frac{a+1-\sqrt{a^2+1}}{\sqrt{2}a}$.

If $\beta(0) = \infty$, we choose $P_0 = (1, h)$, $L_0 = \{(x, y) \colon y = hx\}$, $L_1 = \mathbb{R} \times \{0\}$ for some small $h > 0$. The algorithm's cost is $h\sqrt{1 + \beta(h)^2}$, whereas the optimal cost is h. Choosing h arbitrarily small shows that the competitive ratio is unbounded.

If $\beta(0) \le 0$, fix some $\epsilon \in (0, \frac{1}{2}]$ and choose $a \in (0, \epsilon]$ with $\beta(a) \le \epsilon$. Let $P_0 = (1, 0)$, $L_0 = \mathbb{R} \times \{0\}$ and define L_t as the clockwise rotation of L_{t-1} around the origin $O := (0, 0)$ by angle $\arctan(a)$. Thus, we have $\frac{h_t}{s_t} = a$ for each t. Notice that $s_t\sqrt{1 + a^2} = \ell(O, P_{t-1}) = s_{t-1}(1 - a\beta(a))$, and therefore

$$\frac{s_t}{s_{t-1}} = \frac{1 - a\beta(a)}{\sqrt{1 + a^2}} \ge 1 - \frac{a^2 + a\beta(a)}{1 + a^2} \ge 1 - 2\epsilon a,$$

where the first inequality uses $\sqrt{1 + a^2} \le 1 + a^2$ and the second inequality uses $0 < a \le \epsilon$ and $\beta(a) \le \epsilon$. Hence,

$$s_t \ge (1 - 2\epsilon a)^{t-1}$$

Since $\ell(P_{t-1}, P_t) \ge h_t = as_t$, the total cost of the algorithm is

$$\sum_{t=1}^{m} \ell(P_{t-1}, P_t) \ge a \sum_{t=0}^{m-1} (1 - 2\epsilon a)^t \xrightarrow{m \to \infty} \frac{1}{2\epsilon}.$$

Meanwhile, an optimal algorithm pays total cost 1 by moving to O immediately. Letting $\epsilon \to 0$, we find again that the competitive ratio is unbounded.

It remains to consider the case $0 < \beta(0) < \infty$. Then we can choose arbitrarily small $a > 0$ such that $0 < a\beta(a) < 1$. We choose the initial point $P_0 = (\frac{1}{a}, 1)$, and the request sequence starts with $L_0 = \{(x, y) \colon y = ax\}$ and $L_1 = \mathbb{R} \times \{0\}$. For $t \ge 2$, we define L_t as the clockwise rotation of L_{t-1} around $S_t = S_2 = \left(\frac{1}{a} - \beta(a) + \sqrt{1 + a^2}\left(\beta(a) + \frac{1}{2\beta(a)}\right), 0\right)$ by angle $\arctan(a)$. The idea is that in response to L_1, the algorithm drifts to the left (towards intersection $S_1 = (0, 0)$), but the subsequent requests are such that it would have been cheaper to drift to the right (away from S_1) instead.

We have $s_1 = \frac{1}{a}$ and $s_2 = \frac{\ell(P_1, S_2)}{\sqrt{1+a^2}} = \beta(a) + \frac{1}{2\beta(a)}$. For $t \ge 3$, similarly to the previous case we get

$$\frac{s_t}{s_{t-1}} \ge 1 - \frac{a^2 + a\beta(a)}{1 + a^2} \ge 1 - a^2 - a\beta(a)$$

and therefore

$$s_t \ge \left(\beta(a) + \frac{1}{2\beta(a)}\right)\left(1 - a^2 - a\beta(a)\right)^{t-2} \qquad \text{if } t \ge 2.$$

As $m \to \infty$, the cost of the algorithm is

$$\begin{aligned}\sum_{t=1}^{\infty} \ell(P_{t-1}, P_t) &= \sum_{t=1}^{\infty} h_t\sqrt{1 + \beta(a)^2} = \sqrt{1 + \beta(a)^2}a \sum_{t=1}^{\infty} s_t \\ &\ge \sqrt{1 + \beta(a)^2}\left(1 + \left(\beta(a) + \frac{1}{2\beta(a)}\right)\frac{1}{a + \beta(a)}\right) \\ &\xrightarrow{a \to 0} \sqrt{1 + \beta(0)^2}\left(2 + \frac{1}{2\beta(0)^2}\right),\end{aligned}$$

where the limit $a \to 0$ is taken along a sequence where $\beta(a) \to \beta(0)$. In contrast, an offline algorithm can move immediately from P_0 to S_2, paying cost $\sqrt{1 + \frac{1}{4\beta(0)^2}}$ as $a \to 0$ and

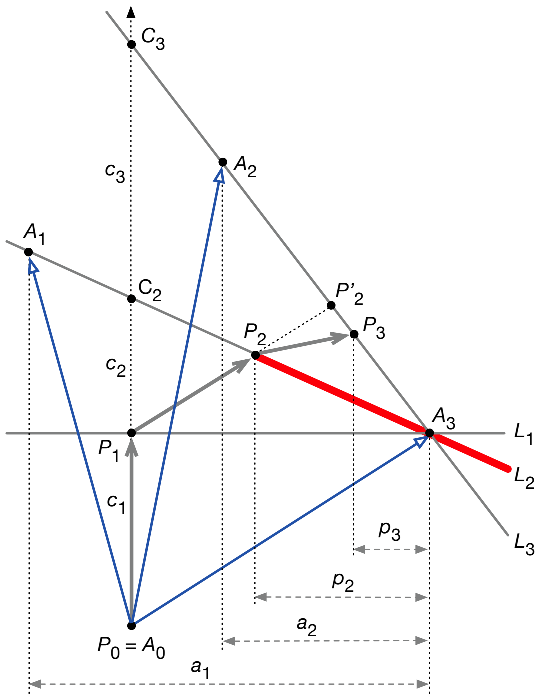


Figure 3 Visual description of our lower bound for arbitrary algorithms. Lines L_1, L_2 and L_3 are presented to an online algorithm. Blue arrows describe possible movements of Opt, while gray thick arrows describe a path of an algorithm that minimizes the competitive ratio for this adversarial construction. Red thick half-line denotes the forbidden region.

$\beta(a) \to \beta(0)$. By dividing, we see that the competitive ratio is at least

$$\sqrt{(1+\beta(0)^2)\left(4+\frac{1}{\beta(0)^2}\right)} = \sqrt{4\beta(0)^2 + \frac{1}{\beta(0)^2} + 5},$$

which is minimized for $\beta(0) = \frac{1}{\sqrt{2}}$, taking value 3. ◀

5 Lower Bound for Arbitrary Algorithms

Finally, in this section, we show how to improve an existing lower bound of $\sqrt{2} \approx 1.41$ for arbitrary algorithms to 1.5358. Our bound holds even in two dimensions, and improves also the lower bound for the more general convex body chasing in two dimensions.

▶ **Theorem 5.** *The competitive ratio of any deterministic online algorithm $\mathcal{A}$ for the line chasing problem is at least* 1.5358*.*

Proof. We describe our adversarial strategy below. On the created input, we will compare the cost of $\mathcal{A}$ to the cost of an offline optimum Opt. We assume that both $\mathcal{A}$ and Opt start at origin point $P_0 = A_0 = (0, 0)$.

Our construction is parameterized with real positive numbers $c_1 = 0.5535$, $c_2 = 0.4965$, $c_3 = 0.8743$, $a_1 = 1.3012$, $a_2 = 0.6663$, $p_2 = 0.5612$, and $p_3 = 0.1696$.

We fix points $P_1 = (0, c_1)$, $C_2 = (0, c_1 + c_2)$, $C_3 = (0, c_1 + c_2 + c_3)$ and $A_3 = (1, c_1)$, see Figure 3 for illustration. For succinctness, we use notation $\triangle(x, y) = \sqrt{x^2 + y^2}$.

Initial part: Line L_1

The first request line is the line P_1A_3, denoted L_1. Without loss of generality, we can assume that $\mathcal{A}$ moves to point P_1. This is because the adversary can either play the strategy described below or its mirror image (flipped against the line P_0P_1), so any deviation from P_1, either to the left or right, can only increase the cost of $\mathcal{A}$.

From now on, for any point Q we denote its projection on line L_1 by Q^x.

Middle part: Line L_2

Next, the adversary issues the request line C_2A_3, denoted L_2. Let $P_2 \in L_2$ and $A_1 \in L_2$ be the points to the left of A_3, such that $\ell_{P_2^x A_3} = p_2$ and $\ell_{A_1^x A_3} = a_1$.

Let $\bar{P}_2$ be the point on L_2 chosen by $\mathcal{A}$. If $\bar{P}_2$ lies to the right of point P_2, then the adversary forces $\mathcal{A}$ to move to A_1 (by giving sufficiently many different lines that go through A_1 at different angles). Opt may then serve the whole sequence by going from A_0 to A_1 at cost

$$\ell_{A_0A_1} = \triangle(c_1 + c_2 \cdot a_1,\, a_1 - 1) \leq 1.23679$$

while the cost of $\mathcal{A}$ is then at least

$$\begin{aligned}\ell_{P_0P_1} + \ell_{P_1P_2} + \ell_{P_2A_1} &= \ell_{P_0P_1} + \ell_{P_1P_2} + \ell_{A_3A_1} - \ell_{A_3P_2} \\ &= c_1 + \triangle(1 - p_2,\, c_2 \cdot p_2) + \triangle(a_1,\, c_2 \cdot a_1) - \triangle(p_2,\, c_2 \cdot p_2) \\ &\geq 1.89948\end{aligned}$$

Hence, the competitive ratio in this case is at least 1.5358.

We call the half-line of L_2 to the right of point P_2 *forbidden region*. From now on, we assume that the point chosen by $\mathcal{A}$ in L_2 does not lie in this region.

Final part: Line L_3

Finally, the adversary issues the request line C_3A_3, denoted L_3. Let P_2' be the intersection of line P_1P_2 with line L_3. Next, let A_2 and P_3 be the points on the line L_3 to the left of A_3, such that $\ell_{A_2^x A_3} = a_2$ and $\ell_{P_3^x A_3} = p_3$. Note that P_3 belongs to the interval $P_2'A_3$.

Let $\bar{P}_3$ be the point on L_3 chosen by $\mathcal{A}$. We consider two cases.

Case 1. $\bar{P}_3$ lies at point P_3 or to its left. In this case, the adversary forces $\mathcal{A}$ to move to A_3. Opt may serve the whole sequence by going from A_0 to A_3 paying

$$\ell_{A_0A_3} = \triangle(1,\, c_1) \leq 1.142963.$$

We may now argue that the cost of $\mathcal{A}$ is minimized if $\bar{P}_3$ is equal to P_3: If $\bar{P}_3$ is to the left of point P_2', then the cost of $\mathcal{A}$ is at least $\ell_{P_0P_1} + \ell_{P_1\bar{P}_3} + \ell_{\bar{P}_3A_3}$. Both the second and the third summand decrease when we move $\bar{P}_3$ towards P_2'. Hence, now we may assume that $\bar{P}_3$ belongs to the interval $P_2'P_3$. As the path of $\mathcal{A}$ must avoid forbidden region, its cost is at least $\ell_{P_0P_1} + \ell_{P_1P_2} + \ell_{P_2\bar{P}_3} + \ell_{\bar{P}_3A_3}$. The sum of the last two summands decreases when we move $\bar{P}_3$ towards P_3. Therefore, we obtain that the cost of $\mathcal{A}$ is at least

$$\begin{aligned}&\ell_{P_0P_1} + \ell_{P_1P_2} + \ell_{P_2P_3} + \ell_{P_3A_3} \\ &\quad = c_1 + \triangle(1 - p_2,\, c_2 \cdot p_2) + \triangle((c_2 + c_3) \cdot p_3 - c_2 \cdot p_2,\, p_2 - p_3) \\ &\qquad + \triangle(p_3,\, (c_2 + c_3) \cdot p_3) \geq 1.75537.\end{aligned}$$

Thus, in this case the competitive ratio is at least 1.5358.

Case 2. If $\bar{P}_3$ lies to the right of point P_3, then the adversary forces $\mathcal{A}$ to move to A_2. Opt may serve the whole sequence by going from A_0 to A_2 at cost

$$\ell_{A_0A_2} = \triangle(c_1 + (c_2 + c_3) \cdot a_2,\, 1 - a_2) \le 1.50435.$$

To go from P_1 to $\bar{P}_3$ and avoid the forbidden region, $\mathcal{A}$ has to pay at least $\ell_{P_1P_2} + \ell_{P_2\bar{P}_3}$. Therefore, its cost is at least

$$\begin{aligned}\ell_{P_0P_1} &+ \ell_{P_1P_2} + \ell_{P_2\bar{P}_3} + \ell_{\bar{P}_3A_2}\\ &\ge \ell_{P_0P_1} + \ell_{P_1P_2} + \ell_{P_2P_3} + \ell_{P_3A_2}\\ &\ge \ell_{P_0P_1} + \ell_{P_1P_2} + \ell_{P_2P_3} + \ell_{A_2A_3} - \ell_{P_3A_3}\\ &= c_1 + \triangle(1 - p_2,\, c_2 \cdot p_2) + \triangle((c_2 + c_3) \cdot p_3 - c_2 \cdot p_2,\, p_2 - p_3)\\ &\qquad + \triangle(a_2,\, (c_2 + c_3) \cdot a_2) - \triangle(p_3,\, (c_2 + c_3) \cdot p_3) \ge 2.31039.\end{aligned}$$

Thus, in this case the ratio is also at least 1.5358. ◀

6 Final Comments

Establishing the optimal competitive ratio for line chasing with memory remains an open problem. We believe that with memory, a competitive ratio better than 3 is achievable.

The intuition is that in the first move, if L and P are the initial line and position and L' is the new request line, then the algorithm should move to the nearest point $\bar{P}$ on L'. More generally, if the requests on L and L' alternate (and their angle is small), the algorithm should initially drift slowly towards $S = L \cap L'$ and only gradually accelerate as it becomes more credible that the adversary is located at S. To gauge this credibility for general request sequences, an algorithm might store the current work function at each step.

It appears also that our lower bound of 1.5358 can be improved by introducing additional steps, although this gives only very small improvements and leads to a very involved analysis. It is possible that an approach fundamentally different from ours may give a better bound with simpler analysis.

References

1 Antonios Antoniadis, Neal Barcelo, Michael Nugent, Kirk Pruhs, Kevin Schewior, and Michele Scquizzato. Chasing Convex Bodies and Functions. In *Proc. 12th Latin American Theoretical Informatics Symposium (LATIN)*, pages 68–81, 2016. doi:10.1007/978-3-662-49529-2_6.

2 C. J. Argue, Sébastien Bubeck, Michael B. Cohen, Anupam Gupta, and Yin Tat Lee. A Nearly-Linear Bound for Chasing Nested Convex Bodies. In *Proc. 30th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 117–122, 2019.

3 Nikhil Bansal, Martin Böhm, Marek Eliás, Grigorios Koumoutsos, and Seeun William Umboh. Nested Convex Bodies are Chaseable. In *Proc. 29th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1253–1260, 2018. doi:10.1137/1.9781611975031.81.

4 Nikhil Bansal, Anupam Gupta, Ravishankar Krishnaswamy, Kirk Pruhs, Kevin Schewior, and Clifford Stein. A 2-Competitive Algorithm For Online Convex Optimization With Switching Costs. In *Proc. Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 96–109, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.96.

5 Allan Borodin, Nathan Linial, and Michael E. Saks. An Optimal On-Line Algorithm for Metrical Task System. *J. ACM*, 39(4):745–763, 1992. doi:10.1145/146585.146588.

6 Sébastien Bubeck, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Chasing Nested Convex Bodies Nearly Optimally. *CoRR*, abs/1811.00999, 2018. URL: http://arxiv.org/abs/1811.00999.

7 Sébastien Bubeck, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Competitively chasing convex bodies. In *Proc. 51st ACM Symp. on Theory of Computing (STOC)*, pages 861–868, 2019. doi:10.1145/3313276.3316314.

8 William R. Burley. Traversing Layered Graphs Using the Work Function Algorithm. *J. Algorithms*, 20(3):479–511, 1996. doi:10.1006/jagm.1996.0024.

9 Marek Chrobak and Lawrence L. Larmore. Metrical Task Systems, the Server Problem and the Work Function Algorithm. In *Online Algorithms, The State of the Art (Proc. Dagstuhl Seminar, June 1996)*, pages 74–96, 1996. doi:10.1007/BFb0029565.

10 Amos Fiat, Dean P. Foster, Howard J. Karloff, Yuval Rabani, Yiftach Ravid, and Sundar Vishwanathan. Competitive Algorithms for Layered Graph Traversal. *SIAM J. Comput.*, 28(2):447–462, 1998. doi:10.1137/S0097539795279943.

11 Joel Friedman and Nathan Linial. On Convex Body Chasing. *Discrete & Computational Geometry*, 9:293–321, 1993. doi:10.1007/BF02189324.

12 Elias Koutsoupias and Christos H. Papadimitriou. On the k-Server Conjecture. *J. ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.

13 Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. Dynamic Right-Sizing for Power-Proportional Data Centers. *IEEE/ACM Trans. Netw.*, 21(5):1378–1391, 2013. doi:10.1109/TNET.2012.2226216.

14 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive Algorithms for Server Problems. *J. Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.

15 H. Ramesh. On Traversing Layered Graphs On-Line. *J. Algorithms*, 18(3):480–512, 1995. doi:10.1006/jagm.1995.1019.

16 René Sitters. The Generalized Work Function Algorithm Is Competitive for the Generalized 2-Server Problem. *SIAM J. Comput.*, 43(1):96–125, 2014. doi:10.1137/120885309.

Nash Equilibria in Games over Graphs Equipped with a Communication Mechanism

Patricia Bouyer
LSV, CNRS, ENS Paris-Saclay, Université Paris-Saclay, France
bouyer@lsv.fr

Nathan Thomasset
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
nathan.thomasset@ens-paris-saclay.fr

Abstract

We study pure Nash equilibria in infinite-duration games on graphs, with partial visibility of actions but communication (based on a graph) among the players. We show that a simple communication mechanism consisting in reporting the deviator when seeing it and propagating this information is sufficient for characterizing Nash equilibria. We propose an epistemic game construction, which conveniently records important information about the knowledge of the players. With this abstraction, we are able to characterize Nash equilibria which follow the simple communication pattern via winning strategies. We finally discuss the size of the construction, which would allow efficient algorithmic solutions to compute Nash equilibria in the original game.

2012 ACM Subject Classification Theory of computation; Theory of computation → Solution concepts in game theory; Theory of computation → Verification by model checking

Keywords and phrases Multiplayer games, Nash equilibria, partial information

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.9

Related Version Full version available at `http://arxiv.org/abs/1906.07753`

Funding This work has been partly supported by ERC project EQualIS (FP7-308087).

1 Introduction

Multiplayer concurrent games over graphs allow to model rich interactions between players. Those games are played as follows. In a state, each player chooses privately and independently an action, defining globally a move (one action per player); the next state of the game is then defined as the successor (on the graph) of the current state using that move; players continue playing from that new state, and form a(n infinite) play. Each player then gets a reward given by a payoff function (one function per player). In particular, objectives of the players may not be contradictory: those games are non-zero-sum games, contrary to two-player games used for controllers or reactive synthesis [15, 12].

Using solution concepts borrowed from game theory, one can describe interactions among the players, and in particular rational behaviours of selfish players. One of the most basic and classically studied solution concepts is that of Nash equilibria [13]. A Nash equilibrium is a strategy profile where no player can improve her payoff by unilaterally changing her strategy. The outcome of a Nash equilibrium can therefore be seen as a rational behaviour of the system. While very much studied by game theorists, e.g. over (repeated) matrix games, such a concept (and variants thereof) has been only rather recently studied over infinite-duration games on graphs. Probably the first works in that direction are [9, 8, 16, 17]. Several series of works have followed. To roughly give an idea of the existing results, pure Nash equilibria always exist in turn-based games for ω-regular objectives [19] but not in concurrent games, even with simple objectives; they can nevertheless be computed [19, 4, 7, 3] for large classes

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 9; pp. 9:1–9:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of objectives. The problem becomes harder with mixed (that is, stochastic) Nash equilibria, for which we often cannot decide the existence [18, 5].

Computing Nash equilibria requires to (i) find a good behaviour of the system; (ii) detect deviations from that behaviour, and identify deviating players; (iii) punish them. This simple characterization of Nash equilibria is made explicit in [10]. Variants of Nash equilibria require slightly different ingredients, but they are mostly of a similar vein.

Many of those results are proven using the construction of a two-player game, in which winning strategies correspond (in some precise sense) to Nash equilibria in the original game. This two-player game basically records the knowledge of the various players about everything which can be uncertain: (a) the possible deviators in [4], and (b) the possible states the game can be in [3]. Extensions of this construction can be used for other solution concepts like robust equilibria [7] or rational synthesis [11].

In this work, we consider infinite-duration games on graphs, in which the game arena is perfectly known by all the players, but players have only a partial information on the actions played by the other players. The partial-information setting of this work is inspired by [14]: it considers repeated games played on matrices, where players only see actions of their neighbours. Neighbours are specified by a communication graph. To ensure a correct detection of deviators, the solution is to propagate the identity of the deviator along the communication graph. A fingerprint (finite sequence of actions) of every player is agreed at the beginning, and the propagation can be made properly if and only if the communication graph is 2-connected, ensuring large sets of Nash equilibria (formalized as a folk theorem). Fingerprints are not adapted to the setting of graphs, since they may delay the time at which a player will learn the identity of the deviator, which may be prohibitive if a bad component of a graph is then reached.

We therefore propose to add real communication among players. Similarly to [14], a player can communicate only with her neighbours (also specified by a communication graph), but can send arbitrary messages (modelled as arbitrary words over alphabet $\{0, 1\}$). We assume that visited states are known by the players, hence only the deviator (if any) may be unknown to the players. In this setting, we show the following results:

- We show that a very simple epidemic-like communication mechanism is sufficient for defining Nash equilibria. It consists in (a) reporting the deviator (for the neighbours of the deviator) as soon as it is detected, and (b) propagating this information (for the other players).
- We build an epistemic game, which tracks those strategy profiles which follow the above simple communication pattern. This is a two-player turn-based game, in which `Eve` (the first player) suggest moves, and `Adam` (the second player) complies (to generate the main outcome), or not (to mimic single-player deviations). The correctness of the construction is formulated as follows: there is a Nash equilibrium in the original game of payoff p if and only if there is a strategy for `Eve` in the epistemic game which is winning for p.
- We analyze the complexity of this construction.

Note that we do not assume connectedness of the communication graph, hence the particular case of a graph with no edges allows to recover the setting of [4] while a complete graph allows to recover the settings of [19, 7].

In Section 2, we define our model and give an example to illustrate the role of the communication graph. In Section 3, we prove the simple communication pattern. In Section 4, we construct the epistemic game and discuss its correctness. In Section 5, we discuss complexity issues. All proofs are available in the technical report [6].

2 Definitions

We use the standard notations $\mathbb{R}$ (resp. $\mathbb{Q}$, $\mathbb{N}$) for the set of real (resp. rational, natural) numbers. If S is a subset of $\mathbb{R}$, we write $\overline{S}$ for $S \cup \{-\infty, +\infty\}$.

Let S be a finite set and $R \subseteq S$. If m is an S-vector over some set Σ, we write $m(R)$ (resp. $m(-R)$) for the vector composed of the R-components of m (resp. all but the R-components). We also use abusively the notations $m(i)$ (resp. $m(-i)$) when i is a single element of S, and may sometimes even use m_i if this is clear in the context. Also, if $s \in S$ and $a \in \Sigma$, then $m[s/a]$ is the vector where the value $m(s)$ is replaced by a.

If S is a finite set, we write S^* (resp. S^+, S^ω) for the set of words (resp. non-empty word, infinite words) defined on alphabet S.

2.1 Concurrent games and communication graphs

We use the model of concurrent multi-player games [4], based on the two-player model of [1].

▶ **Definition 1.** *A* concurrent multiplayer game *is a tuple* $\mathcal{G} = \langle V, v_{\mathsf{init}}, \mathsf{Act}, \mathcal{P}, \Sigma, \mathsf{Allow}, \mathsf{Tab}, (\mathsf{payoff}_a)_{a\in\mathcal{P}}\rangle$*, where* V *is a finite set of vertices,* $v_{\mathsf{init}} \in V$ *is the initial vertex,* Act *is a finite set of actions,* $\mathcal{P}$ *is a finite set of players,* Σ *is a finite alphabet,* $\mathsf{Allow}\colon V \times \mathcal{P} \to 2^{\mathsf{Act}} \setminus \{\emptyset\}$ *is a mapping indicating the actions available to a given player in a given vertex,*[1] $\mathsf{Tab}\colon V \times \mathsf{Act}^{\mathcal{P}} \to V$ *associates, with a given vertex and a given action tuple the target vertex, for every* $a \in \mathcal{P}$*,* $\mathsf{payoff}_a\colon V^\omega \to \mathbb{D}$ *is a payoff function with values in a domain* $\mathbb{D} \subseteq \overline{\mathbb{R}}$.

An element of $\mathsf{Act}^{\mathcal{P}}$ is called a *move*. Standardly (see [1] for two-player games and [4] for the multiplayer extension), concurrent games are played as follows: from a given vertex v, each player selects independently an action (allowed by Allow), which altogether form a move m; then, the game proceeds to the next vertex, given by $\mathsf{Tab}(v, m)$; the game continues from that new vertex.

Our setting will refine this model, in that at each round, each player will also broadcast a message, which will be received by some of the players. The players that can receive a message will be specified using a communication graph that we will introduce later. The role of the messages will remain unclear until we commit to the definition of a strategy.

Formally, a *full history* h in $\mathcal{G}$ is a finite sequence

$$v_0 \cdot (m_0, \mathsf{mes}_0) \cdot v_1 \cdot (m_1, \mathsf{mes}_1) \cdot v_2 \dots (m_{s-1}, \mathsf{mes}_{s-1}) \cdot v_s \in V \cdot \Big(\big(\mathsf{Act}^{\mathcal{P}} \times (\{0,1\}^*)^{\mathcal{P}}\big) \cdot V\Big)^*$$

such that for every $0 \le r < s$, for every $a \in \mathcal{P}$, $m_r(a) \in \mathsf{Allow}(v_r, a)$, and $v_{r+1} = \mathsf{Tab}(v_r, m_r)$. For every $0 \le r < s$, for every $a \in \mathcal{P}$, the set $\mathsf{mes}_r(a)$ is the message appended to action $m_r(a)$ at step $r+1$, which will be broadcast to some other players. For readability we will also write h as

$$v_0 \xrightarrow{m_0, \mathsf{mes}_0} v_1 \xrightarrow{m_1, \mathsf{mes}_1} v_2 \dots \xrightarrow{m_{s-1}, \mathsf{mes}_{s-1}} v_s$$

We write $\mathit{vertices}(h) = v_0 \cdot v_1 \cdots v_s$, and $\mathit{last}(h)$ for the last vertex of h (that is, v_s). If $r \le s$, we also write $h_{\ge r}$ (resp. $h_{\le r}$) for the suffix $v_r \cdot (m_r, \mathsf{mes}_r) \cdot v_{r+1} \cdot (m_{r+1}, \mathsf{mes}_{r+1}) \dots (m_{s-1}, \mathsf{mes}_{s-1}) \cdot v_s$ (resp. prefix $v_0 \cdot (m_0, \mathsf{mes}_0) \cdot v_1 \cdot (m_1, \mathsf{mes}_1) \dots (m_{r-1}, \mathsf{mes}_{r-1}) \cdot v_r$). We write $\mathsf{Hist}_{\mathcal{G}}(v_0)$ (or simply $\mathsf{Hist}(v_0)$ if $\mathcal{G}$ is clear in the context) for the set of full histories in $\mathcal{G}$ that start at v_0. If $h \in \mathsf{Hist}(v_0)$ and $h' \in \mathsf{Hist}(\mathit{last}(h))$, then we write $h \cdot h'$ for the obvious concatenation of histories (it then belongs to $\mathsf{Hist}(v_0)$).

[1] This condition ensures that the game is non-blocking.

We add a *communication (directed) graph* $G = (\mathcal{P}, E)$ to the context. The set of vertices of G is the set of players, and edges define a neighbourhood relation. An edge $(a,b) \in E$ (with $a \neq b$) means that player b can see which actions are played by player a together with the messages broadcast by player a. Later we write $a \mathrel{-\!\!\triangleright} b$ whenever $(a,b) \in E$ or $a = b$, and $\mathbb{n}(b) = \{a \in \mathcal{P} \mid a \mathrel{-\!\!\triangleright} b\}$ for the so-called *neighbourhood* of b (that is, the set of players about which player b has information). If $a, b \in \mathcal{P}$, we write $\mathsf{dist}_G(a,b)$ for the distance in G from a to b ($+\infty$ if there is no path from a to b).

Let $a \in \mathcal{P}$ be a player. The projection of h for a is denoted $\pi_a(h)$ and is defined by

$$v_0 \cdot (m_0(\mathbb{n}(a)), \mathsf{mes}_0(\mathbb{n}(a))) \cdot v_1 \cdot (m_1(\mathbb{n}(a)), \mathsf{mes}_1(\mathbb{n}(a))) \cdot v_2 \dots$$
$$\dots (m_{s-1}(\mathbb{n}(a)), \mathsf{mes}_{s-1}(\mathbb{n}(a))) \cdot v_s \in V \cdot \Big(\Big(\mathsf{Act}^{\mathbb{n}(a)} \times (\{0,1\}^*)^{\mathbb{n}(a)}\Big) \cdot V\Big)^*$$

This will be the information available to player a. In particular, messages broadcast by the players are part of this information. Note that we assume perfect recall, that is, while playing, player a will remember all her past knowledge, that is, all of $\pi_a(h)$ if h has been played so far. We define the *undistinguishability relation* $\sim_a$ as the equivalence relation over full histories induced by π_a: for two histories h and h', $h \sim_a h'$ iff $\pi_a(h) = \pi_a(h')$. While playing, if $h \sim_a h'$, a will not be able to know whether h or h' has been played. We write $\mathsf{Hist}_{\mathcal{G},a}(v_0)$ for the set of histories for player a (also called a-histories) from v_0.

We extend all the above notions to infinite sequences in a straightforward way and to the notion of *full play*. We write $\mathsf{Plays}_{\mathcal{G}}(v_0)$ (or simply $\mathsf{Plays}(v_0)$ if $\mathcal{G}$ is clear in the context) for the set of full plays in $\mathcal{G}$ that start at v_0.

A *strategy* for a player $a \in \mathcal{P}$ from vertex v_0 is a mapping $\sigma_a \colon \mathsf{Hist}_{\mathcal{G}}(v_0) \to \mathsf{Act} \times \{0,1\}^*$ such that for every history $h \in \mathsf{Hist}_{\mathcal{G}}(v_0)$, $\sigma_a(h)[1] \in \mathsf{Allow}(\mathit{last}(h), a)$, where the notation $\sigma_a(h)[1]$ denotes the first component of the pair $\sigma(h)$. The value $\sigma_a(h)[1]$ represents the action that player a will do after h, while $\sigma_a(h)[2]$ is the message that she will append to her action and broadcast to all players b such that $a \mathrel{-\!\!\triangleright} b$. The strategy σ_a is said *G-compatible* if furthermore, for all histories $h, h' \in \mathsf{Hist}(v_0)$, $h \sim_a h'$ implies $\sigma_a(h) = \sigma_a(h')$. In that case, σ_a can equivalently be seen as a mapping $\mathsf{Hist}_{\mathcal{G},a}(v_0) \to \mathsf{Act} \times \{0,1\}^*$. An *outcome* of σ_a is a(n infinite) play $\rho = v_0 \cdot (m_0, \mathsf{mes}_0) \cdot v_1 \cdot (m_1, \mathsf{mes}_1) \dots$ such that for every $r \geq 0$, $\sigma_a(\rho_{\leq r}) = (m_r(a), \mathsf{mes}_r(a))$. We write $\mathsf{out}(\sigma_a, v_0)$ for the set of outcomes of σ_a from v_0.

A *strategy profile* is a tuple $\sigma_{\mathcal{P}} = (\sigma_a)_{a \in \mathcal{P}}$, where, for every player $a \in \mathcal{P}$, σ_a is a strategy for player a. The strategy profile is said *G-compatible* whenever each σ_a is G-compatible. We write $\mathsf{out}(\sigma_{\mathcal{P}}, v_0)$ for the unique full play from v_0, which is an outcome of all strategies part of $\sigma_{\mathcal{P}}$.

When $\sigma_{\mathcal{P}}$ is a strategy profile and σ'_d a player-d G-compatible strategy, we write $\sigma_{\mathcal{P}}[d/\sigma'_d]$ for the profile where player d plays according to σ'_d, and each other player a ($\neq d$) plays according to σ_a. The strategy σ'_d is a *deviation* of player d, or a *d-deviation* w.r.t. $\sigma_{\mathcal{P}}$. Such a d-deviation is said *profitable* w.r.t. $\sigma_{\mathcal{P}}$ whenever $\mathsf{payoff}_d\Big(\mathit{vertices}(\mathsf{out}(\sigma_{\mathcal{P}}, v_0))\Big) < \mathsf{payoff}_d\Big(\mathit{vertices}(\mathsf{out}(\sigma_{\mathcal{P}}[d/\sigma'_d], v_0))\Big)$.

▶ **Definition 2.** *A* Nash equilibrium *from v_0 is a G-compatible strategy profile $\sigma_{\mathcal{P}}$ such that for every $d \in \mathcal{P}$, there is no profitable d-deviation w.r.t. $\sigma_{\mathcal{P}}$.*

In this definition, deviation σ'_d needs not really to be G-compatible, since the only meaningful part of σ'_d is along $\mathsf{out}(\sigma[d/\sigma'_d], v_0)$, where there are no $\sim_d$-equivalent histories: any deviation can be made G-compatible without affecting the profitability of the resulting outcome.

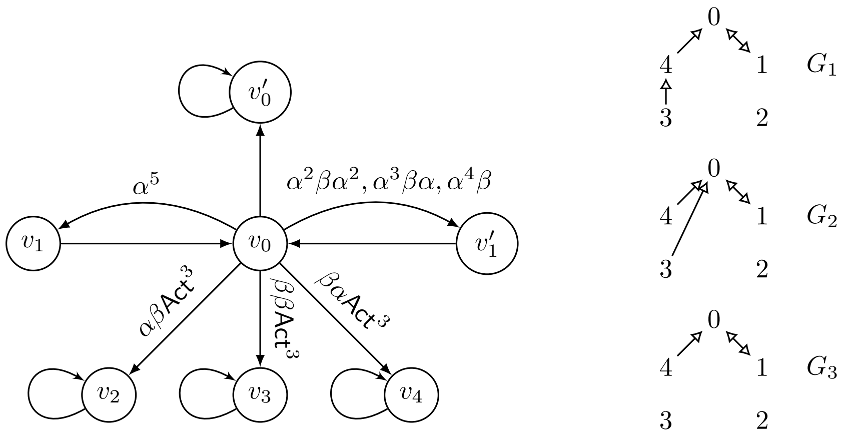

v_0'
α^5
$\alpha^2\beta\alpha^2, \alpha^3\beta\alpha, \alpha^4\beta$
v_1
v_0
v_1'
$\alpha\beta\mathsf{Act}^3$
$\beta\beta\mathsf{Act}^3$
$\beta\alpha\mathsf{Act}^3$
v_2
v_3
v_4
0
4
1
3
2
G_1
G_2
G_3

Figure 1 A five-player game (left) and three communication graphs (right); self-loops $a \mathrel{-\!\triangleright} a$ omitted from the picture. The action alphabet is $\{\alpha, \beta\}$. The transition function is represented as arrows from one vertex to another labeled with the action profile(s) allowing to go from the origin vertex to the destination one. We write action profiles with length-5 words. Convention: no label means complementary labels (e.g. one goes from v_0 to v_0' using any action profile that is not in $\alpha^5 + (\alpha\beta + \beta\beta + \beta\alpha)\mathsf{Act}^3 + \alpha^2(\beta\alpha^2 + \alpha\beta\alpha + \alpha^2\beta)$).

▶ Remark 3. Before pursuing our study, let us make clear what information players have: a player knows the full arena of the game and the whole communication graph; when playing the game, a player sees the states which are visited, and see actions of and messages from her neighbours (in the communication graph). When playing the profile of a Nash equilibrium, all players know all strategies, hence a player knows precisely what is expected to be the main outcome; in particular, when the play leaves the main outcome, each player knows that a deviation has occurred, even though she didn't see the deviator or received any message. Note that deviations which do not leave the main outcome may occur; in this case, only the neighbours of the deviator will know that such a deviation occurred; we will see that it is useless to take care of such deviations.

2.2 An example

We consider the five-player game described in Figure 1 in which we denote the players $\mathscr{P} = \{0, 1, 2, 3, 4\}$. The action alphabet is $\mathsf{Act} = \{\alpha, \beta\}$, and the initial vertex is assumed to be v_0. We suppose the payoff function vector is defined as (to be read as the list of payoffs of the players):

$$\mathsf{payoff}(\rho) = \begin{cases} (0,0,1,1,1) & \text{if } \rho \text{ visits } v_1 \text{ infinitely often} \\ (0,0,2,2,2) & \text{if } \rho \text{ visits } v_1 \text{ finitely often and } v_1' \text{ infinitely often} \\ (0,0,0,2,2) & \text{if } \rho \text{ ends up in } v_2 \\ (0,0,2,0,2) & \text{if } \rho \text{ ends up in } v_3 \\ (0,0,2,2,0) & \text{if } \rho \text{ ends up in } v_4 \\ (0,0,3,3,3) & \text{if } \rho \text{ ends up in } v_0' \end{cases}$$

We consider a (partial) strategy profile σ whose main outcome is:

$$\rho = \big(v_0 \cdot (\alpha^5, \mathsf{mes}_\epsilon) \cdot v_1 \cdot (\alpha^5, \mathsf{mes}_\epsilon)\big)^\omega$$

where $\mathsf{mes}_\epsilon(a) = \epsilon$ for every $a \in \mathscr{P}$. Note that players 0 and 1 cannot benefit from any deviation since their payoffs is uniformly 0. Then notice that no one alone can deviate from

ρ to v'_0. Now, the three players 2, 3 and 4 can alone deviate to v'_1 and (try to) do so infinitely often. We examine those deviations. If players 0 and 1 manage to learn who is the deviator, then, together, they can punish the deviator: if they learn that player 2 (resp. 3, 4) is the deviator, then they will enforce vertex v_2 (resp. v_3, v_4). If they do not manage to learn who is the deviator, then they will not know what to do, and therefore, in any completion of the strategy profile, there will be some profitable deviation for at least one of the players (hence there will not be any Nash equilibrium whose main outcome is ρ).

We examine now the three communication graphs G_1, G_2 and G_3 depicted in Figure 1. Using communication based on graph G_1, if player 4 deviates, then player 0 will see this immediately and will be able to communicate this fact to player 1; if player 3 deviates, then player 4 will see this immediately and will be able to communicate this fact to player 0, which will transmit to player 1; if player 2 deviates, then no one will see anything, hence they will deduce the identity of the deviator in all the cases.

Using communication based on graph G_2, if either player 3 or player 4 deviates, then player 0 will see this immediately and will be able to communicate this fact to player 1 using the richness of the communication scheme (words over $\{0, 1\}$). Like before, the identity of deviator 2 will be guessed after a while.

Using communication based on graph G_3, if player 4 deviates, then player 0 will see this immediately and will be able to communicate this fact to player 1 (as before); now, no one (except players 2 and 3) will be able to learn who is deviating, if player 2 or player 3 deviates.

We can conclude that there is a Nash equilibrium with graph G_1 or G_2 whose main outcome is ρ, but not with graph G_3.

2.3 Two-player turn-based game structures

Two-player turn-based game structures are specific cases of the previous model, where at each vertex, at most one player has more than one action in her set of allowed actions. But for convenience, we will give a simplified definition, with only objects that will be useful.

A two-player turn-based game structure is a tuple $G = \langle S, S_{\texttt{Eve}}, S_{\texttt{Adam}}, s_{\mathsf{init}}, A, \mathsf{Allow}, \mathsf{Tab}\rangle$, where $S = S_{\texttt{Eve}} \sqcup S_{\texttt{Adam}}$ is a finite set of states (states in $S_{\texttt{Eve}}$ belong to player Eve whereas states in $S_{\texttt{Adam}}$ belong to player Adam), $s_{\mathsf{init}} \in S$ is the initial state, A is a finite alphabet, $\mathsf{Allow}\colon S \to 2^A \setminus \{\emptyset\}$ gives the set of available actions, and $\mathsf{Tab}\colon S \times A \to S$ is the next-state function. If $s \in S_{\texttt{Eve}}$ (resp. $S_{\texttt{Adam}}$), $\mathsf{Allow}(s)$ is the set of actions allowed to Eve (resp. Adam) in state s.

In this context, strategies will use sequences of states. That is, if a denotes Eve or Adam, an a-strategy is a partial function $\sigma_a : S^* \cdot S_a \to A$ such that for every $H \in S^* \cdot S_a$ such that $\sigma_a(H)$ is defined, $\sigma_a(H) \in \mathsf{Allow}(\mathit{last}(H))$. Note that we do not include any winning condition or payoff function in the tuple, hence the name structure.

2.4 The problems we are looking at

We are interested in the constrained existence of a Nash equilibrium. For simplicity, we define rectangular threshold constraints, but could well impose more complex constraints, like Boolean combinations of linear constraints.

▶ **Problem 1** (Constrained existence problem). *Given a concurrent game $\mathcal{G} = \langle V, v_{\mathsf{init}}, \mathcal{P}, \mathsf{Act}, \Sigma, \mathsf{Allow}, \mathsf{Tab}, (\mathsf{payoff}_a)_{a\in\mathcal{P}}\rangle$, a communication graph G for $\mathcal{P}$, a predicate P over $\mathbb{R}^{|\mathcal{P}|}$, can we decide whether there exists a Nash equilibrium $\sigma_{\mathcal{P}}$ from v_{init} such that $\mathsf{payoff}(\mathit{vertices}(\mathsf{out}(\sigma_{\mathcal{P}}, v_{\mathsf{init}}))) \in P$? If so, compute one. If the predicate P is trivial, we simply speak of the existence problem.*

The case where the communication graph has no edge was studied in depth in [4], with a generic two-player construction called the suspect construction, allowing to decide the constrained existence problem for many kinds of payoff functions. The case where the communication graph is a clique was the subject of the work [7]. The general case of a communication graph has not been investigated so far, but induces interesting developments. In the next section, we show that we can restrict the search of Nash equilibria to the search of so-called normed strategy profiles, where the communication via messages follows a very simple pattern. We also argue that deviations which do not impact the visited vertices should not be considered in the analysis. Given those reductions, we then propose the construction of a two-player game, which will track those normed profiles. This construction is inspired by the suspect-game construction of [4] and of the epistemic game of [3].

3 Reduction to profiles following a simple communication mechanism

We fix a concurrent game $\mathcal{G} = \langle V, v_{\mathsf{init}}, \mathsf{Act}, \mathscr{P}, \Sigma, \mathsf{Allow}, \mathsf{Tab}, (\mathsf{payoff}_a)_{a\in\mathscr{P}}\rangle$ and a communication graph G. We assume that $v_{\mathsf{init}} = v_0$. We will reduce the search for Nash equilibria to the search for strategy profiles with a very specific shape. In particular, we will show that the richness of the communication offered by the setting is somehow useless, and that a very simple communication pattern will be sufficient for characterizing Nash equilibria.

In the following, we write mes_ϵ for the vector assigning the empty word ϵ to every player $a \in \mathscr{P}$. Furthermore, for every $d \in \mathscr{P}$, we pick some word $\mathsf{id}_d \in \{0,1\}^+$ which are all distinct (and different from ϵ).

We first define restrictions for deviations. Let $\sigma_{\mathscr{P}}$ be a strategy profile. A player-d deviation σ'_d is said *immediately visible* whenever, writing h for the longest common prefix of $\mathsf{out}(\sigma_{\mathscr{P}}, v_0)$ and $\mathsf{out}(\sigma_{\mathscr{P}}[d/\sigma'_d], v_0)$, $\mathsf{Tab}(\mathit{last}(h), m) \neq \mathsf{Tab}(\mathit{last}(h), m')$, where $m = \sigma_{\mathscr{P}}(h)[1]$ and $m' = \big(\sigma_{\mathscr{P}}[d/\sigma'_d](h)\big)[1]$ are the next moves according to $\sigma_{\mathscr{P}}$ and $\sigma_{\mathscr{P}}[d/\sigma'_d]$. That is, at the first position where player d changes her strategy, it becomes public information that a deviation has occurred (even though some players know who deviated – all the players a with $d \rightarrowtriangle a$ – , and some other don't know). It is furthermore called *honest* whenever for every $h' \in \mathsf{out}(\sigma_{\mathscr{P}}[d/\sigma'_d], v_0)$ such that h is a (non-strict) prefix of h', $\sigma'_d(h')[2] = \mathsf{id}_d$. Somehow, player d admits she deviated, and does so immediately and forever.

The simple communication mechanism that we will design consists in reporting the deviator (role of the direct neighbours of the deviator), and propagating this information along the communication graph (for all the other players). Formally, let $\sigma_{\mathscr{P}}$ be a strategy profile, and let ρ be its main outcome. The profile $\sigma_{\mathscr{P}}$ will be said *normed* whenever the following conditions hold:

1. for every $h \in \mathsf{out}(\sigma_{\mathscr{P}}) \cup \bigcup_{d\in\mathscr{P},\ \sigma'_d} \mathsf{out}(\sigma_{\mathscr{P}}[d/\sigma'_d], v_0)$, if $\mathit{vertices}(h)$ is a prefix of $\mathit{vertices}(\rho)$, then for every $a \in \mathscr{P}$, $\sigma_a(h)[2] = \epsilon$;
2. for every $d \in \mathscr{P}$, for every d-strategy σ'_d, if $h \cdot (m, \mathsf{mes}) \cdot v \in \mathsf{out}(\sigma_{\mathscr{P}}[d/\sigma'_d], v_0)$ is the first step out of $\mathit{vertices}(\rho)$, then for every $d \rightarrowtriangle a$, $\sigma_a(h \cdot (m, \mathsf{mes}) \cdot v)[2] = \mathsf{id}_d$;
3. for every $d \in \mathscr{P}$, for every d-strategy σ'_d, if $h \cdot (m, \mathsf{mes}) \cdot v \in \mathsf{out}(\sigma_{\mathscr{P}}[d/\sigma'_d], v_0)$ has left the main outcome for more than one step, then for every $a \in \mathscr{P}$, $\sigma_a(h \cdot (m, \mathsf{mes}) \cdot v)[2] = \epsilon$ if for all $b \rightarrowtriangle a$, $\mathsf{mes}(b) = \epsilon$ and $\sigma_a(h \cdot (m, \mathsf{mes}) \cdot v)[2] = \mathsf{id}_d$ if there is $b \rightarrowtriangle a$ such that $\mathsf{mes}(b) = \mathsf{id}_d$; note that this is well defined since at most one id can be transmitted.

The first condition says that, as long as a deviation is not visible, then no message needs to be sent; the second condition says that as soon as a deviation becomes visible, then messages denouncing the deviator should be sent by "those who know", that is, the (immediate) neighbours of the deviator; the third condition says that the name (actually, the id) of the deviator should propagate according to the communication graph in an epidemic way.

Note that the profiles discussed in Section 2.2 were actually normed.

▶ **Theorem 4.** *The existence of a Nash equilibrium $\sigma_{\mathcal{P}}$ with payoff p is equivalent to the existence of a normed strategy profile $\sigma'_{\mathcal{P}}$ with payoff p, which is resistant to immediately visible and honest single-player deviations.*

The proof of this theorem, which is rather technical, can be found in [6]. We only give some intuition here. First, we explain why being resistant to immediately visible and honest deviations is enough. Notice that as long as the sequence of vertices follows the main outcome, then one can simply ignore the deviation and act only when the deviation becomes visible, in a way as if the deviator had started deviating only at this moment. This will be enough to punish the deviator. The "honest" part comes from the fact that one should simply ignore the messages sent by the deviator as it can be only in her interest to not ignore them (if it was not, then why would she send any message at all?).

Second, we show why no one should communicate as long as the sequence of vertices follows the main outcome. The reason is that if no one has deviated then any message is essentially useless, and if a deviation has happened, as explained earlier it can just be ignored as long as it has not become visible.

Finally, we demonstrate why the richness of the communication mechanism is in a way useless. Intuitively, one can understand that the only factors that should matter when playing are the sequence of the vertices that have been visited (because payoff functions only take into account the visited vertices) and the identity of the deviator. Thus the messages should only be used so that players can know of the identity of the deviator in the fastest possible way, and we show that nothing is faster than a sort of epidemic mechanism where one simply broadcasts the identity of the deviator whenever one received the information.

4 The epistemic game abstraction

We fix a concurrent game $\mathcal{G} = \langle V, v_{\mathsf{init}}, \mathsf{Act}, \mathcal{P}, \Sigma, \mathsf{Allow}, \mathsf{Tab}, (\mathsf{payoff}_a)_{a\in\mathcal{P}}\rangle$ for the rest of the section, and G be a communication graph for $\mathcal{P}$. We will implement an epistemic abstraction, which will track normed strategy profiles, and check that there is no profitable immediately visible and honest single-player deviations.

4.1 Description of the epistemic game

A *situation* is a triple (d, I, K) in $\mathcal{P} \times 2^{\mathcal{P}} \times \left(2^{\mathcal{P}}\right)^{\mathcal{P}}$, which consists of a deviator $d \in \mathcal{P}$, a list of players I having received the information that d is the deviator, and a knowledge function K that associates to every player a a list of suspects $K(a)$; in particular, it should be the case that $d \in I$ and for every $a \in I$, $K(a) = \{d\}$. We write Sit for the set of situations.

The epistemic game $\mathcal{E}^G_{\mathcal{G}}$ of $\mathcal{G}$ and G is defined as a two-player game structure $\langle S, S_{\mathtt{Eve}}, S_{\mathtt{Adam}}, s_{\mathsf{init}}, \Sigma', \mathsf{Allow}', \mathsf{Tab}'\rangle$. We describe the states and the transitions leaving those states; in particular, components Σ', Allow', Tab' of the above tuple will only be implicitely defined.

$\mathtt{Eve}$'s states $S_{\mathtt{Eve}}$ consist of elements of $V \times 2^{\mathsf{Sit}}$ such that if (v, X) is a state then for all $a \in \mathcal{P}$ the set $\{(d, I, K) \in X \mid d = a\}$ is either a singleton or empty (there is at most one situation associated with a given player a). We write $\mathsf{dev}(X)$ the set $\{d \in \mathcal{P} \mid \exists (d, I, K) \in X\}$ of agents which are a deviator in one situation of X. If $d \in \mathsf{dev}(X)$, we write (d, I^X_d, K^X_d) for the unique triple belonging to X having deviator d. Hence, $X = \{(d, I^X_d, K^X_d) \mid d \in \mathsf{dev}(X)\}$.

Intuitively, an $\texttt{Eve}$'s state (v, X) will correspond to a situation where the game has proceeded to vertex v, but, if $\mathsf{dev}(X) \neq \emptyset$, several players may have deviated. Each player $d \in \mathsf{dev}(X)$ may be responsible for the deviation; some people will have received a message denouncing d (those are in the set I_d^X), and some will deduce things from what they observe (this is given by K_d^X). Note that the (un)distinguishability relation of a player a will be deduced from X: if d deviated and $a \in I_d^X$, then a will know d deviated; if a is neither in I_d^X nor in $I_{d'}^X$, then a will not be able to know whether d or d' deviated (as we will prove later, in Lemma 5).

First let us consider the case where $X = \emptyset$, which is to be understood as the case where no deviation has arisen yet. In state $(v, \emptyset)$, $\texttt{Eve}$'s actions are moves in $\mathcal{G}$ enabled in v. When she plays move $m \in \mathsf{Act}^{\mathscr{P}}$, the game progresses to $\texttt{Adam}$'s state $((v, \emptyset), m) \in S_{\texttt{Adam}}$ where $\texttt{Adam}$'s actions are vertices $v' \in V$ such that there exists a player $d \in \mathscr{P}$ and an action $\delta \in \mathsf{Act}$ such that $\mathsf{Tab}(v, (m[d/\delta])) = v'$. When $\texttt{Adam}$ plays v', either $v' = \mathsf{Tab}(v, m)$ and the game progresses to $\texttt{Eve}$'s state $(v', \emptyset)$ or $v' \neq \mathsf{Tab}(v, m)$ and the game progresses to $\texttt{Eve}$'s state (v', X') where:

- $d \in \mathsf{dev}(X')$ if and only if there is $\delta \in \mathsf{Act}$ such that $\mathsf{Tab}(v, (m[d/\delta])) = v'$. It means that given the next state v', d is a possible deviator;
- if $d \in \mathsf{dev}(X')$, then:
  - $I_d^{X'} = \{a \in \mathscr{P} \mid d \rightarrowtriangle a\}$;
  - for every $a \in I_d^{X'}$, $K_d^{X'}(a) = \{d\}$;
  - for every $a \notin I_d^{X'}$, $K_d^{X'}(a) = \{b \in \mathscr{P} \mid \exists \beta \in \mathsf{Act} \text{ s.t. } \mathsf{Tab}(v, (m[b/\beta])) = v'\} \setminus \{b \in \mathscr{P} \mid b \rightarrowtriangle a\}$. Those are all the players that can be suspected by a, given the vertex v', and the absence of messages so far.

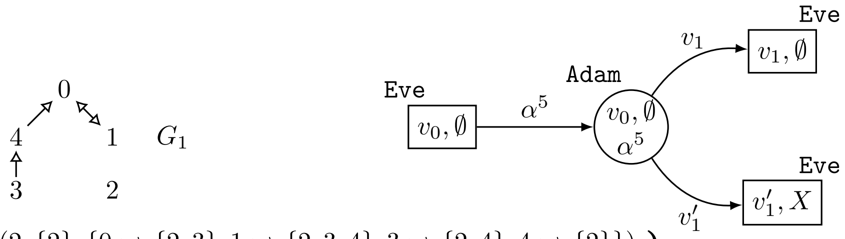
We write $X' = \mathsf{upd}((v, \emptyset), m, v')$. Note that $X' = \emptyset$ whenever (and only when) $\mathsf{Tab}(v, m) = v'$.

In a state $(v, X) \in S_{\texttt{Eve}}$ where $X \neq \emptyset$, $\texttt{Eve}$'s actions consist of functions from $\mathsf{dev}(X)$ to $\mathsf{Act}^{\mathscr{P}}$ that are compatible with players' knowledge, that is: $f : \mathsf{dev}(X) \to \mathsf{Act}^{\mathscr{P}}$ is an action enabled in (v, X) if and only if (i) for all $d \in \mathsf{dev}(X)$, for each $a \in \mathscr{P}$, $f(d)(a) \in \mathsf{Allow}(v, a)$, (ii) for all $d, d' \in \mathsf{dev}(X)$, for all $a \in \mathscr{P}$, if $a \notin I_d^X \cup I_{d'}^X$ and $K_d^X(a) = K_{d'}^X(a)$ then $f(d)(a) = f(d')(a)$;[2] that is, if a player has not received any message so far but has the same knowledge about the possible deviators in two situations, then $\texttt{Eve}$'s suggestion for that player's action must be the same in both situations. When $\texttt{Eve}$ plays action f in (v, X), the next state is $((v, X), f) \in S_{\texttt{Adam}}$, where $\texttt{Adam}$'s actions correspond to states of the game that are compatible with (v, X) and f, that is states v' such that there exists $d \in \mathsf{dev}(X)$ and $\delta \in \mathsf{Act}$ such that $\mathsf{Tab}(v, f(d)[d/\delta]) = v'$.

When $\texttt{Adam}$ chooses the action v' in $((v, X), f)$, the game progresses to $\texttt{Eve}$'s state (v', X'), where:

- $d \in \mathsf{dev}(X')$ if and only if $d \in \mathsf{dev}(X)$ and there exists $\delta \in \mathsf{Act}$ such that $\mathsf{Tab}(v, f(d)[d/\delta]) = v'$. It corresponds to a case where d was already a possible deviator and can continue deviating so that the game goes to v';
- if $d \in \mathsf{dev}(X')$, then:
  - $I_d^{X'} = I_d^X \cup \{a \in \mathscr{P} \mid \exists b \in I_d^X \text{ s.t. } b \rightarrowtriangle a\}$. New players receive a message with the deviator id;
  - for every $a \in I_d^{X'}$, $K_d^{X'}(a) = \{d\}$;

[2] Note in particular that "$K_d^X(a)$ singleton" does not imply $a \in I_d^X$, those are two distinguishable situations: the message with the identity of the deviator may not have been received in the first case, while it has been received in the second case.



$$X = \left\{ \begin{array}{l} (2, \{2\}, \{0 \mapsto \{2,3\}, 1 \mapsto \{2,3,4\}, 3 \mapsto \{2,4\}, 4 \mapsto \{2\}\}), \\ (3, \{3,4\}, \{0 \mapsto \{2,3\}, 1 \mapsto \{2,3,4\}, 2 \mapsto \{3,4\}\}), \\ (4, \{0,4\}, \{1 \mapsto \{2,3,4\}, 2 \mapsto \{3,4\}, 3 \mapsto \{2,4\}\}) \end{array} \right\}$$

Figure 2 Part of the epistemic game corresponding to the game described in Figure 1 (with graph G_1). This does not represent the whole epistemic game and a lot of actions accessible in the states we show here are not written. In situations (d, I, K) we describe K by the list of its values for players $a \notin I$, as for all a in I we have $K(a) = \{d\}$ by definition.

- for every $a \notin I_d^{X'}$, $K_d^{X'}(a) = \{b \in K_d^X(a) \mid \exists \beta \in \mathsf{Act} \text{ s.t. } \mathsf{Tab}(v, f(b)[b/\beta]) = v'\} \setminus \{c \in \mathcal{P} \mid \mathsf{dist}_G(c, a) \leq \max\{\mathsf{dist}_G(c, c') \mid c' \in I_c^X\} + 1\}$. Those are the players that could have deviated but for which player a would not have received the signal yet.

We write $X' = \mathsf{upd}((v, X), f, v')$. Note that $X' \neq \emptyset$ and that $\mathsf{dev}(X') \subseteq \mathsf{dev}(X)$.

We let $R = (v_0, X_0) \cdot ((v_0, X_0), f_0) \cdot (v_1, X_1) \ldots$ be an infinite play from $(v_0, X_0) = (v_{\mathsf{init}}, \emptyset)$. We write *visited*$(R)$ for $v_0 v_1 \cdots \in V^\omega$ the sequence of vertices visited along R. We also define $\mathsf{dev}(R) = \emptyset$ if $X_r = \emptyset$ for every r, and $\mathsf{dev}(R) = \lim_{r \to +\infty} \mathsf{dev}(X_r)$ otherwise. This is the set of possible deviators along R.

4.1.1 Winning condition of Eve

A zero-sum game will be played on the game structure $\mathcal{E}_\mathcal{G}^G$, and the winning condition of Eve will be given on the branching structure of the set of outcomes of a strategy for Eve, and not individually on each outcome, as standardly in two-player zero-sum games. We write $s_{\mathsf{init}} = (v_{\mathsf{init}}, \emptyset)$ for the initial state. Let $p = (p_a)_{a \in \mathcal{P}} \in \mathbb{R}^\mathcal{P}$, and ζ be a strategy for Eve in $\mathcal{E}_\mathcal{G}^G$; it is said winning for p from s_{init} whenever $\mathsf{payoff}(\mathit{visited}(R)) = p$, where R is the unique outcome of ζ from s_{init} where Adam complies to Eve's suggestions, and for every other outcome R' of ζ, for every $d \in \mathsf{dev}(R')$, $\mathsf{payoff}_d(\mathit{visited}(R')) \leq p_d$.

4.2 An example

In Figure 2 we present a part of the epistemic game corresponding to the game we described in Figure 1 with graph G_1. In state $(v_0, \emptyset)$, Eve can play the action profile α^5 and make the game go to $((v_0, \emptyset), \alpha^5)$ where Adam can either play $v_1 = \mathsf{Tab}(v_0, \alpha^5)$ (we say that Adam complies with Eve) or choose a different state accessible from v_0 and an action profile that consists in a single-player deviation from α, for instance $v_1' = \mathsf{Tab}(v_0, \alpha^2\beta\alpha^2) = \mathsf{Tab}(v_0, \alpha^3\beta\alpha) = \mathsf{Tab}(v_0, \alpha^4\beta)$. If Adam chooses v_1', then three players are possible deviators: 2, 3 and 4. We write X for the corresponding set of situations, and we already know that $\mathsf{dev}(X) = \{2, 3, 4\}$.

- If player 2 is the deviator, then no one (except himself) directly receives this information. Player 0 knows that player 4 did not deviate (since $4 \to 0$ in G_1), hence $K_2^X(0) = \{2, 3\}$; Player 1 has no information hence $K_2^X(1) = \{2, 3, 4\}$; Player 3 knows that he is not the deviator but cannot know more, hence $K_2^X(3) = \{2, 4\}$; Finally, player 4 can deduce many things: he knows he is not the deviator, and he saw that player 3 is not the deviator (since $3 \to 4$ in G_1), hence $K_2^X(4) = \{2\}$.

- If player 3 is the deviator, then both players 3 and 4 get the information, hence $I_3^X = \{3,4\}$. Other players can guess some things, for instance player 0 sees that player 4 cannot be the deviator, this is why $K_3^X(0) = \{2,3\}$. Etc.
- The reasoning for player 4 is similar.

In the situation we have just described, when the game will proceed to v_1', then either player 0 knows that player 4 has deviated, or he knows that player 4 didn't deviate but he suspects both 2 and 3. On the other hand, player 4 will precisely know who deviated. And player 3 knows whether he deviated or not, but if he didn't, then he cannot know whether it was player 2 or player 4 who deviated. This knowledge is stored in situation X we have described, and which is fully given in Figure 2.

Let us now illustrate how actions of Eve are defined in states with a non-empty set of situations. Assume we are in Eve's state (v_0, X), with X as previously defined. From that state, an action for Eve is a mapping $f : \{2,3,4\} \to \mathsf{Act}^{\mathcal{P}}$ such that:

$$f(2)(0) = f(3)(0) \quad f(2)(1) = f(3)(1) = f(4)(1) \quad f(3)(2) = f(4)(2) \quad f(2)(3) = f(4)(3)$$

The intuition behind these constraints is the following: Player 0 knows whether Player 4 deviated or not, but in the case she did not cannot know whether Player 2 or Player 3 deviated; Player 1 does not know who deviated, hence should play the same action in the three cases (that she cannot distinguish); Player 2 does only know whether she deviated hence in the case she did not cannot know whether Player 3 or Player 4 deviated; the case for Player 3 is similar; finally Player 4 knows for sure who deviated: she saw if Player 3 deviated and knows whether she herself deviated, thus can distinguish between the three cases.

4.3 Correctness of the epistemic game construction

When constructing the epistemic game, we mentioned that Eve's states will allow to properly define the undistinguishability relation for all the players. Towards that goal, we show by an immediate induction the following result:

▶ **Lemma 5.** *If (v, X) is an Eve's state reachable from some $(v_0, \emptyset)$ in $\mathcal{E}_{\mathcal{G}}^{G}$, then for all $d \in \mathsf{dev}(X)$:*

- *for all $a \in I_d^X$, $K_d^X(a) = \{d\}$;*
- *for all $a \notin I_d^X$, $K_d^X(a) = \mathsf{dev}(X) \setminus \{d' \in \mathsf{dev}(X) \mid a \in I_{d'}^X\}$.*

In particular, for all $d, d' \in \mathsf{dev}(X)$, for all $a \notin I_d^X \cup I_{d'}^X$, $K_d^X(a) = K_{d'}^X(a)$.

So, either a player a will have received from a neighbour the identity of the deviator, or she will not have received any deviator identity yet, and she will have a set of suspected deviators that she will not be able to distinguish.

This allows to deduce the following correspondence between $\mathcal{G}$ and $\mathcal{E}_{\mathcal{G}}^{G}$:

▶ **Proposition 6.** *There is a winning strategy for Eve in $\mathcal{E}_{\mathcal{G}}^{G}$ for payoff p if and only if there is a normed strategy profile in $\mathcal{G}$, whose main outcome has payoff p and which is resistant to single-player immediately visible and honest deviations.*

The proof of correctness of the epistemic game then goes through the following steps, which are detailed in [6]. First, given an Eve's strategy ζ, we build a function E_ζ associating with a-histories (for every $a \in \mathcal{P}$) in the original game Eve's histories in the epistemic game such that Eve plays according to ζ along E_ζ.

Then we use this function to create a strategy profile $\Omega(\zeta)$ for the original game where the action prescribed by this profile to player a after history h corresponds in some sense to

$\zeta(E_\zeta(h))(d)(a)$, where d is a suspected deviator according to player a. This works because, thanks to Lemma 5, we know that either player a knows who the deviator is, or player a has a subset of suspect deviators and Eve's suggestion for a (by construction of $\mathcal{E}^G_{\mathcal{G}}$) is the same for all those possible deviators.

Finally we prove that if ζ is a winning strategy for Eve then $\Omega(\zeta)$ is both normed and resistant to single-player immediately visible and honest deviations in $\mathcal{G}$.

To prove the converse proposition we build a function Λ associating with Eve's histories in the epistemic game families of single-player histories in the original game. We then use this correspondence to build a function Υ associating with normed strategy profiles Eve's strategies in a natural way.

Finally we prove that if σ is normed and resistant to single-player immediately visible honest deviations, then $\Upsilon(\sigma)$ is a winning strategy for Eve.

Gathering results of Theorem 4 and of this proposition, we get the following theorem:

▶ **Theorem 7.** *There is a Nash equilibrium with payoff p in $\mathcal{G}$ if and only if there is a winning strategy for* Eve *in $\mathcal{E}^G_{\mathcal{G}}$ for payoff p.*

▶ Remark 8. Note that all the results are constructive, hence if one can synthesize a winning strategy for Eve in $\mathcal{E}^G_{\mathcal{G}}$, then one can synthesize a correponding Nash equilibrium in $\mathcal{G}$.

5 Complexity analysis

We borrow all notations of previous sections. A rough analysis of the size of the epistemic game $\mathcal{E}^G_{\mathcal{G}}$ gives an exponential bound. We will give a more precise bound, pinpointing the part with an exponential blowup. We write $\mathsf{diam}(G)$ for the diameter of G, that is $\mathsf{diam}(G) = \max\{\mathsf{dist}_G(a,b) \mid \mathsf{dist}_G(a,b) < +\infty\}$.

▶ **Lemma 9.** *Assuming that* Tab *is given explicitely in $\mathcal{G}$, the number of states in the reachable part of $\mathcal{E}^G_{\mathcal{G}}$ from $s_{\mathsf{init}} = (v_{\mathsf{init}}, \emptyset)$ is bounded by*

$$|S_{\mathtt{Eve}}| \leq |V| + |V| \cdot |\mathsf{Tab}|^2 \cdot (\mathit{diam}(G) + 2) \qquad \textit{and} \qquad |S_{\mathtt{Adam}}| \leq |S_{\mathtt{Eve}}| \cdot |\mathsf{Act}|^{|\mathcal{P}|^2}$$

The number of edges is bounded by $|S_{\mathtt{Adam}}| + |S_{\mathtt{Adam}}| \cdot |S_{\mathtt{Eve}}|$.

If $|\mathcal{P}|$ is assumed to be a constant of the problem, then the size of $\mathcal{E}^G_{\mathcal{G}}$ is polynomial in the size of $\mathcal{G}$.

We will not detail algorithmics issues, but the winning condition of Eve in $\mathcal{E}^G_{\mathcal{G}}$ is very similar to the winning condition of Eve in the suspect-game construction of [4] (for Boolean or ordered objectives), or in the deviator-game construction of [7] (for mean-payoff), or in a closer context to the epistemic-game construction of [3]. Hence, when the size of the epistemic game is polynomial, rather efficient algorithms can be designed to compute Nash equilibria. For instance, in a setting where the size of $\mathcal{E}^G_{\mathcal{G}}$ is polynomial, using a bottom-up labelling algorithm similar to that of [2, Sect. 4.3], one obtains a polynomial space algorithm for deciding the (constrained) existence of a Nash equilibrium when payoffs are Boolean payoffs corresponding to parity conditions.

6 Conclusion

In this paper, we have studied multiplayer infinite-duration games over graphs, and focused on games where players can communicate with neighbours, given by a directed graph. We have shown that a very simple communication mechanism was sufficient to describe Nash equilibria.

This mechanism is sort of epidemic, in that if a player deviates, then his neighbours will see it and transmit the information to their own neighbours; the information then propagates along the communication graph. This framework encompasses two standard existing frameworks, one where the actions are invisible (represented with a graph with no edges), and one where all actions are visible (represented by the complete graph). We know from previous works that in both frameworks, one can compute Nash equilibria for many kinds of payoff functions. In this paper, we also show that we can compute Nash equilibria in this generalized framework, by providing a reduction to a two-player game, the so-called epistemic game construction. Winning condition in this two-player game is very similar to winning conditions encountered in the past, yielding algorithmic solution to the computation of Nash equilibria. We have also analyzed the size of the abstraction, which is polynomial when the number of players is considered as a constant of the problem.

The current framework assumes messages can be appended to actions by players, allowing a rich communication between players. The original framework of [14] did not allow additional messages, but did encode identities of deviators by sequences of actions. This was possible in [14] since games were repeated matrix games, but it is harder to see how we could extend this approach and how we could encode identities of players with actions, taking into account the graph structure. For instance, due to the graph, having too long identifiers might be prohibitive to transmit in a short delay the identity of the deviator. Nevertheless, that could be interesting to see if something can be done in this framework.

References

1 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002. doi:10.1145/585265.585270.

2 Patricia Bouyer. Games on graphs with a public signal monitoring. Research report, arXiv, 2017. arXiv:1710.07163.

3 Patricia Bouyer. Games on graphs with a public signal monitoring. In *Proc. 21st International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'18)*, volume 10803 of *Lecture Notes in Computer Science*, pages 530–547. Springer, 2018.

4 Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Pure Nash Equilibria in Concurrent Games. *Logical Methods in Computer Science*, 11(2:9), 2015.

5 Patricia Bouyer, Nicolas Markey, and Daniel Stan. Mixed Nash Equilibria in Concurrent Games. In *Proc. 33rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'14)*, volume 29 of *LIPIcs*, pages 351–363. Leibniz-Zentrum für Informatik, 2014.

6 Patricia Bouyer and Nathan Thomasset. Nash equilibria in games over graphs equipped with a communication mechanism. Research report, arXiv, 2019. arXiv:1906.07753.

7 Romain Brenguier. Robust Equilibria in Mean-Payoff Games. In *Proc. 19th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'16)*, volume 9634 of *Lecture Notes in Computer Science*, pages 217–233. Springer, 2016.

8 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdziński. Games with Secure Equilibria. *Theoretical Computer Science*, 365(1-2):67–82, 2006.

9 Krishnendu Chatterjee, Rupak Majumdar, and Marcin Jurdziński. On Nash Equilibria in Stochastic Games. In *Proc. 18th International Workshop on Computer Science Logic (CSL'04)*, volume 3210 of *Lecture Notes in Computer Science*, pages 26–40. Springer, 2004.

10 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The Complexity of Rational Synthesis. In *Proc. 43rd International Colloquium on Automata, Languages and Programming (ICALP'16)*, volume 55 of *LIPIcs*, pages 121:1–121:15. Leibniz-Zentrum für Informatik, 2016.

11 Rodica Condurache, Youssouf Oualhadj, and Nicolas Troquard. The Complexity of Rational Synthesis for Concurrent Games. In *Proc. 29th International Conference on Concurrency Theory (CONCUR'18)*, volume 118 of *LIPIcs*, pages 38:1–38:15. Leibniz-Zentrum für Informatik, 2018.

12 Thomas A. Henzinger. Games in system design and verification. In *Proc. 10th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'05)*, pages 1–4, 2005.

13 John F. Nash. Equilibrium Points in n-Person Games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49, 1950.

14 Jérôme Renault and Tristant Tomala. Repeated Proximity Games. *International Journal of Game Theory*, 27(4):539–559, 1998.

15 Wolfgang Thomas. Infinite Games and Verification. In *Proc. 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 58–64. Springer, 2002. Invited Tutorial.

16 Michael Ummels. Rational Behaviour and Strategy Construction in Infinite Multiplayer Games. In *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 212–223. Springer, 2006.

17 Michael Ummels. The Complexity of Nash Equilibria in Infinite Multiplayer Games. In *Proc. 11th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'08)*, volume 4962 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2008.

18 Michael Ummels and Dominik Wojtczak. The Complexity of Nash Equilibria in Limit-Average Games. In *Proc. 22nd International Conference on Concurrency Theory (CONCUR'11)*, volume 6901 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2011.

19 Michael Ummels and Dominik Wojtczak. The Complexity of Nash Equilibria in Stochastic Multiplayer Games. *Logical Methods in Computer Science*, 7(3), 2011.

Parity Games: Zielonka's Algorithm in Quasi-Polynomial Time

Paweł Parys
Institute of Informatics, University of Warsaw, Poland
parys@mimuw.edu.pl

Abstract

Calude, Jain, Khoussainov, Li, and Stephan (2017) proposed a quasi-polynomial-time algorithm solving parity games. After this breakthrough result, a few other quasi-polynomial-time algorithms were introduced; none of them is easy to understand. Moreover, it turns out that in practice they operate very slowly. On the other side there is Zielonka's recursive algorithm, which is very simple, exponential in the worst case, and the fastest in practice. We combine these two approaches: we propose a small modification of Zielonka's algorithm, which ensures that the running time is at most quasi-polynomial. In effect, we obtain a simple algorithm that solves parity games in quasi-polynomial time. We also hope that our algorithm, after further optimizations, can lead to an algorithm that shares the good performance of Zielonka's algorithm on typical inputs, while reducing the worst-case complexity on difficult inputs.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Parity games, Zielonka's algorithm, quasi-polynomial time

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.10

Funding Work supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

Acknowledgements The author would like to thank Wojciech Czerwiński, Laure Daviaud, Marcin Jurdziński, Eryk Kopczyński, Ranko Lazić, Karoliina Lehtinen, and Igor Walukiewicz for all the discussions that preceded writing of this paper, and the anonymous reviewers of all versions of this paper for their valuable comments.

1 Introduction

The fundamental role of parity games in automata theory and logic, and their applications to verification and synthesis is doubtless, hence it is pointless to elaborate on their importance. Let us only mention that the algorithmic problem of finding the winner in parity games is polynomial-time equivalent to the emptiness problem for nondeterministic automata on infinite trees with parity acceptance conditions, and to the model-checking problem for modal μ-calculus [10]. It also lies at the heart of algorithmic solutions to Church's synthesis problem [29]. The impact of parity games reaches relatively far areas of computer science, like Markov decision processes [11] and linear programming [15].

It is a long-standing open question whether parity games can be solved in polynomial time. Several results show that they belong to some classes "slightly above" polynomial time. Namely, deciding the winner of parity games was shown to be in $\mathsf{NP} \cap \mathsf{coNP}$ [10], and in $\mathsf{UP} \cap \mathsf{coUP}$ [18], while computing winning strategies is in PLS, PPAD, and even in their subclass CLS [9]. The same holds for other kinds of games: mean-payoff games [36], discounted games, and simple stochastic games [7]; parity games, however, are the easiest among them, in the sense that there are polynomial-time reductions from parity games to the other kinds of games [18, 36], but no reductions in the opposite direction are known.

Describing the algorithmic side of solving parity games, one has to start with Zielonka's algorithm [35], being an adaptation of an approach proposed by McNaughton to solve Muller

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 10; pp. 10:1–10:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

games [28]. This algorithm consists of a single recursive procedure, being simple and very natural; one may say that it computes who wins the game "directly from the definition". Its running time is exponential in the worst case [14, 1, 16], but on many typical inputs it works much faster. For over two decades researchers were trying to cutback the complexity of solving parity games, which resulted in a series of algorithms, all of which were either exponential [5, 31, 19, 34, 30, 2], or mildly subexponential [3, 21]. The next era came unexpectedly in 2017 with a breakthrough result of Calude, Jain, Khoussainov, Li, and Stephan [6] (see also [17, 23]), who designed an algorithm working in quasi-polynomial time. This invoked a series of quasi-polynomial-time algorithms, which appeared soon after [20, 13, 24]. These algorithms are quite involved (at least compared to the simple recursive algorithm of Zielonka), and it is not so trivial to understand them.

The four quasi-polynomial-time algorithms [6, 20, 13, 24], at first glance being quite different, actually proceed along a similar line (as observed by Bojańczyk and Czerwiński [4] and Czerwiński et al. [8]). Namely, out of all the four algorithms one can extract a construction of a safety automaton (nondeterministic in the case of Lehtinen [24], and deterministic in the other algorithms), which accepts all words encoding plays that are decisively won by one of the players (more precisely: plays consistent with some positional winning strategy), and rejects all words encoding plays in which the player loses (for plays that are won by the player, but not decisively, the automaton can behave arbitrarily). This automaton does not depend at all on the game graph; it depends only on its size. Having an automaton with the above properties, it is not difficult to convert the original parity game into an equivalent safety game (by taking a "product" of the parity game and the automaton), which can be solved easily – and all the four algorithms actually proceed this way, even if it is not stated explicitly that such an automaton is constructed. As shown in Czerwiński et al. [8], all automata having the aforementioned properties have to look very similar: their states have to be leaves of some so-called universal tree; particular papers propose different constructions of these trees, and of the resulting automata (of quasi-polynomial size). Moreover, Czerwiński et al. [8] show a quasi-polynomial lower bound for the size of such an automaton.

In this paper we propose a novel quasi-polynomial-time algorithm solving parity games. It is obtained by applying a small modification to Zielonka's recursive algorithm; this modification guarantees that the worst-case running time of this algorithm, being originally exponential, becomes quasi-polynomial. The simplicity of Zielonka's algorithm remains in place; we avoid complicated considerations accompanying all the previous quasi-polynomial-time algorithms. Another point is that our algorithm exploits the structure of parity games in a rather different way from the four previous quasi-polynomial-time algorithms. Indeed, the other algorithms construct automata that are completely independent from a particular game graph given on input – they work in exactly the same way for every game graph of a considered size. The behaviour of our algorithm, in contrast, is highly driven by an analysis of the game graph given on input. In particular, although our algorithm is not faster than quasi-polynomial, it does not fit to the "separator approach" in which a quasi-polynomial lower bound of Czerwiński et al. [8] exists.

The running time of our algorithm is quasi-polynomial, and the space complexity is quadratic (more precisely, $O(n \cdot h)$, where n is the number of nodes in the game graph, and h is the maximal priority appearing there).

We remark that Lehtinen, Schewe, and Wojtczak in their recent follow up paper [25] suggested a variation of our algorithm that improves the complexity to meet the state-of-the-art complexity of broadly $2^{O((\log n)(\log h))}$, while providing polynomial bounds when the number of priorities is logarithmic.

Let us also mention the practical side of the world. It turns out that parity games are one of the areas where theory does not need to meet practice: the quasi-polynomial-time algorithms, although fastest in theory, are actually the slowest. The most exhaustive comparison of existing algorithms was performed by Tom van Dijk [32]. In his Oink tool he has implemented several algorithms, with different optimizations. Then, he has evaluated them on a benchmark of Keiren [22], containing multiple parity games obtained from model checking and equivalence checking tasks, as well as on different classes of random games. It turns out that the classic recursive algorithm of Zielonka [35] performs the best, *ex aequo* with the recent priority promotion algorithm [2]. After that, we have the strategy improvement algorithm [34, 12], being a few times slower. Far later, we have the small progress measure algorithm [19]. At the very end, with a lot of timeouts, we have the quasi-polynomial-time algorithm of Fearnley, Jain, Schewe, Stephan, and Wojtczak [13]. The other quasi-polynomial-time algorithms were not implemented due to excessive memory usage.

While developing the current algorithm, we hoped that it will share the good performance with Zielonka's algorithm, on which it is based. Unfortunately, preliminary experiments have shown that this is not necessarily the case. It turns out that

- on random games our algorithm performs similarly to the slowest algorithms implemented in Oink;
- on crafted game families that are difficult for Zielonka's algorithm, our algorithm is indeed faster from it, but not dramatically faster;
- the only think that is optimistic is that on games with a very low number of priorities our algorithm performs similarly to the fastest algorithms.

Because the empirical results of a direct implementation of the algorithm are completely unsatisfactory, we do not include a full description of our experiments. Instead, we leave an efficient implementation for a future work. Beside of the discouraging outcomes, we believe that our idea, via further optimizations, can lead to an algorithm that is both fast in practice and has a good worst-case complexity (see the concluding section for more comments).

2 Preliminaries

A parity game is played on a *game graph* between two players, called Even or Odd (shortened sometimes to E and O). A game graph consists of

- a directed graph G, where we require that every node has at least one successor, and where there are no self-loops (i.e., edges from a node to itself);
- a labeling of every node v of G by a positive natural number $\pi(v)$, called its *priority*;
- a partition of nodes of G between nodes owned by Even and nodes owned by Odd.

An infinite path in G is called a *play*, while a finite path in G is called a *partial play*. The game starts in a designated starting node. Then, the player to which the current node belongs, selects a successor of this node, and the game continues there. In effect, after a finite time a partial play is obtained, and at the end, after infinite time, this results in a play. We say that a play $v_1, v_2, \dots$ is winning for Even if $\limsup_{i\to\infty} \pi(v_i)$ is even (i.e., if the maximal priority seen infinitely often is even). Conversely, the play is winning for Odd if $\limsup_{i\to\infty} \pi(v_i)$ is odd.

A *strategy* of player $P \in \{\mathrm{Even}, \mathrm{Odd}\}$ is a function that maps every partial play that ends in a node of P to some its successor. Such a function says how P will play in every situation of the game (depending on the history of that game). When a (partial) play π follows a strategy σ in every step in which player P is deciding, we say that π *agrees* with σ.

A strategy σ is *winning* for P from a node v if every play that starts in v and agrees with σ is winning for P. While saying "player P wins from a node v" we usually mean that P has a winning strategy from v. Let $Win_P(G)$ be the set of nodes of G from which P wins; it is called the *winning region* of P. By Martin's theorem [27] we know that parity games are determined: in every game graph G, and for every node v of G either Even wins from v, or Odd wins from v. In effect, $Win_E(G)$ and $Win_O(G)$ form a partition of the node set of G.

During the analysis, we also consider games with other winning conditions. A *winning condition* is a set of plays. The winning conditions of Even and Odd considered in parity games are denoted LimsupEven and LimsupOdd, respectively. Beside of that, for every set S of nodes, let Safety(S) be the set of plays that use only nodes from S.

A *dominion* for Even is a set S of nodes such that from every $v \in S$ Even wins the game with the condition LimsupEven $\cap$ Safety(S); in other words, from every node of S he can win the parity game without leaving S. Likewise, a dominion for Odd is a set S of nodes such that from every $v \in S$ Odd wins the game with the condition LimsupOdd $\cap$ Safety(S). Notice that the whole $Win_P(G)$ is a dominion for P (where $P \in \{\text{Even}, \text{Odd}\}$). Indeed, if Even is going to win from some $v \in Win_E(G)$, the play cannot leave $Win_E(G)$ and enter a node $v' \in Win_O(G)$, as then Odd could use his winning strategy from v' and win the whole game; here we use the fact that all suffixes of a play in LimsupEven are also in LimsupEven. For $P =$ Odd the situation is symmetric.

3 Standard Zielonka's Algorithm

Before presenting our algorithm, we recall the standard Zielonka's algorithm, as a reference.

For a set of nodes N in a game graph G, and for a player $P \in \{\text{Even}, \text{Odd}\}$, we define the *attractor* of N, denoted $\textsc{Atr}_P(G, N)$, to be the set of nodes of G from which P can force to reach a node from N. In other words, $\textsc{Atr}_P(G, N)$ is the smallest set such that

- $N \subseteq \textsc{Atr}_P(G, N)$,
- if v is a node of P and some its successor is in $\textsc{Atr}_P(G, N)$, then $v \in \textsc{Atr}_P(G, N)$, and
- if v is a node of the opponent of P and all its successors are in $\textsc{Atr}_P(G, N)$, then $v \in \textsc{Atr}_P(G, N)$.

Clearly $\textsc{Atr}_P(G, N)$ can be computed in time proportional to the size of G.

Algorithm 1 Standard Zielonka's Algorithm.

```
procedure Solve_E(G, h)                          ▷ $h$ is an even upper bound for priorities in $G$
begin
    do begin
        $N_h = \{v \in \mathsf{nodes}(G) \mid \pi(v) = h\}$;      ▷ nodes with the highest priority
        $H = G \setminus \textsc{Atr}_E(G, N_h)$;               ▷ new game: reaching priority $h \to$ win
        $W_O = \textsc{Solve}_O(H, h-1)$;                  ▷ in $W_O$ we lose before reaching priority $h$
        $G = G \setminus \textsc{Atr}_O(G, W_O)$;               ▷ possibly $N_h \cap \textsc{Atr}_O(G, W_O) \neq \emptyset$
    end while $W_O \neq \emptyset$;
    return $\mathsf{nodes}(G)$;
end
```

Algorithm 1 is the standard Zielonka's algorithm. The procedure $\textsc{Solve}_E(G, h)$ returns $Win_E(G)$, the winning region of Even, if h is an even number that is greater or equal than all priorities appearing in G. A procedure $\textsc{Solve}_O(G, h)$ is also needed; it is identical to $\textsc{Solve}_E(G, h)$ except that the roles of E and O are swapped; it returns $Win_O(G)$, the

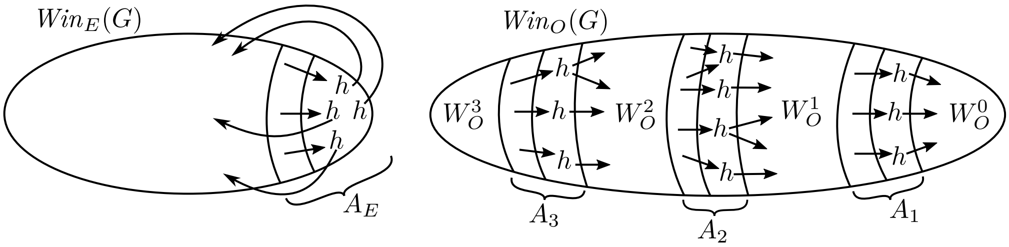


Figure 1 The structure of winning regions in a parity game.

winning region of Odd. While writing $G \setminus S$, we mean the game obtained by removing from G all nodes in S, and all edges leading to nodes in S or starting from nodes in S. We use this construct only when S is an attractor; in such a case, if all successors of a node v are removed, then v is also removed (i.e., if all successors of v belong to an attractor, then v belongs to the attractor as well). In effect $G \setminus S$ is a valid game graph (every its node has at least one successor).

We remark that the algorithm is presented in a slightly different way than usually. Namely, we use here a loop, while the usual presentation does not use a loop but rather calls recursively $\textsc{Solve}_E(G \setminus \textsc{Atr}_O(G, W_O), h)$ at the end of the procedure. This is only a superficial difference in the presentation, but is useful while modifying the algorithm in the next section.

The algorithm can be understood while looking at Figure 1. Let h be the highest priority used in G; assume that it is even. The game graph G can be divided into two parts: $Win_E(G)$ and $Win_O(G)$. In $Win_E(G)$ we can distinguish the attractor of nodes with priority h (denoted A_E). Odd either loses inside $Win_E(G) \setminus A_E$, or enters A_E, which causes that a node with priority h is seen, and then the game continues in some node of $Win_E(G)$. The winning region of Odd, $Win_O(G)$, can be divided into multiple parts. We have a part W_O^0, where Odd can win without seeing a node of priority h. Then, we have nodes of priority h from which Even is forced to enter W_O^0, and their attractor, denoted A_1. Then, we have a part W_O^1, where Odd can ensure that the play is either winning for him inside W_O^1 or enters A_1; in other words, from nodes of W_O^1 Odd can win while seeing h at most once. We also have parts W_O^i for larger i, and corresponding attractors A_i.

While running the algorithm, this partition of G is not known, and has to be discovered. To this end, the algorithm assumes first (in the game H) that all nodes of priority h are winning for Even. The first call to $\textsc{Solve}_O(H, h-1)$ returns the set W_O^0 of nodes where Odd wins without seeing a node of priority h. We then remove them from the game, together with the attractor A_1. In the next step, $\textsc{Solve}_O(H, h-1)$ returns the set W_O^1, and so on. At the end the whole $Win_O(G)$ becomes removed, and the procedure returns $Win_E(G)$.

4 Quasi-Polynomial-Time Algorithm

We now present a modification to Algorithm 1 that results in obtaining quasi-polynomial running time, in the worst case.

The modification can be understood while looking again at Figure 1. The key observation is that, while $Win_O(G)$ is of size at most n (where n is the number of nodes in G), then most of its parts W_O^i are smaller. Namely, most of them have to be of size at most $\frac{n}{2}$, and only one of them can be larger than $\frac{n}{2}$. We use this observation, and while looking for W_O^i, we search for a winning region (for a dominion) of size at most $\frac{n}{2}$. Usually this is enough;

only once it is not enough: one W_O^i can be larger than $\frac{n}{2}$ and it will not be found if we only look for a set of size at most $\frac{n}{2}$. But when the algorithm finds no set of size at most $\frac{n}{2}$, we can once search for W_O^i of an arbitrary size. After that, we know that all following sets W_O^i are again of size at most $\frac{n}{2}$. While going recursively, we notice that every W_O^i can be further subdivided in a similar way, while splitting on the priority $h-2$. If $|W_O^i| \leq \frac{n}{2}$, we again have the property that most of the parts of W_O^i are of size at most $\frac{n}{4}$, and only one of them can be larger than $\frac{n}{4}$.

To exploit this observation, in the recursive calls we pass two precision parameters, p_E and p_O (one for every of the players), saying that we search for winning sets of size at most p_E for Even, and at most p_O for Odd. The modified procedure is presented as Algorithm 2. Again, one also needs a procedure $\textsc{Solve}_O$, which is obtained from $\textsc{Solve}_E$ by literally changing every E to O and *vice versa*.

Algorithm 2 Quasi-Polynomial-Time Algorithm.

```
 1: procedure $\textsc{Solve}_E(G, h, p_E, p_O)$          ▷ $p_E, p_O$ are new "precision" parameters
 2: begin
 3:     if $G = \emptyset \lor p_E \leq 1$ then
 4:         return $\emptyset$;                        ▷ we assume that there are no self-loops in $G$
 5:     do begin
 6:         $N_h = \{v \in \mathsf{nodes}(G) \mid \pi(v) = h\}$;
 7:         $H = G \setminus \textsc{Atr}_E(G, N_h)$;
 8:         $W_O = \textsc{Solve}_O(H, h-1, \lfloor p_O/2 \rfloor, p_E)$;          ▷ precision decreased
 9:         $G = G \setminus \textsc{Atr}_O(G, W_O)$;
10:     end while $W_O \neq \emptyset$;
11:     $N_h = \{v \in \mathsf{nodes}(G) \mid \pi(v) = h\}$;
12:     $H = G \setminus \textsc{Atr}_E(G, N_h)$;
13:     $W_O = \textsc{Solve}_O(H, h-1, p_O, p_E)$;          ▷ we try once with the full precision
14:     $G = G \setminus \textsc{Atr}_O(G, W_O)$;
15:     while $W_O \neq \emptyset$ do begin
16:         $N_h = \{v \in \mathsf{nodes}(G) \mid \pi(v) = h\}$;
17:         $H = G \setminus \textsc{Atr}_E(G, N_h)$;
18:         $W_O = \textsc{Solve}_O(H, h-1, \lfloor p_O/2 \rfloor, p_E)$;          ▷ again, precision decreased
19:         $G = G \setminus \textsc{Atr}_O(G, W_O)$;
20:     end;
21:     return $\mathsf{nodes}(G)$;
22: end
```

We start the algorithm with $p_E = p_O = n$, where n is the number of nodes in G. In the procedure we have now, in a sense, three copies of the previous procedure, corresponding to three stages. In the first stage, in lines 5-10, we look for sets W_O^i of size at most $\lfloor \frac{p_O}{2} \rfloor$. If the returned set is empty, this may mean that the next W_O^i either is empty, or is of size greater than $\lfloor \frac{p_O}{2} \rfloor$. Then, in lines 11-14, we once search for a set W_O^i of size at most p_O (knowing that if it is nonempty, then its size is greater than $\lfloor \frac{p_O}{2} \rfloor$). Finally, in the loop in lines 15-20, we again look for sets W_O^i of size at most $\lfloor \frac{p_O}{2} \rfloor$ (because we have already found a set of size greater than $\lfloor \frac{p_O}{2} \rfloor$, all the remaining sets have size at most $\lfloor \frac{p_O}{2} \rfloor$).

5 Complexity Analysis

Let us analyze the complexity of our algorithm.

First, we observe that the space complexity is $O(n \cdot h)$, where n is the number of nodes, and h is the maximal priority. Indeed, the depth of the recursion is at most h, and on every step we only need to remember some sets of nodes.

We now come to the running time. As it is anyway worse than the running time of the other quasi-polynomial-time algorithms, we do not aim in proving a very tight upper bound; we only prove that the running time is quasi-polynomial.

Let $R(h,l)$ be the number of (nontrivial) executions of the $\textsc{Solve}_E$ and $\textsc{Solve}_O$ procedures performed during one call to $\textsc{Solve}_E(G,h,p_E,p_O)$ with $\lfloor \log p_E \rfloor + \lfloor \log p_O \rfloor = l$, and with G having at most n nodes (where n is fixed). We only count here nontrivial executions, that is, such that do not leave the procedure in line 4. Clearly $R(0,l) = R(h,0) = 0$. For $h,l \geq 1$ it holds that

$$R(h,l) \leq 1 + n \cdot R(h-1,l-1) + R(h-1,l)\,. \tag{1}$$

Indeed, in $\textsc{Solve}_E$ after every call to $\textsc{Solve}_O$ we remove at least one node from G, with the exception of two such calls: the last call in line 8, and the last call ever. In effect, in lines 8 and 18 we have at most n calls to $\textsc{Solve}_O$ with decreased precision (plus, potentially, the $(n+1)$-th call with empty G, which is not included in $R(h,l)$), and in line 13 we have one call to $\textsc{Solve}_O$ with full precision. Notice that $\lfloor \log p_O \rfloor$ (hence also l) decreases by 1 in the decreased-precision call.

Using Inequality (1) we now prove by induction that $R(h,l) \leq n^l \cdot \binom{h+l}{l} - 1$. For $h = 0$ and for $l = 0$ the inequality holds. For $h,l \geq 1$ we have that

$$\begin{aligned} R(h,l) &\leq 1 + n \cdot R(h-1,l-1) + R(h-1,l) \\ &\leq 1 + n \cdot \left(n^{l-1} \cdot \binom{h-1+l-1}{l-1} - 1 \right) + n^l \cdot \binom{h-1+l}{l} - 1 \\ &\leq n^l \cdot \left(\binom{h-1+l}{l-1} + \binom{h-1+l}{l} \right) - 1 \\ &= n^l \cdot \binom{h+l}{l} - 1\,. \end{aligned}$$

In effect, $R(h,l) \leq n^l \cdot (h+l)^l$. Recalling that we start with $l = 2 \cdot \lfloor \log n \rfloor$, we see that this number is quasi-polynomial in n and h. This concludes the proof, since obviously a single execution of the $\textsc{Solve}_E$ procedure (not counting the running time of recursive calls) costs polynomial time.

6 Correctness

We now justify correctness of the algorithm. This amounts to proving the following lemma.

▶ **Lemma 6.1.** *Procedure* $\textsc{Solve}_E(G,h,p_E,p_O)$ *returns a set* W_E *such that for every* $S \subseteq \mathsf{nodes}(G)$,

- *if S is a dominion for Even, and $|S| \leq p_E$, then $S \subseteq W_E$, and*
- *if S is a dominion for Odd, and $|S| \leq p_O$, then $S \cap W_E = \emptyset$.*

Notice that in G there may be nodes that do not belong to any dominion smaller than p_E or p_O; for such nodes we do not specify whether or not they are contained in W_E.

Recall that $Win_E(G)$ is a dominion for Even, and $Win_O(G)$ is a dominion for Odd. Thus, using Lemma 6.1 we can conclude that for $p_E = p_O = n$ the procedure returns $Win_E(G)$, the winning region of Even.

One may wonder why we use dominions in the statement of the lemma, instead of simply saying that if $|Win_E(G)| \leq p_E$, then $Win_E(G) \subseteq W_E$. Such a simplified statement, however, is not suitable for induction. Indeed, while switching from the game G to the game H (created in lines 7, 12, 17) the winning regions of Even may increase dramatically, because in H Odd is not allowed to visit any node with priority h. Nevertheless, the winning region of Even in G, and any dominion of Even in G, remains a dominion in H (when restricted to nodes of H).

Before proving Lemma 6.1, let us observe two facts about dominions. In their statements $P \in \{\mathrm{Even}, \mathrm{Odd}\}$ is one of the players, and $\overline{P}$ is his opponent.

▶ **Fact 6.2.** *If S is a dominion for P in a game G, and X is a set of nodes of G, then $S \setminus \mathrm{ATR}_P(G, X)$ is a dominion for P in $G \setminus \mathrm{ATR}_P(G, X)$.*

Proof. Denote $S' = S \setminus \mathrm{ATR}_P(G, X)$ and $G' = G \setminus \mathrm{ATR}_P(G, X)$. By definition, from every node $v \in S$ player P wins with the condition $\mathrm{Limsup}P \cap \mathrm{Safety}(S)$ in G, using some winning strategy. Observe that using the same strategy he wins with the condition $\mathrm{Limsup}P \cap \mathrm{Safety}(S')$ in G' (assuming that the starting node v is in S'). The strategy remains valid in G', because every node u of player P that remains in G' has the same successors in G' as in G (conversely: if some of successors of u belongs to $\mathrm{ATR}_P(G, X)$, then u also belongs to $\mathrm{ATR}_P(G, X)$). ◀

▶ **Fact 6.3.** *If S is a dominion for P in a game G, and X is a set of nodes of G such that $S \cap X = \emptyset$, then S is a dominion for P in $G \setminus \mathrm{ATR}_{\overline{P}}(G, X)$ (in particular $S \subseteq \mathsf{nodes}(G \setminus \mathrm{ATR}_{\overline{P}}(G, X))$).*

Proof. Denote $G' = G \setminus \mathrm{ATR}_{\overline{P}}(G, X)$. Suppose that there is some $v \in S \cap \mathrm{ATR}_{\overline{P}}(G, X)$. On the one hand, P can guarantee that, while starting from v, the play stays in S (by the definition of a dominion); on the other hand, $\overline{P}$ can force to reach the set X (by the definition of an attractor), which is disjoint from S. Thus such a node v could not exist, we have $S \subseteq \mathsf{nodes}(G')$.

It remains to observe that from every node $v \in S$ player P wins with the condition $\mathrm{Limsup}P \cap \mathrm{Safety}(S)$ also in the restricted game G', using the same strategy as in G. Indeed, a play in G following this strategy never leaves S, and the whole S remains unchanged in G'. ◀

We are now ready to prove Lemma 6.1.

Proof of Lemma 6.1. We prove the lemma by induction on h. Consider some execution of the procedure. By G^i, N_h^i, H^i, W_O^i we denote values of the variables G, N_h, H, W_O just after the i-th call to SOLVE_O in one of the lines 8, 13, 18; in lines 9, 14, 19 we create G^{i+1} out of G^i and W_O^i. In particular G^1 equals the original game G, and at the end we return $\mathsf{nodes}(G^{m+1})$, where m is the number of calls to SOLVE_O.

Concentrate on the first item of the lemma: fix an Even's dominion S in G (i.e., in G^1) such that $|S| \leq p_E$. Assume that $S \neq \emptyset$ (for $S = \emptyset$ there is nothing to prove). Notice first that a nonempty dominion has at least two nodes (by assumption there are no self-loops in G, hence every play has to visit at least two nodes), thus, because $S \subseteq \mathsf{nodes}(G)$ and $|S| \leq p_E$, we have that $G \neq \emptyset$ and $p_E > 1$. It means that the procedure does not return in line 4. We thus need to prove that $S \subseteq \mathsf{nodes}(G^{m+1})$.

We actually prove that S is a dominion for Even in G^i for every $i \in \{1, \dots, m+1\}$, meaning in particular that $S \subseteq \mathsf{nodes}(G^i)$. This is shown by an internal induction on i. The base case ($i = 1$) holds by assumption. For the induction step, consider some $i \in \{1, \dots, m\}$. By the induction assumption S is a dominion for Even in G^i, and we need to prove that it is a dominion for Even in G^{i+1}.

Consider $S^i = S \cap \mathsf{nodes}(H^i)$. Because $S^i = S \setminus \textsc{Atr}_E(G^i, N^i_h)$, by Fact 6.2 the set S^i is a dominion for Even in $H^i = G^i \setminus \textsc{Atr}_E(G^i, N^i_h)$, and obviously $|S^i| \leq |S| \leq p_E$. By the assumption of the external induction (which can be applied to $\textsc{Solve}_O$, by symmetry) it follows that $S^i \cap W^i_O = \emptyset$, so also $S \cap W^i_O = \emptyset$ (because W^i_O contains only nodes of G^i, while $S \setminus S^i$ contains no nodes of G^i). Thus, by Fact 6.3 the set S is a dominion for Even in $G^{i+1} = G^i \setminus \textsc{Atr}_O(G^i, W^i_O)$. This finishes the proof of the first item.

Now we prove the second item of the lemma. To this end, fix some Odd's dominion S in G such that $|S| \leq p_O$. If $p_E \leq 1$, we return $W_E = \emptyset$ (line 4), so clearly $S \cap W_E = \emptyset$. The interesting case is when $p_E \geq 2$. Denote $S^i = S \cap \mathsf{nodes}(G^i)$ for all $i \in \{1, \dots, m+1\}$; we first prove that S^i is a dominion for Odd in G^i. This is shown by induction on i. The base case of $i = 1$ holds by assumption, because $G^1 = G$ and $S^1 = S$. For the induction step, assume that S^i is a dominion for Odd in G^i, for some $i \in \{1, \dots, m\}$. By definition $G^{i+1} = G^i \setminus \textsc{Atr}_O(G^i, W^i_O)$ and $S^{i+1} = S^i \setminus \textsc{Atr}_O(G^i, W^i_O)$, so S^{i+1} is a dominion for Odd in G^{i+1} by Fact 6.2, which finishes the inductive proof.

For $i \in \{1, \dots, m\}$, let Z^i be the set of nodes (in $S^i \setminus N^i_h$) from which Odd wins with the condition $\mathsf{LimsupOdd} \cap \mathsf{Safety}(S^i \setminus N^i_h)$ in G^i (that is, where Odd can win without seeing priority h – the highest even priority). Let us observe that if $S^i \neq \emptyset$ then $Z^i \neq \emptyset$ $(\clubsuit)$. Indeed, suppose to the contrary that $Z^i = \emptyset$, and consider an Odd's strategy that allows him to win with the condition $\mathsf{LimsupOdd} \cap \mathsf{Safety}(S^i)$ in G^i, from some node $v_0 \in S^i$. Because $v_0 \notin Z^i$, this strategy in not winning for the condition $\mathsf{LimsupOdd} \cap \mathsf{Safety}(S^i \setminus N^i_h)$, so Even, while playing against this strategy, can reach a node v_1 in N^i_h (as he cannot violate the parity condition nor leave S^i). For the same reason, because $v_1 \notin Z^i$, Even can continue and reach a node v_2 in N^i_h. Repeating this forever, Even gets priority h (which is even and is the highest priority) infinitely many times, contradicting the fact that the strategy was winning for Odd.

Observe also that from nodes of Z^i Odd can actually win with the condition $\mathsf{LimsupOdd} \cap \mathsf{Safety}(Z^i)$ in G^i, using the strategy that allows him to win with the condition $\mathsf{LimsupOdd} \cap \mathsf{Safety}(S^i \setminus N^i_h)$. Indeed, if a play following this strategy enters some node v, then from this node v Odd can still win with the condition $\mathsf{LimsupOdd} \cap \mathsf{Safety}(S^i \setminus N^i_h)$, which means that these nodes belongs to Z^i. It follows that Z^i is a dominion for Odd in G^i. Moreover, because $Z^i \cap N^i_h = \emptyset$, from Fact 6.3 we have that Z^i is a dominion for Odd in $H^i = G^i \setminus \textsc{Atr}_E(G^i, N^i_h)$.

Let k be the number of the call to $\textsc{Solve}_O$ that is performed in line 13 (calls number $1, \dots, k-1$ are performed in line 8, and calls number $k+1, \dots, m$ are performed in line 18). Recall that W^i_O is the set returned by a call to $\textsc{Solve}_O(H^i, h-1, p^i_O, p_E)$, where $p^k_O = p_O$, and $p^i_O = \lfloor \frac{p_O}{2} \rfloor$ if $i \neq k$. From the assumption of the external induction, if $|Z^i| \leq \lfloor \frac{p_O}{2} \rfloor$ or if $i = k$ (since $Z^i \subseteq S^i \subseteq S$ and $|S| \leq p_O$, clearly $|Z^i| \leq p_O$), we obtain that $Z^i \subseteq W^i_O$ $(\spadesuit)$.

We now prove that $|S^{k+1}| \leq \lfloor \frac{p_O}{2} \rfloor$. This clearly holds if $S^{k-1} = \emptyset$, because $S^{k+1} \subseteq S^k \subseteq S^{k-1}$. Suppose thus that $S^{k-1} \neq \emptyset$. Then $Z^{k-1} \neq \emptyset$, by $(\clubsuit)$. On the other hand, $W^{k-1}_O = \emptyset$, because we are just about to leave the loop in lines 5-10 (the k-th call to $\textsc{Solve}_O$ is in line 13). By $(\spadesuit)$, if $|Z^{k-1}| \leq \lfloor \frac{p_O}{2} \rfloor$, then $Z^{k-1} \subseteq W^{k-1}_O$, which does not hold in our case. Thus $|Z^{k-1}| > \lfloor \frac{p_O}{2} \rfloor$. Because $W^{k-1}_O = \emptyset$, we simply have $G^k = G^{k-1}$, and $S^k = S^{k-1}$, and $Z^k = Z^{k-1}$. Using $(\spadesuit)$ for $i = k$, we obtain that $Z^k \subseteq W^k_O$, and because $S^{k+1} = S^k \setminus \textsc{Atr}_O(G^k, W^k_O) \subseteq S^k \setminus W^k_O \subseteq S^k \setminus Z^k$ we obtain that $|S^{k+1}| \leq |S^k| - |Z^k| \leq p_O - (\lfloor \frac{p_O}{2} \rfloor + 1) \leq \lfloor \frac{p_O}{2} \rfloor$, as initially claimed.

If $k = m$, we have $Z^m \subseteq W_O^m$ by (♠). If $k+1 \leq m$, we have $S^m \subseteq S^{k+1}$ (our procedure only removes nodes from the game) and $Z^m \subseteq S^m$, so $|Z^m| \leq \lfloor \frac{p_O}{2} \rfloor$ by the above paragraph, and also $Z^m \subseteq W_O^m$ by (♠). Because after the m-th call to SOLVE$_O$ the procedure ends, we have $W_O^m = \emptyset$, so also $Z^m = \emptyset$, and thus $S^m = \emptyset$ by (♣). We have $S^{m+1} \subseteq S^m$, so $S^{m+1} = S \cap \mathsf{nodes}(G^{m+1}) = \emptyset$. This is exactly the conclusion of the lemma, since the set returned by the procedure is $\mathsf{nodes}(G^{m+1})$. ◀

7 Conclusions

To the list of the four existing quasi-polynomial-time algorithms solving parity games, we have added a new one. It uses a rather different approach: it analyses recursively the game graph, like Zielonka's algorithm.

Notice that the number of recursive calls in our algorithm may be smaller than in the original Zielonka's algorithm, because of the precision parameters, but it may also be larger. Indeed, while SOLVE$_E$ in the original Zielonka's algorithm stops after the first time when a recursive call returns $\emptyset$, in our algorithm the procedure stops after the second time when a recursive call returns $\emptyset$.

The algorithm, as is, turns out not to be very efficient in practice. Beside of that, we believe that it can serve as a good starting point for a more optimized algorithm. Over the years, some optimizations to Zielonka's algorithm were proposed. For example, Liu, Duan, and Tian [26] replace the loop guard $W_O = \emptyset$ by $W_O = \text{ATR}_O(G, W_O)$ (which ensures that W_O will be empty in the next iteration of the loop). Verver [33] proposes to check whether $\text{ATR}_E(G, N_h)$ contains all nodes of priority $h-1$, and if so, to extend N_h by nodes of the next highest Even priority (i.e., $h-2$). It seems that these optimizations can be applied to our algorithm as well.

A straightforward optimization is to decrease p_O and p_E to $|G|$ at the beginning of every recursive call.

Another idea is to extend the recursive procedure so that it will return also a Boolean value saying whether the returned set surely equals the whole winning region (i.e., whether the precision parameters have not restricted anything). If while making the recursive call with smaller precision (line 8) the answer is positive, but the returned set W_O is empty, we can immediately stop the procedure, without making the recursive call with the full precision (line 13).

One can also observe that the call to SOLVE$_O$ in line 13 (with the full precision) gets the same subgame H as the last call to SOLVE$_O$ in line 8 (with decreased precision). A very rough idea is to make some use of the computations performed by the decreased-precision call during the full-precision call.

We leave implementation and evaluation of the above (and potentially some other) optimizations for a future work.

References

1 Massimo Benerecetti, Daniele Dell'Erba, and Fabio Mogavero. Robust Exponential Worst Cases for Divide-et-Impera Algorithms for Parity Games. In Patricia Bouyer, Andrea Orlandini, and Pierluigi San Pietro, editors, *Proceedings Eighth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017, Roma, Italy, 20-22 September 2017.*, volume 256 of *EPTCS*, pages 121–135, 2017. doi:10.4204/EPTCS.256.9.

2 Massimo Benerecetti, Daniele Dell'Erba, and Fabio Mogavero. Solving parity games via priority promotion. *Formal Methods in System Design*, 52(2):193–226, 2018. doi:10.1007/s10703-018-0315-1.

3 Henrik Björklund and Sergei G. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210–229, 2007. doi:10.1016/j.dam.2006.04.029.

4 Mikołaj Bojańczyk and Wojciech Czerwiński. An Automata Toolbox, February 2018. URL: https://www.mimuw.edu.pl/~bojan/papers/toolbox-reduced-feb6.pdf.

5 Anca Browne, Edmund M. Clarke, Somesh Jha, David E. Long, and Wilfredo R. Marrero. An Improved Algorithm for the Evaluation of Fixpoint Expressions. *Theor. Comput. Sci.*, 178(1-2):237–255, 1997. doi:10.1016/S0304-3975(96)00228-9.

6 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. 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 252–263. ACM, 2017. doi:10.1145/3055399.3055409.

7 Anne Condon. The Complexity of Stochastic Games. *Inf. Comput.*, 96(2):203–224, 1992. doi:10.1016/0890-5401(92)90048-K.

8 Wojciech Czerwiński, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdziński, Ranko Lazić, and Paweł Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2333–2349. SIAM, 2019. doi:10.1137/1.9781611975482.142.

9 Constantinos Daskalakis and Christos H. Papadimitriou. Continuous Local Search. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 790–804. SIAM, 2011. doi:10.1137/1.9781611973082.62.

10 E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model checking for the μ-calculus and its fragments. *Theor. Comput. Sci.*, 258(1-2):491–522, 2001. doi:10.1016/S0304-3975(00)00034-7.

11 John Fearnley. Exponential Lower Bounds for Policy Iteration. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 551–562. Springer, 2010. doi:10.1007/978-3-642-14162-1_46.

12 John Fearnley. Efficient Parallel Strategy Improvement for Parity Games. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 137–154. Springer, 2017. doi:10.1007/978-3-319-63390-9_8.

13 John Fearnley, Sanjay Jain, Sven Schewe, Frank Stephan, and Dominik Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In Hakan Erdogmus and Klaus Havelund, editors, *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017*, pages 112–121. ACM, 2017. doi:10.1145/3092282.3092286.

14 Oliver Friedmann. Recursive algorithm for parity games requires exponential time. *RAIRO - Theor. Inf. and Applic.*, 45(4):449–457, 2011. doi:10.1051/ita/2011124.

15 Oliver Friedmann, Thomas Dueholm Hansen, and Uri Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 283–292. ACM, 2011. doi:10.1145/1993636.1993675.

16 Maciej Gazda. *Fixpoint Logic, Games, and Relations of Consequence*. PhD thesis, Eindhoven University of Technology, 2016. URL: `https://pure.tue.nl/ws/files/16681817/20160315_Gazda.pdf`.

17 Hugo Gimbert and Rasmus Ibsen-Jensen. A short proof of correctness of the quasi-polynomial time algorithm for parity games. *CoRR*, abs/1702.01953, 2017. `arXiv:1702.01953`.

18 Marcin Jurdziński. Deciding the Winner in Parity Games is in UP ∩ co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998. `doi:10.1016/S0020-0190(98)00150-1`.

19 Marcin Jurdziński. Small Progress Measures for Solving Parity Games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000. `doi:10.1007/3-540-46541-3_24`.

20 Marcin Jurdziński and Ranko Lazić. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–9. IEEE Computer Society, 2017. `doi:10.1109/LICS.2017.8005092`.

21 Marcin Jurdziński, Mike Paterson, and Uri Zwick. A Deterministic Subexponential Algorithm for Solving Parity Games. *SIAM J. Comput.*, 38(4):1519–1532, 2008. `doi:10.1137/070686652`.

22 Jeroen J. A. Keiren. Benchmarks for Parity Games. In Mehdi Dastani and Marjan Sirjani, editors, *Fundamentals of Software Engineering - 6th International Conference, FSEN 2015 Tehran, Iran, April 22-24, 2015, Revised Selected Papers*, volume 9392 of *Lecture Notes in Computer Science*, pages 127–142. Springer, 2015. `doi:10.1007/978-3-319-24644-4_9`.

23 Bakhadyr Khoussainov. A Brief Excursion to Parity Games. In Mizuho Hoshi and Shinnosuke Seki, editors, *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, volume 11088 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2018. `doi:10.1007/978-3-319-98654-8_3`.

24 Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 639–648. ACM, 2018. `doi:10.1145/3209108.3209115`.

25 Karoliina Lehtinen, Sven Schewe, and Dominik Wojtczak. Improving the complexity of Parys' recursive algorithm. *CoRR*, abs/1904.11810, 2019. `arXiv:1904.11810`.

26 Yao Liu, Zhenhua Duan, and Cong Tian. An Improved Recursive Algorithm for Parity Games. In *2014 Theoretical Aspects of Software Engineering Conference, TASE 2014, Changsha, China, September 1-3, 2014*, pages 154–161. IEEE Computer Society, 2014. `doi:10.1109/TASE.2014.24`.

27 Donald A. Martin. Borel determinacy. *The Annals of Mathematics*, 102(2):363–371, 1975.

28 Robert McNaughton. Infinite Games Played on Finite Graphs. *Ann. Pure Appl. Logic*, 65(2):149–184, 1993. `doi:10.1016/0168-0072(93)90036-D`.

29 Michael Oser Rabin. *Automata on Infinite Objects and Church's Problem.* American Mathematical Society, Boston, MA, USA, 1972.

30 Sven Schewe. Solving parity games in big steps. *J. Comput. Syst. Sci.*, 84:243–262, 2017. `doi:10.1016/j.jcss.2016.10.002`.

31 Helmut Seidl. Fast and Simple Nested Fixpoints. *Inf. Process. Lett.*, 59(6):303–308, 1996. `doi:10.1016/0020-0190(96)00130-5`.

32 Tom van Dijk. Oink: An Implementation and Evaluation of Modern Parity Game Solvers. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I*, volume 10805 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2018. `doi:10.1007/978-3-319-89960-2_16`.

33 Maks Verver. Practical Improvements to Parity Game Solving. Master's thesis, University of Twente, 2013. URL: http://essay.utwente.nl/64985/1/practical-improvements-to-parity-game-solving.pdf.

34 Jens Vöge and Marcin Jurdziński. A Discrete Strategy Improvement Algorithm for Solving Parity Games. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215. Springer, 2000. doi:10.1007/10722167_18.

35 Wiesław Zielonka. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

36 Uri Zwick and Mike Paterson. The Complexity of Mean Payoff Games on Graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996. doi:10.1016/0304-3975(95)00188-3.

Bidding Mechanisms in Graph Games

Guy Avni
IST Austria, Klosterneuburg, Austria
guy.avni@ist.ac.at

Thomas A. Henzinger
IST Austria, Klosterneuburg, Austria
tah@ist.ac.at

Đorđe Žikelić
IST Austria, Klosterneuburg, Austria
djordje.zikelic@ist.ac.at

Abstract

In two-player games on graphs, the players move a token through a graph to produce a finite or infinite path, which determines the qualitative winner or quantitative payoff of the game. We study *bidding games* in which the players bid for the right to move the token. Several bidding rules were studied previously. In *Richman* bidding, in each round, the players simultaneously submit bids, and the higher bidder moves the token and pays the other player. *Poorman* bidding is similar except that the winner of the bidding pays the "bank" rather than the other player. *Taxman* bidding spans the spectrum between Richman and poorman bidding. They are parameterized by a constant $\tau \in [0,1]$: portion τ of the winning bid is paid to the other player, and portion $1-\tau$ to the bank. While finite-duration (reachability) taxman games have been studied before, we present, for the first time, results on *infinite-duration* taxman games. It was previously shown that both Richman and poorman infinite-duration games with qualitative objectives reduce to reachability games, and we show a similar result here. Our most interesting results concern quantitative taxman games, namely *mean-payoff* games, where poorman and Richman bidding differ significantly. A central quantity in these games is the *ratio* between the two players' initial budgets. While in poorman mean-payoff games, the optimal payoff of a player depends on the initial ratio, in Richman bidding, the payoff depends only on the structure of the game. In both games the optimal payoffs can be found using (different) probabilistic connections with *random-turn* games in which in each turn, instead of bidding, a coin is tossed to determine which player moves. While the value with Richman bidding equals the value of a random-turn game with an un-biased coin, with poorman bidding, the bias in the coin is the initial ratio of the budgets. We give a complete classification of mean-payoff taxman games that is based on a probabilistic connection: the value of a taxman bidding game with parameter τ and initial ratio r, equals the value of a random-turn game that uses a coin with bias $F(\tau,r)=\frac{r+\tau\cdot(1-r)}{1+\tau}$. Thus, we show that Richman bidding is the exception; namely, for every $\tau<1$, the value of the game depends on the initial ratio. Our proof technique simplifies and unifies the previous proof techniques for both Richman and poorman bidding.

2012 ACM Subject Classification Theory of computation → Solution concepts in game theory; Theory of computation → Formal languages and automata theory

Keywords and phrases Bidding games, Richman bidding, poorman bidding, taxman bidding, mean-payoff games, random-turn games

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.11

Related Version The full version can be found in `https://arxiv.org/abs/1905.03588`.

Funding This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE), Z211-N23 (Wittgenstein Award), and M 2369-N33 (Meitner fellowship), and from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 665385.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 11; pp. 11:1–11:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Two-player infinite-duration games on graphs are a central class of games in formal verification [2], where they are used, for example, to solve synthesis [19], and they have deep connections to foundations of logic [21]. A graph game proceeds by placing a token on a vertex in the graph, which the players move throughout the graph to produce an infinite path ("play") π. The game is zero-sum and π determines the winner or payoff. Graph games can be classified according to the players' objectives. For example, the simplest objective is *reachability*, where Player 1 wins iff an infinite path visits a designated target vertex. Another classification of graph games is the *mode of moving* the token. The most studied mode of moving is *turn based*, where the players alternate turns in moving the token.

In *bidding games*, in each turn, an "auction" is held between the two players in order to determine which player moves the token. The bidding mode of moving was introduced in [13, 14] for reachability games, where the following bidding rules where defined. In *Richman* bidding (named after David Richman), each player has a budget, and before each turn, the players submit bids simultaneously, where a bid is legal if it does not exceed the available budget. The player who bids higher wins the bidding, pays the bid to the other player, and moves the token. A second bidding rule called *poorman* bidding in [13], is similar except that the winner of the bidding pays the "bank" rather than the other player. Thus, the bid is deducted from his budget and the money is lost. A third bidding rule on which we focus in this paper, called *taxman* in [13] spans the spectrum between poorman and Richman bidding. Taxman bidding is parameterized by $\tau \in [0, 1]$: the winner of a bidding pays portion τ of his bid to the other player and portion $1 - \tau$ to the bank. Taxman bidding with $\tau = 1$ coincides with Richman bidding and taxman bidding with $\tau = 0$ coincides with poorman bidding.

Bidding games are relevant for several communities in Computer Science. In formal methods, graph games are used to reason about systems. Poorman bidding games naturally model concurrent systems where processes pay the scheduler for moving. Block-chain technology like Etherium is an example of such a system, which is a challenging to formally verify [9, 3]. In Algorithmic Game Theory [17], auction design is a central research topic that is motivated by the abundance of auctions for online advertisements [16]. Infinite-duration bidding games can model ongoing auctions and can be used to devise bidding strategies for objectives like: "In the long run, an advertiser's ad should show at least half of the time". In Artificial Intelligence, bidding games with Richman bidding have been used to reason about combinatorial negotiations [15]. Finally, *discrete-bidding* games [11], in which the granularity of the bids is restricted by assuming that the budgets are given using coins, have been studied mostly for recreational games, like bidding chess [6].

Both Richman and poorman infinite-duration games have a surprising, elegant, though different, mathematical structure as we elaborate below. Our study of taxman bidding aims at a better understanding of this structure and at shedding light on the differences between the seemingly similar bidding rules.

A central quantity in bidding games is the *initial ratio* of the players budgets. Formally, assuming that, for $i \in \{1, 2\}$, Player i's initial budget is B_i, we say that Player 1's initial ratio is $B_1/(B_1 + B_2)$. The central question that was studied in [13] regards the existence of a necessary and sufficient initial ratio to guarantee winning the game. Formally, the *threshold ratio* in a vertex v, denoted $\texttt{Th}(v)$, is such that if Player 1's initial ratio exceeds $\texttt{Th}(v)$, he can guarantee winning the game, and if his initial ratio is less than $\texttt{Th}(v)$, Player 2 can guarantee winning the game[1]. Existence of threshold ratios in reachability games for all three bidding mechanisms was shown in [13].

Reachability Richman-bidding games have an interesting probabilistic connection [14]. To state the connection, we first need to introduce *random-turn* games. Let $p \in [0, 1]$. In a random-turn game that is parameterized by p, in each turn, rather than bidding, the player who moves is chosen by throwing a (possibly) biased coin: with probability p, Player 1 chooses how to move the token, and Player 2 chooses with probability $1 - p$. Formally, a random-turn game is a special case of a stochastic game [10]. Consider a reachability Richman-bidding game $\mathcal{G}$. We construct a "uniform" random-turn game on top of $\mathcal{G}$, denoted $\mathtt{RT}^{0.5}(\mathcal{G})$, in which we throw an unbiased coin in each turn. The objective of Player 1 remains reaching his target vertex. It is well known that each vertex in $\mathtt{RT}^{0.5}(\mathcal{G})$ has a *value*, which is, informally, the probability of reaching the target when both players play optimally, and which we denote by $val(\mathtt{RT}^{0.5}(\mathcal{G}), v)$. We are ready to state the probabilistic connection: For every vertex v in the Richman game $\mathcal{G}$, the threshold ratio in v equals $1 - val(\mathtt{RT}^{0.5}(\mathcal{G}), v)$. We note that such a connection is not known and is unlikely to exist in reachability games with neither poorman nor taxman bidding. Random-turn games have been extensively studied in their own right, mostly with unbiased coin tosses, since the seminal paper [18].

Infinite-duration bidding games have been recently studied with Richman [4] and poorman [5] bidding. For qualitative objectives, namely games in which one player wins and the other player loses, both bidding rules have similar properties. By reducing general qualitative games to reachability games, it is shown that threshold ratios exist for both types of bidding rules. We show a similar result for qualitative games with taxman bidding.

Things get interesting in *mean-payoff* games, which are quantitative games: an infinite play has a *payoff*, which is Player 1's reward and Player 2's cost (see an example of a mean-payoff game in Figure 1). We thus call the players in a mean-payoff game Max and Min, respectively. We focus on games that are played on strongly-connected graphs. With Richman bidding [4], the initial budget of the players does not matter: A mean-payoff Richman-bidding game $\mathcal{G}$ has a value $c \in \mathbb{R}$ that depends only on the structure of the game such that Min can guarantee a cost of at most c with any positive budget, and with any positive budget, Max can guarantee a payoff of at least $c - \epsilon$, for every $\epsilon > 0$. Moreover, the value c of $\mathcal{G}$ equals the value of a random-turn game $\mathtt{RT}^{0.5}(\mathcal{G})$ that is constructed on top of $\mathcal{G}$. Since $\mathcal{G}$ is a mean-payoff game, $\mathtt{RT}^{0.5}(\mathcal{G})$ is a mean-payoff stochastic game, and its value, which again, is a well-known concept, is the expected payoff when both players play optimally.

Mean-payoff poorman-bidding games have different properties. Unlike with Richman bidding, the value of the game depends on the initial ratio. That is, with a higher initial ratio, Max can guarantee a better payoff. While the probabilistic connection for mean-payoff Richman games is not entirely unexpected given the probabilistic connection for reachability Richman games, we find it surprising that mean-payoff poorman games exhibit a probabilistic connection, which is in fact richer than for Richman bidding. The connection for poorman games is the following: Suppose Max's initial ratio is $r \in [0, 1]$ in a game $\mathcal{G}$. Then, the value in $\mathcal{G}$ with respect to r is the value of the random-turn game $\mathtt{RT}^r(\mathcal{G})$ in which in each turn, we toss a biased coin that chooses Max with probability r and Min with probability $1 - r$.

[1] When the initial ratio is exactly $\mathtt{Th}(v)$, the winner depends on the mechanism with which ties are broken. Our results do not depend on a specific tie-breaking mechanism.Tie-breaking mechanisms are particularly important in discrete-bidding games [1].

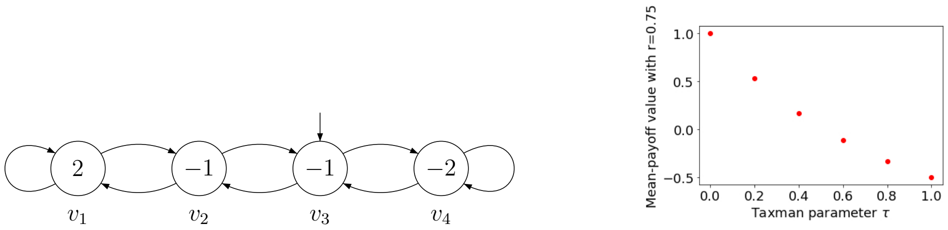


Figure 1 On the left, a mean-payoff game $\mathcal{G}$. On the right, the mean-payoff value of $\mathcal{G}$, where the initial ratio is fixed to 0.75 and the taxman parameter τ varies. The value of $\mathcal{G}$ with Richman bidding is -0.5, with poorman bidding, it is 1, and, for example, with $\tau = 0.2$, it is 0.533.

Given this difference between the two bidding rules, one may wonder how do mean-payoff taxman games behave, since these bidding rules span the spectrum between Richman and poorman bidding. Our main contribution is a complete solution to this question: we identify a probabilistic connection for a taxman game $\mathcal{G}$ that depends on the parameter τ of the bidding and the initial ratio r. That is, we show that the value of the game equals the value of the random-turn game $\mathtt{RT}^{F(\tau,r)}(\mathcal{G})$, where $F(\tau, r) = \frac{r+\tau\cdot(1-r)}{1+\tau}$. The construction gives rise to optimal strategies w.r.t. τ and the initial ratio. As a sanity check, note that for $\tau = 1$, we have $F(\tau, r) = 0.5$, which agrees with the result on Richman bidding, and for $\tau = 0$, we have $F(\tau, r) = r$, which agrees with the result on poorman bidding. In Figure 1, we depict some mean-payoff values for a fixed initial ratio and varying taxman parameter. Previous results only give the two endpoints in the plot, and the mid points in the plot are obtained using the results in this paper.

The main technical challenge is constructing an optimal strategy for Max, which, intuitively, performs a de-randomization; with a deterministic bidding strategy, Max guarantees that the ratio of the time that is spent in each vertex is the same as in a random behavior. The construction of Max strategy involves two components. First, we assign an "importance" to each vertex v, which we call *strength* and denote $\mathrm{St}(v)$. Intuitively, if $\mathrm{St}(v) > \mathrm{St}(u)$, then it is more important for Max to move in v than in u. Second, when the game reaches a vertex v, Max's bid is a careful normalization of $\mathrm{St}(v)$ so that changes in Max's ratio are matched with the accumulated weights in the game. Finding the right normalization is intricate and it consists of the main technical contribution of this paper. Previous such normalizations were constructed for Richman and poorman mean-payoff games [4, 5]. The construction for Richman bidding is much more complicated than the one we present here. The construction for poorman bidding is ad-hoc and does not generalize. Our construction for taxman bidding thus unifies these constructions and simplifies them. It uses techniques that can generalize beyond taxman bidding. Finally, we study, for the first time, complexity problems for taxman games.

Due to lack of space, some proofs appear in the full version.

2 Preliminaries

A graph game is played on a directed graph $G = \langle V, E\rangle$, where V is a finite set of vertices and $E \subseteq V \times V$ is a set of edges. The *neighbors* of a vertex $v \in V$, denoted $N(v)$, is the set of vertices $\{u \in V : \langle v, u\rangle \in E\}$. A *path* in G is a finite or infinite sequence of vertices $v_1, v_2, \ldots$ such that for every $i \geq 1$, we have $\langle v_i, v_{i+1}\rangle \in E$.

Bidding games. Each Player i has a budget $B_i \in \mathbb{R}^{\geq 0}$. In each turn a bidding determines which player moves the token. Both players simultaneously submit bids, where a bid b_i for Player i is legal if $b_i \leq B_i$. The player who bids higher wins the bidding, where we assume some mechanism to break ties, e.g., always giving Player 1 the advantage, and our results are not affected by the specific tie-breaking mechanism at use. The winner moves the token and pays his bid, where we consider three bidding mechanisms that differ in where the winning bid is paid. Suppose Player 1 wins a bidding with his bid of b.

- In **Richman** bidding, the winner pays to the loser, thus the new budgets are $B_1 - b$ and $B_2 + b$.
- In **poorman** bidding, the winner pays to the bank, thus the new budgets are $B_1 - b$ and B_2.
- In **taxman** bidding with parameter $\tau \in [0, 1]$, the winner pays portion τ to the other player and $(1 - \tau)$ to the bank, thus the new budgets are $B_1 - b$ and $B_2 + (1 - \tau) \cdot b$.

A central quantity in bidding games is the *ratio* of a player's budget from the total budget.

▶ **Definition 1** (Ratio). *Suppose the budget of Player i is B_i, for $i \in \{1, 2\}$, at some point in the game. Then, Player i's* ratio *is $B_i/(B_1 + B_2)$. The* initial ratio *refers to the ratio of the initial budgets, namely the budgets before the game begins. We restrict attention to games in which both players start with positive initial budgets, thus the initial ratio is in* $(0, 1)$.

Strategies and plays. A *strategy* is a recipe for how to play a game. It is a function that, given a finite *history* of the game, prescribes to a player which *action* to take, where we define these two notions below. For example, in turn-based games, a strategy takes as input, the sequence of vertices that were visited so far, and it outputs the next vertex to move to. In bidding games, histories and strategies are more involved as they maintain the information about the bids and winners of the bids. Formally, a history in a bidding game is $\pi = \langle v_1, b_1, i_1\rangle, \ldots, \langle v_k, b_k, i_k\rangle, v_{k+1} \in (V \times \mathbb{R} \times \{1, 2\})^* \cdot V$, where for $1 \leq j \leq k + 1$, the token is placed on vertex v_j at round j, for $1 \leq j \leq k$, the winning bid is b_j and the winner is Player i_j. Consider a finite history π. For $i \in \{1, 2\}$, let $W_i(\pi) \subseteq \{1, \ldots, k\}$ denote the indices in which Player i is the winner of the bidding in π. Let B^I_i be the initial budget of Player i. Player i's budget following π, denoted $B_i(\pi)$, depends on the bidding mechanism. For example, in Richman bidding, $B_1(\pi) = B^I_i - \sum_{j \in W_1(\pi)} b_j + \sum_{j \in W_2(\pi)} b_j$, B_2 is defined dually, and the definition is similar for taxman and poorman bidding. Given a history π that ends in v, a strategy for Player i prescribes an action $\langle b, v\rangle$, where $b \leq B_i(\pi)$ is a bid that does not exceed the available budget and v is a vertex to move to upon winning, where we require that v is a neighbor of v_{k+1}. An initial vertex, initial budgets, and two strategies for the players determine a unique infinite *play* π for the game. The vertices that π visits form an infinite path $path(\pi)$.

Objectives. An objective O is a set of infinite paths. Player 1 wins an infinite play π iff $path(\pi) \in O$. We call a strategy f *winning* for Player 1 w.r.t. an objective O if for every strategy g of Player 2 the play that f and g determine is winning for Player 1. Winning strategies for Player 2 are defined dually. We consider the following qualitative objectives:

1. In *reachability games*, Player 1 has a target vertex t and an infinite play is winning iff it visits t.

2. In *parity games*, each vertex is labeled with an index in $\{1, \ldots, d\}$. An infinite path is winning for Player 1 iff the parity of the maximal index that is visited infinitely often is odd.
3. *Mean-payoff games* are played on weighted directed graphs, with weights given by a function $w : V \to \mathbb{Q}$. Consider an infinite path $\eta = v_1, v_2, \dots \in V^\omega$. For $n \in \mathbb{N}$, the prefix of length n of η is η^n, and we define its *energy* to be $E(\eta^n) = \sum_{i=1}^{n} w(v_i)$. The *payoff* of η is $\mathtt{MP}(\eta) = \liminf_{n\to\infty} E(\eta^n)/n$. Player 1 wins η iff $\mathtt{MP}(\eta) \geq 0$.

Mean-payoff games are quantitative games. We think of the payoff as Player 1's reward and Player 2's cost, thus in mean-payoff games, we refer to Player 1 as Max and to Player 2 as Min.

Threshold ratios. The first question that arises in the context of bidding games asks what is the necessary and sufficient initial ratio to guarantee an objective.

▶ **Definition 2** (Threshold ratios). *Consider a bidding game $\mathcal{G}$, a vertex v, an initial ratio r, and an objective O for Player 1. The threshold ratio in v, denoted $\mathit{Th}(v)$, is a ratio in $[0, 1]$ such that if $r > \mathit{Th}(v)$, then Player 1 has a winning strategy that guarantees that O is satisfied, and if $r < \mathit{Th}(v)$, then Player 2 has a winning strategy that violates O.*

Random-turn games. A *stochastic game* [10] is a graph game in which the vertices are partitioned between two players and a *nature* player. As in turn-based games, whenever the game reaches a vertex that is controlled by Player i, for $i = 1, 2$, he choses how the game proceeds, and whenever the game reaches a vertex v that is controlled by nature, the next vertex is chosen according to a probability distribution that depends only on v.

Consider a bidding game $\mathcal{G}$ that is played on a graph $\langle V, E\rangle$. The *random-turn game* with ratio $r \in [0, 1]$ that is associated with $\mathcal{G}$ is a stochastic game that intuitively simulates the following process. In each turn we throw a biased coin that turns heads with probability r and tails with probability $1 - r$. If the coin turns heads, then Player 1 moves the token, and otherwise Player 2 moves the token. Formally, we define $\mathtt{RT}^r(\mathcal{G}) = \langle V_1, V_2, V_N, E, \Pr\rangle$, where each vertex in V is split into three vertices, each controlled by a different player, thus for $\alpha \in \{1, 2, N\}$, we have $V_\alpha = \{v_\alpha : v \in V\}$, nature vertices simulate the fact that Player 1 chooses the next move with probability r, thus $\Pr[v_N, v_1] = r = 1 - \Pr[v_N, v_2]$, and reaching a vertex that is controlled by one of the two players means that he chooses the next move, thus $E = \{\langle v_\alpha, u_N\rangle : \langle v, u\rangle \in E \text{ and } \alpha \in \{1, 2\}\}$. When $\mathcal{G}$ is a mean-payoff game, the vertices are weighted and we define the weights of v_1, v_2, and v_N to be equal to the weight of v.

The following definitions are standard, and we refer the reader to [20] for more details. A strategy in a stochastic game is similar to a turn-based game; namely, given the history of vertices visited so far, the strategy chooses the next vertex. Fixing two such strategies f and g for both players gives rise to a distribution $D(f, g)$ on infinite paths. Intuitively, Player 1's goal is to maximize the probability that his objective is met. An *optimal* strategy for Player 1 guarantees that the objective is met with probability at least c and, intuitively, he cannot do better, thus Player 2 has a strategy that guarantees that the objective is violated with probability at least $(1 - c)$. It is well known that optimal positional strategies exist for the objectives that we consider.

▶ **Definition 3** (Values in stochastic games). *Consider a bidding game $\mathcal{G}$, let $r \in [0, 1]$, and consider two optimal strategies f and g for the two players in $\mathit{RT}^r(\mathcal{G})$. When $\mathcal{G}$ is a qualitative game with objective O, the* value *of $\mathit{RT}^r(\mathcal{G})$, denoted $val(\mathit{RT}^r(\mathcal{G}))$, is $\Pr_{\eta\sim D(f,g)} \Pr[\eta \in O]$. When $\mathcal{G}$ is a mean-payoff game, the* mean-payoff value *of $\mathit{RT}^r(\mathcal{G})$, denoted $\mathit{MP}(\mathit{RT}^r(\mathcal{G}))$, is $\mathbb{E}_{\eta\in D(f,g)}\mathit{MP}(\eta)$.*

3 Qualitative Taxman Games

In [14, 13], reachability bidding games were studied with a slightly different definition, which we call *double-reachability*: both players have a target, where we denote by t_i the target of Player i, for $i \in \{1,2\}$, all vertices have a path to both targets, and the game ends once one of the targets is reached. They show the following results.

▶ **Theorem 4.** *[14, 13] Consider a double-reachability bidding game $\mathcal{G}$ and a vertex v. The threshold ratio exists in v with Richman, poorman, and taxman bidding. Moreover, threshold ratios have the following properties. For the target vertex t_1 of Player 1, we have $\mathit{Th}(t_1) = 0$, and for the target t_2 of Player 2, we have $\mathit{Th}(v) = 1$. Consider some other vertex v and denote $v^+, v^- \in N(v)$ the vertices with the minimal and maximal thresholds in the neighborhood of v, thus for every $u \in N(v)$, we have $\mathit{Th}(v^-) \leq \mathit{Th}(u) \leq \mathit{Th}(v^+)$.*

- *In Richman bidding, we have $\mathit{Th}(v) = \frac{1}{2}\big(\mathit{Th}(v^+) + \mathit{Th}(v^-)\big)$.*
- *In poorman bidding, we have $\mathit{Th}(v) = \mathit{Th}(v^+)/(1 + \mathit{Th}(v^+) - \mathit{Th}(v^-))$.*
- *In taxman bidding with parameter τ, we have $\mathit{Th}(v) = \big(\mathit{Th}(v^-) + \mathit{Th}(v^+) - \tau \cdot \mathit{Th}(v^-)\big)/\big(2 - \tau \cdot (1 + \mathit{Th}(v^-) - \mathit{Th}(v^+))\big)$.*

Moreover, only double-reachability Richman-bidding games exhibit the following probabilistic connection: for every vertex v, we have $\mathit{Th}(v) = 1 - val(\mathit{RT}^{0.5}(\mathcal{G}), v)$. Thus, for games played on finite graphs, the threshold ratios are all rational numbers. However, threshold ratios with poorman-bidding need not be rational in finite games.

The equivalence between double-reachability bidding games and reachability games with Richman- and poorman-bidding is shown in [4] and [5]. The following proposition is the key component in showing the equivalence as well as in the reduction from parity taxman games to reachability taxman games.

▶ **Lemma 5.** *Consider a reachability taxman game $\mathcal{G}$. Suppose that every vertex in $\mathcal{G}$ has a path to the target of Player 1. Then, for any taxman parameter, Player 1 wins from every vertex with any positive initial budget. Thus, for every vertex v, we have $\mathit{Th}(v) = 0$.*

Proof. Let $\mathcal{G} = \langle V, E, t\rangle$, where $n = |V| - 1$. Suppose the game starts from a vertex v, and let $\epsilon > 0$ be the initial budget of Player 1. Since there is a path from v to Player 1's target, there is a path of length at most n. Thus, if Player 1 wins n consecutive biddings, he wins the game. Intuitively, Player 1 carefully chooses n increasing bids such that if Player 2 wins one of these bids, Player 1's ratio increases by a constant over his initial budget. By repeatedly playing according to such a strategy, Player 1 guarantees that his ratio increases and will eventually allow him to win n biddings in a row. Formally, if $\tau = 0$, then $\mathcal{G}$ is a Richman game and the proof of the lemma can be found in [4]. Otherwise, pick a sufficiently large $r \in \mathbb{N}$ such that $\tau > \frac{2}{r-1}$ and $r \geq 3$. Fix $0 < m < \frac{\epsilon}{r^n}$. Player 1 proceeds as follows: after winning i times, for $0 \leq i$, he bids $m \cdot r^i$ and, upon winning the bidding, he moves towards t along any shortest path. Since $m + mr + \cdots + mr^{n-1} < mr^n < \epsilon$, Player 1 has sufficient budget to win n consecutive biddings. If Player 2 does not win any of the first n biddings, Player 1 wins the game. On the other hand, if Player 2 wins the k-th bidding with $1 \leq k \leq n$, we show in the full version that his ratio increases by a fixed amount $b = \frac{mr}{(1-\epsilon)(r-1)} > 0$. ◀

The following corollary shows the equivalence between reachability and double-reachability taxman games.

▶ **Corollary 6.** *Consider a reachability taxman game $\mathcal{G} = \langle V, E, t\rangle$. Let $S \subseteq V$ be the set of vertices that have no path to t. Let $T \subseteq V$ be a set of vertices such that $t \in T$ and every*

$u \in T$ has a path to t and no path to a vertex in S. Then, for every $v \in T$, we have $\mathtt{Th}(v) = 0$, for every $v \in S$, we have $\mathtt{Th}(v) = 1$. Let $\mathcal{G}'$ be a double-reachability taxman game that is obtained from $\mathcal{G}$ by merging the vertices in S and T into two targets t_1 and t_2 for Players 1 and 2, respectively. Then, for every $v \in (V \setminus (S \cup T))$, the threshold of v in $\mathcal{G}$ equals the threshold of v in $\mathcal{G}'$.

The following theorem, whose proof can be found in the full version, uses Lemma 5 to classify the bottom-strongly-connected components of a parity taxman game as those that are winning and losing for Player 1, thereby constructing a reachability taxman game.

▶ **Theorem 7.** *Parity taxman games are linearly reducible to reachability taxman games. Specifically, threshold ratios exist in parity taxman games.*

4 Mean-Payoff Taxman Games

This section consists of our main technical contribution. We start by showing a complete classification of the value in strongly-connected mean-payoff taxman games depending on the taxman parameter τ and the initial ratio. We then extend the solution to general games, where the solution to strongly-connected games constitutes the main ingredient in the solution of the general case.

4.1 Strongly-connected mean-payoff taxman games

We start by formally defining the value of a strongly-connected mean-payoff game. Lemma 5 implies that in a strongly-connected game, a player can draw the game from every vertex to any other vertex with any positive initial budget. Since mean-payoff objectives are prefix independent, it follows that the vertex from which the game starts does not matter. Indeed, if the game starts at a vertex v with Max having initial ratio $r + \epsilon$, then Max can use $\epsilon/2$ of his budget to draw the game to a vertex u and continue as if he starts the game with initial ratio $r + \epsilon/2$.

▶ **Definition 8** (Mean-payoff value). *Consider a strongly-connected mean-payoff game $\mathcal{G}$, a ratio $r \in (0,1)$, and a taxman parameter $\tau \in [0,1]$. The mean-payoff value of $\mathcal{G}$ w.r.t. r and τ, is a value $c \in \mathbb{R}$ such that for every $\epsilon > 0$*

- *if Min's initial ratio is greater than $(1 - r)$, then he has a strategy that guarantees that the payoff is at most $c + \epsilon$, and*
- *if Max's initial ratio is greater than r, then he has a strategy that guarantees that the payoff is greater than $c - \epsilon$.*

The following theorem, which we prove in the next two sections, summarizes the properties of mean-payoff taxman games.

▶ **Theorem 9.** *Consider a strongly-connected mean-payoff taxman game $\mathcal{G}$ with taxman parameter $\tau \in [0,1]$ and an initial ratio $r \in (0,1)$. The value of $\mathcal{G}$ w.r.t. τ and r equals the value of the random-turn game $\mathtt{RT}^{F(\tau,r)}(\mathcal{G})$ in which Max is chosen to move with probability $F(\tau,r)$ and Min with probability $1 - F(\tau,r)$, where $F(\tau,r) = \frac{r+\tau(1-r)}{1+\tau}$.*

We show that in order to prove Theorem 9, it suffices to prove the following intermediate lemma, whose proof can be found in the full version, and follows from the advantage of Min in the definition of payoff.

▶ **Lemma 10.** *Consider a strongly-connected mean-payoff taxman game $\mathcal{G}$, a taxman parameter τ, and an initial ratio $r \in (0,1)$ such that $\mathit{MP}(\mathit{RT}^{F(\tau,r)}) = 0$ for $F(\tau, r) = \frac{r+\tau(1-r)}{1+\tau}$. Then, for every $\epsilon > 0$ Max has a strategy that guarantees that no matter how Min plays, the payoff is greater than $-\epsilon$.*

4.2 The importance of moving

The first part of the construction of an optimal strategy for Max as in Lemma 10 is to assign, to each vertex $v \in V$, a *strength*, denoted $\mathrm{St}(v)$, where $\mathrm{St}(v) \in \mathbb{Q}_{\geq 0}$. Intuitively, if $\mathrm{St}(v) > \mathrm{St}(u)$, for $u, v \in V$, it is more important for Max to move in v than it is in u. We follow the construction in [5], which uses the concept of *potentials*, which is a well-known concept in stochastic games (see [20]) and was originally defined in the context of the strategy iteration algorithm [12]. For completeness, we present the definitions below.

Consider a strongly-connected mean-payoff game $\mathcal{G}$, and let $p \in [0,1]$. Let f and g be two optimal positional strategies in $\mathtt{RT}^p(\mathcal{G})$, for Min and Max, respectively. For a vertex $v \in V$, let $v^-, v^+ \in V$ be such that Max proceeds from v to v^+ according to g and Min proceeds from v to v^- according to f. It is not hard to see that the mean-payoff value in all vertices in $\mathtt{RT}^p(\mathcal{G})$ is the same and we denote it by $\mathtt{MP}(\mathtt{RT}^p(\mathcal{G}))$. We denote the potential of v by $\mathrm{Pot}^p(v)$ and the strength of v by $\mathrm{St}^p(v)$, and we define them as follows.

$$\mathrm{Pot}^p(v) = p \cdot \mathrm{Pot}^p(v^+) + (1-p) \cdot \mathrm{Pot}^p(v^-) + w(v) - \mathtt{MP}(\mathtt{RT}^p(\mathcal{G})) \text{ and}$$
$$\mathrm{St}^p(v) = p \cdot (1-p) \cdot \big(\mathrm{Pot}^p(v^+) - \mathrm{Pot}^p(v^-)\big)$$

There are optimal strategies for which $\mathrm{Pot}^p(v^-) \leq \mathrm{Pot}^p(v') \leq \mathrm{Pot}^p(v^+)$, for every $v' \in N(v)$, which can be found, for example, using the strategy iteration algorithm. Note that $\mathrm{St}(v) \geq 0$, for every $v \in V$.

Consider a finite path $\pi = v_1, \ldots, v_n$ in $\mathcal{G}$. We intuitively think of π as a play, where for every $1 \leq i < n$, the bid of Max in v_i is $\mathrm{St}(v_i)$ and he moves to v_i^+ upon winning. Thus, if $v_{i+1} = v_i^+$, we say that Max won in v_i, and if $v_{i+1} \neq v_i^+$, we say that Max lost in v_i. Let $W(\pi)$ and $L(\pi)$ respectively be the indices in which Max wins and loses in π. We call Max wins *investments* and Max loses *gains*, where intuitively he *invests* in increasing the energy and *gains* a higher ratio of the budget whenever the energy decreases. Let $G(\pi)$ and $I(\pi)$ be the sum of gains and investments in π, respectively, thus $G(\pi) = \sum_{i \in L(\pi)} \mathrm{St}(v_i)$ and $I(\pi) = \sum_{i \in W(\pi)} \mathrm{St}(v_i)$. Recall that the energy of π is $E(\pi) = \sum_{1 \leq i < n} w(v_i)$. The following lemma, whose proof can be found in the full version, which generalizes a similar lemma in [5], connects the strength with the change in energy.

▶ **Lemma 11.** *Consider a strongly-connected mean-payoff game $\mathcal{G}$ and $p \in [0,1]$. For every finite path $\pi = v_1, \ldots, v_n$ in $\mathcal{G}$, we have $\mathit{Pot}^p(v_1) - \mathit{Pot}^p(v_n) + (n-1) \cdot \mathit{MP}(\mathit{RT}^p(\mathcal{G})) \leq E(\pi) + G(\pi)/(1-p) - I(\pi)/p$. In particular, when $p = \nu/(\mu+\nu)$ for $\nu, \mu > 0$, there is a constant $P = \min_v \mathit{Pot}^p(v) - \max_v \mathit{Pot}^p(v)$ such that $\frac{\nu \cdot \mu}{\nu + \mu} \cdot \big(E(\pi) - P - (n-1) \cdot \mathit{MP}(\mathit{RT}^{\frac{\nu}{\mu+\nu}}(\mathcal{G}))\big) \geq \mu \cdot I(\pi) - \nu \cdot G(\pi)$.*

4.3 Normalizing the bids

Whenever the game reaches a vertex v, Max's bid is obtained by carefully normalizing the strength of v. More formally, assuming an initial ratio r, in v, Max bids $r \cdot (1-r) \cdot \mathrm{St}(v) \cdot \beta_x$, where β_x is the normalization factor and $x \in \mathbb{R}_{\geq 1}$. In this section we show how to choose the normalization factor. We associate with every $x \geq 1$, two numbers: a ratio r_x and β_x both in $(0,1]$. We think of $(r_x)_{x \geq 1}$ as a sequence and a play gives rise to a walk on the sequence, which corresponds to the changes in energy in the bidding game. When the walk is in $x \geq 1$, Max uses the normalization factor β_x. If Max wins a bidding, we take a step up on the sequence, modeling the increase of energy, and when Min wins, we talk a step down. The size of the step depends on the strength of v. We prove existence of sequences with properties given in the following lemma and formally define Max's strategy after it.

▶ **Lemma 12.** *Consider a game $\mathcal{G}$, a finite set of non-negative strengths $S \subseteq \mathbb{R}_{\geq 0}$, a ratio $r \in (0,1)$, and a taxman parameter $\tau \in [0,1]$. For every $K > \frac{\tau r^2 + r(1-r)}{\tau(1-r)^2 + r(1-r)}$ there exist sequences $(r_x)_{x \geq 1}$ and $(\beta_x)_{x \geq 1}$ with the following properties.*

1. *Max's bid does not exceed his budget, thus, for each position $x \in \mathbb{R}_{\geq 1}$ and strength $s \in S$, we have $\beta_x \cdot s \cdot r \cdot (r-1) < r_x$.*
2. *Min cannot force the game beyond position 1, thus for every $s \in S \backslash \{0\}$ and $1 \leq x < 1+rs$, we have $\beta_x \cdot s \cdot r \cdot (r-1) > 1 - r_x$.*
3. *The ratios tend to r from above, thus for every $x \in \mathbb{R}_{\geq 1}$, we have $r_x \geq r$, and $\lim_{x \to \infty} r_x = r$.*
4. *No matter who wins a bidding, Max's ratio can only improve. Thus, in case of winning and in case of losing, we respectively have*
$$\frac{r_x - \beta_x \cdot s \cdot r \cdot (r-1)}{1 - (1-\tau) \cdot \beta_x \cdot s \cdot r \cdot (r-1)} \geq r_{x+(1-r) \cdot K \cdot s} \textit{ and } \frac{r_x + \tau \cdot \beta_x \cdot s \cdot r \cdot (r-1)}{1 - (1-\tau) \cdot \beta_x \cdot s \cdot r \cdot (r-1)} \geq r_{x - s \cdot r}$$

We first show how Lemma 12 implies Theorem 9.

Proof that Lemma 12 implies Lemma 10. Fix $\epsilon > 0$, we construct strategy for Max guaranteeing a payoff greater than $-\epsilon$, as wanted. Observe that

$$\frac{r}{r + (1-r)\frac{\tau r^2 + r(1-r)}{\tau(1-r)^2 + r(1-r)}} = \frac{r(\tau(1-r)+r)}{\tau r(1-r) + r^2 + \tau r^2 + r(1-r)} = \frac{r + \tau(1-r)}{1+\tau} = F(\tau, r).$$

Thus, since by assumption $\texttt{MP}(\texttt{RT}^{F(\tau,r)}(\mathcal{G})) = 0$ and $\texttt{MP}(\texttt{RT}^{p}(\mathcal{G}))$ is a continuous function in $p \in [0,1]$ [8, 22], we can pick $K > F(\tau, r)$ such that $\texttt{MP}(\texttt{RT}^{\frac{r}{r+(1-r)K}}(\mathcal{G})) > -\epsilon$.

We now describe Max's strategy. We think of the change in Max's ratio as a walk on $\mathbb{R}_{\geq 1}$. Each position $x \in \mathbb{R}_{\geq 1}$ is associated with a ratio r_x. The walk starts in a position x_0 such that Max's initial ratio is at least r_{x_0}. Let $\nu = r$ and $\mu = K(1-r)$. Suppose the token is placed on a vertex $v \in V$. Then, Max's bid is $r \cdot (1-r) \cdot \beta_x \cdot \mathrm{St}(v)$, where the ratios of Max and Min are normalized to sum up to 1, and he proceeds to v^+ upon winning. If Max wins, the walk proceeds up $\mu \cdot \mathrm{St}(v)$ steps to $x + \mu \mathrm{St}(v)$, and if he loses, the walk proceeds down to $x - \nu \mathrm{St}(v)$. Suppose Min fixes some strategy and let $\pi = v_1, \ldots, v_n$ be a finite prefix of the play that is generated by the two strategies. Suppose the walk following π reaches $x \in \mathbb{R}$. Then, using the terminology of the previous section, we have $x = x_0 - G(\pi) \cdot \nu + I(\pi) \cdot \mu$. Lemma 12 shows that the walk always stays above 1, thus $x \geq 1$. Combining with Lemma 11, we get $\frac{\nu+\mu}{\nu \cdot \mu}(1 - x_0) + P + (n-1) \cdot \texttt{MP}(\texttt{RT}^{\frac{\nu}{\nu+\mu}}(\mathcal{G})) \leq E(\pi)$. Thus, dividing both sides by n and letting $n \to \infty$, since x_0 and P are constants depending only on K we conclude that this strategy guarantees payoff at least $\texttt{MP}(\texttt{RT}^{\frac{\nu}{\nu+\mu}}(\mathcal{G})) > -\epsilon$. ◀

We continue to prove Lemma 12.

Proof of Lemma 12. Note that $\frac{\tau r^2 + r(1-r)}{\tau(1-r)^2 + r(1-r)}$ is well-defined for $r \in (0,1)$. Fix $\tau \in [0,1]$ and $r \in (0,1)$. Let $K > \frac{\tau r^2 + r(1-r)}{\tau(1-r)^2 + r(1-r)}$. Observe that the two inequalities in Point 4 are equivalent to:

$$\begin{aligned} r_{x-rs} - r_x &\leq \tau r(1-r)\beta_x s + (1-\tau) r(1-r)\beta_x s r_{x-rs}, \\ r_x - r_{x+K(1-r)s} &\geq r(1-r)\beta_x s - (1-\tau) r(1-r)\beta_x s r_{x+K(1-r)s}. \end{aligned}$$

Point 3 combined with monotonicity in the above expressions, implies that we can replace the last term in each of them by r in order to obtain stronger inequalities. Therefore, it suffices for $(r_x)_{x\ge 1}$ and $(\beta_x)_{x\ge 1}$ to satisfy

$$\begin{aligned} r_{x-rs} - r_x &\le \tau r(1-r)\beta_x s + (1-\tau)r(1-r)\beta_x s r,\\ r_x - r_{x+K(1-r)s} &\ge r(1-r)\beta_x s - (1-\tau)r(1-r)\beta_x s r, \end{aligned}$$

which is equivalent to

$$\begin{aligned} r_{x-rs} - r_x &\le r(1-r)\beta_x s[\tau + (1-\tau)r],\\ r_x - r_{x+K(1-r)s} &\ge r(1-r)\beta_x s[1 - (1-\tau)r]. \end{aligned} \tag{1}$$

We seek $(r_x)_{x\ge 1}$ and $(\beta_x)_{x\ge 1}$ in the form $r_x = \gamma^{x-1} + (1-\gamma^{x-1})r$ and $\beta_x = \beta\gamma^{x-1}$ for some $\gamma, \beta \in (0,1)$. Note that this choice ensures Points 1 and 3. Therefore, we just need to show that we can find $\gamma, \beta \in [0,1]$ for which the inequalities in (1) hold for any $s \in S$. Substituting r_x and β_x in terms of γ and β, the inequalities in (1) reduce to

$$\begin{aligned} r_{x-rs} - r_x &= \gamma^{x-1}(\gamma^{-rs} - 1)(1-r) \overset{?}{\le} \beta\gamma^{x-1} r(1-r)s[\tau + (1-\tau)r],\\ r_x - r_{x+K(1-r)s} &= \gamma^{x-1}(1 - \gamma^{K(1-r)s})(1-r) \overset{?}{\ge} \beta\gamma^{x-1} r(1-r)s[1 - (1-\tau)r]. \end{aligned}$$

First, when $s = 0$, both sides of both inequalities are equal to 0 so both inequalities clearly hold. Recall that S is a finite set of non-negative strengths. Thus, when $s > 0$, it takes values in $0 < s_1 \le \ldots \le s_n$, and the above inequalities are equivalent to

$$\begin{aligned} &\gamma \ge \big(1 + \beta rs[\tau + (1-\tau)r]\big)^{-\frac{1}{rs}},\\ &\gamma \le \big(1 - \beta rs[1 - (1-\tau)r]\big)^{\frac{1}{K(1-r)s}}. \end{aligned} \tag{2}$$

Since both of these expressions are in $(0,1)$, to conclude that $\gamma, \beta \in (0,1)$ exist, it suffices to show that there is some $\beta \in (0,1)$ such that

$$\max_{s\in\{s_1,\ldots,s_n\}} \big(1 + \beta rs[\tau + (1-\tau)r]\big)^{-\frac{1}{rs}} \le \min_{s\in\{s_1,\ldots,s_n\}} \big(1 - \beta rs[1 - (1-\tau)r]\big)^{\frac{1}{K(1-r)s}}. \tag{3}$$

Note that the LHS of (3) is monotonically increasing in $s > 0$ whereas the RHS is monotonically decreasing in $s > 0$, therefore it suffices to find $\beta \in (0,1)$ for which

$$\big(1 + \beta rs_n[\tau + (1-\tau)r]\big)^{-\frac{1}{rs_n}} \le \big(1 - \beta rs_1[1 - (1-\tau)r]\big)^{\frac{1}{K(1-r)s_1}}. \tag{4}$$

By Taylor's theorem $(1+y)^\alpha = 1 + \alpha y + O(y^2)$, so Taylor expanding both sides of (4) in $\beta > 0$ we get

$$\begin{aligned} &\big(1 + \beta rs_n[\tau + (1-\tau)r]\big)^{-\frac{1}{rs_n}} = 1 - \beta[\tau + (1-\tau)r] + O(\beta^2),\\ &\big(1 - \beta rs_1[1 - (1-\tau)r]\big)^{\frac{1}{K(1-r)s_1}} = 1 - \beta\frac{r}{K(1-r)}[1 - (1-\tau)r] + O(\beta^2). \end{aligned}$$

Therefore, if we show that $[\tau + (1-\tau)r] > \frac{r}{K(1-r)}[1 - (1-\tau)r]$, the linear coefficient of β on the LHS of (4) will be strictly smaller than the linear coefficient of β on the RHS. Thus, for sufficiently small $\beta > 0$, (4) will hold, which concludes the proof of the lemma. This condition is equivalent to

$$K > \frac{r[1 - (1-\tau)r]}{(1-r)[\tau + (1-\tau)r]} = \frac{r[\tau r + (1-r)]}{(1-r)[\tau(1-r) + r]} = \frac{\tau r^2 + r(1-r)}{\tau(1-r)^2 + r(1-r)},$$

which is true by assumption. Thus, Points 1, 3, and 4 hold. In the full version, we show that Point 2 holds. ◀

4.4 General mean-payoff taxman games

We extend the solution to general games. Recall that the threshold ratio in mean-payoff games is a necessary and sufficient initial ratio with which Max can guarantee a payoff of at least 0.

▶ **Theorem 13.** *Threshold ratios exist in mean-payoff taxman games.*

Proof. Consider a mean-payoff taxman game $\mathcal{G} = \langle V, E, w\rangle$ with taxman parameter τ. If $\mathcal{G}$ is strongly-connected, then by Theorem 9, the threshold ratio in all vertices in $\mathcal{G}$ is the same and is $r \in (0, 1)$ for r such that $\texttt{MP}(\texttt{RT}^{F(\tau,r)}(\mathcal{G})) = 0$. If no such r exists, then either $\texttt{MP}(\texttt{RT}^{F(\tau,1)}(\mathcal{G})) < 0$, in which case the threshold ratios are 1, or $\texttt{MP}(\texttt{RT}^{F(\tau,0)}(\mathcal{G})) > 0$, in which case the threshold ratios are 0. The proof for general games follows along the same lines as the proof for reachability games. For each bottom strongly-connected component S_i of $\mathcal{G}$ we find the threshold ratio $r_i \in (0, 1)$ as in the above. We play a "generalized" reachability game on $\mathcal{G}$ as follows. The game ends once the token reaches one of the BSCCs in $\mathcal{G}$. Max wins the game iff the first time the game enters a BSCC S_i, Max's ratio is greater than r_i. Showing existence of threshold ratios in the generalized game follows the same argument as for reachability games [13]. ◀

5 Computational Complexity

We show, for the first time, computational complexity results for taxman games. We study the following problem, which we call THRESH: given a taxman game $\mathcal{G}$ with taxman parameter τ and a vertex v_0 in $\mathcal{G}$, decide whether $\texttt{Th}(v_0) \geq 0.5$. The correspondence in Theorem 9 gives the second part of the following theorem, and for the first part, in the full version, we show a reduction from THRESH to the *existential theory of the reals* [7].

▶ **Theorem 14.** *For taxman reachability, parity, and mean-payoff games THRESH is in PSPACE. For strongly-connected mean-payoff games, THRESH is in NP ∩ coNP.*

6 Discussion

We study, for the first time, infinite-duration taxman-bidding games, which span the spectrum between Richman and poorman bidding. For qualitative objectives, we show that the properties of taxman coincide with these of Richman and poorman bidding. For mean-payoff games, where Richman and poorman bidding have an elegant though surprisingly different mathematical structure, we show a complete understanding of taxman games. Our study of mean-payoff taxman games sheds light on these differences and similarities between the two bidding rules. Unlike previous proof techniques, which were ad-hoc, we expect our technique to be easier to generalize beyond taxman games, where they can be used to introduce concepts like multi-players or partial information into bidding games.

References

1 M. Aghajohari, G. Avni, and T. A. Henzinger. Determinacy in Discrete-Bidding Infinite-Duration Games. In *In Proc. 30th CONCUR*, 2019.

2 K.R. Apt and E. Grädel. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, 2011.

3 N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on Ethereum smart contracts. *IACR Cryptology ePrint Archive*, 2016:1007, 2016.

4 G. Avni, T. A. Henzinger, and V. Chonev. Infinite-Duration Bidding Games. *J. ACM*, 66(4):31:1–31:29, 2019.

5 G. Avni, T. A. Henzinger, and R. Ibsen-Jensen. Infinite-Duration Poorman-Bidding Games. In *Proc. 14th WINE*, volume 11316 of *LNCS*, pages 21–36. Springer, 2018.

6 J. Bhatt and S. Payne. Bidding Chess. *Math. Intelligencer*, 31:37–39, 2009.

7 J. F. Canny. Some Algebraic and Geometric Computations in PSPACE. In *Proc. 20th STOC*, pages 460–467, 1988.

8 K. Chatterjee. Robustness of Structurally Equivalent Concurrent Parity Games. In *Proc. 15th FoSSaCS*, pages 270–285, 2012.

9 K. Chatterjee, A. K. Goharshady, and Y. Velner. Quantitative Analysis of Smart Contracts. In *Proc. 27th ESOP*, pages 739–767, 2018.

10 A. Condon. The Complexity of Stochastic Games. *Inf. Comput.*, 96(2):203–224, 1992.

11 M. Develin and S. Payne. Discrete Bidding Games. *The Electronic Journal of Combinatorics*, 17(1):R85, 2010.

12 A. R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

13 A. J. Lazarus, D. E. Loeb, J. G. Propp, W. R. Stromquist, and D. H. Ullman. Combinatorial Games under Auction Play. *Games and Economic Behavior*, 27(2):229–264, 1999.

14 A. J. Lazarus, D. E. Loeb, J. G. Propp, and D. Ullman. Richman Games. *Games of No Chance*, 29:439–449, 1996.

15 R. Meir, G. Kalai, and M. Tennenholtz. Bidding games and efficient allocations. *Games and Economic Behavior*, 2018. `doi:10.1016/j.geb.2018.08.005`.

16 S. Muthukrishnan. Ad Exchanges: Research Issues. In *Proc. 5th WINE*, pages 1–12, 2009.

17 N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

18 Y. Peres, O. Schramm, S. Sheffield, and D. B. Wilson. Tug-of-war and the infinity Laplacian. *J. Amer. Math. Soc.*, 22:167–210, 2009.

19 A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *Proc. 16th POPL*, pages 179–190, 1989.

20 M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 2005.

21 M.O. Rabin. Decidability of Second Order Theories and Automata on Infinite Trees. *Transaction of the AMS*, 141:1–35, 1969.

22 E. Solan. Continuity of the value of competitive Markov decision processes. *Journal of Theoretical Probability*, 16:831–845, 2003.

Cluster Deletion on Interval Graphs and Split Related Graphs

Athanasios L. Konstantinidis
Department of Mathematics, University of Ioannina, Greece
skonstan@cc.uoi.gr

Charis Papadopoulos
Department of Mathematics, University of Ioannina, Greece
charis@cs.uoi.gr

Abstract

In the Cluster Deletion problem the goal is to remove the minimum number of edges of a given graph, such that every connected component of the resulting graph constitutes a clique. It is known that the decision version of Cluster Deletion is NP-complete on (P_5-free) chordal graphs, whereas Cluster Deletion is solved in polynomial time on split graphs. However, the existence of a polynomial-time algorithm of Cluster Deletion on interval graphs, a proper subclass of chordal graphs, remained a well-known open problem. Our main contribution is that we settle this problem in the affirmative, by providing a polynomial-time algorithm for Cluster Deletion on interval graphs. Moreover, despite the simple formulation of the algorithm on split graphs, we show that Cluster Deletion remains NP-complete on a natural and slight generalization of split graphs that constitutes a proper subclass of P_5-free chordal graphs. Although the later result arises from the already-known reduction for P_5-free chordal graphs, we give an alternative proof showing an interesting connection between edge-weighted and vertex-weighted variations of the problem. To complement our results, we provide faster and simpler polynomial-time algorithms for Cluster Deletion on subclasses of such a generalization of split graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Cluster deletion, interval graphs, split graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.12

Related Version `https://arxiv.org/abs/1904.09470`

Funding This research is co-financed by Greece and the European Union (European Social Fund – ESF) through the Operational Programme "Human Resources Development, Education and Lifelong Learning" in the context of the project "Strengthening Human Resources Research Potential via Doctorate Research" (MIS–5000432), implemented by the State Scholarships Foundation (IKY).

1 Introduction

In graph theoretic notions, *clustering* is the task of partitioning the vertices of the graph into subsets, called clusters, in such a way that there should be many edges within each cluster and relatively few edges between the clusters. In many applications, the clusters are restricted to induced cliques, as the represented data of each edge corresponds to a similarity value between two objects [18, 19]. Under the term cluster graph, which refers to a disjoint union of cliques, one may find a variety of applications that have been extensively studied [1, 5, 23]. Here we consider the Cluster Deletion problem which asks for a minimum number of edge deletions from an input graph, so that the resulting graph is a disjoint union of cliques. In the decision version of the problem, we are also given an integer k and we want to decide whether at most k edge deletions are enough to produce a cluster graph.

Although Cluster Deletion is NP-hard on general graphs [24], settling its complexity status restricted on graph classes has attracted several researchers. Regarding the maximum degree of a graph, Komusiewicz and Uhlmann [22] have shown an interesting dichotomy result: Cluster Deletion remains NP-hard on C_4-free graphs with maximum degree four, whereas it can be solved in polynomial time on graphs having maximum degree at most three. Quite recently, Golovach et al. [14] have shown that it remains NP-hard on planar graphs. For graph classes characterized by forbidden induced subgraphs, Gao et al. [11] showed that Cluster Deletion is NP-hard on $(C_5, P_5, \mathrm{bull}, \mathrm{fork}, \text{co-gem}, \text{4-pan}, \text{co-4-pan})$-free graphs and on $(2K_2, 3K_1)$-free graphs. Regarding H-free graphs, Grüttemeier et al. [16], showed a complexity dichotomy result for any graph H consisting of at most four vertices. In particular, for any graph H on four vertices with $H \notin \{P_4, \mathrm{paw}\}$, Cluster Deletion is NP-hard on H-free graphs, whereas it can be solved in polynomial time on P_4- or paw-free graphs [16]. Interestingly, Cluster Deletion remains NP-hard on P_5-free chordal graphs [3].

On the positive side, Cluster Deletion has been shown to be solved in polynomial time on cographs [11], proper interval graphs [3], split graphs [3], and P_4-reducible graphs [2]. More precisely, iteratively picking maximum cliques defines a clustering on the graph which actually gives an optimal solution on cographs (i.e., P_4-free graphs), as shown by Gao et al. in [11]. In fact, the greedy approach of selecting a maximum clique provides a 2-approximation algorithm, though not necessarily in polynomial-time [7]. As the problem is already NP-hard on chordal graphs [3], it is natural to consider subclasses of chordal graphs such as interval graphs and split graphs. Although for split graphs there is a simple polynomial-time algorithm, restricted to interval graphs only the complexity on proper interval graphs was determined by giving a solution that runs in polynomial-time [3]. Settling the complexity of Cluster Deletion on interval graphs, was left open [3, 2, 11].

For proper interval graphs, Bonomo et al. [3] characterized their optimal solution by consecutiveness of each cluster with respect to their natural ordering of the vertices. Based on this fact, a dynamic programming approach led to a polynomial-time algorithm. It is not difficult to see that such a consecutiveness does not hold on interval graphs, as potential clusters might require to break in the corresponding vertex ordering. Here we characterize an optimal solution of interval graphs whenever a cluster is required to break. In particular, we take advantage of their consecutive arrangement of maximal cliques and describe subproblems of maximal cliques containing the last vertex. One of our key observations is that the candidate clusters containing the last vertex can be enumerated in polynomial time given two vertex orderings of the graph. We further show that each such candidate cluster separates the graph in a recursive way with respect to optimal subsolutions, that enables to define our dynamic programming table to keep track about partial solutions. Thus, our algorithm for interval graphs suggests to consider a particular consecutiveness of a solution and apply a dynamic programming approach defined by two vertex orderings. The overall running time of our algorithm is $O(n^6)$ for an interval graph on n vertices.

Furthermore, we complement the previously-known NP-hardness of Cluster Deletion on P_5-free chordal graphs, by providing a proper subclass of such graphs for which we prove that the problem remains NP-hard. This result is inspired and motivated by the very simple characterization of an optimal solution on split graphs: either a maximal clique constitutes the only non-edgeless cluster, or there are exactly two non-edgeless clusters whenever there is a vertex of the independent set that is adjacent to all the vertices of the clique except one [3]. Due to the fact that true twins belong to the same cluster in an optimal solution, it is natural to consider true twins at the independent set, as they are expected not to influence the solution characterization. Surprisingly, we show that Cluster Deletion remains NP-complete even on such a slight generalization of split graphs. This is achieved by observing that the constructed graphs given in the reduction for P_5-free graphs [3], constitute such split-related graphs. However, here we give a different reduction that highlights an interesting connection between edge-weighted and vertex-weighted split graphs. We then study two different classes of such generalization of split graphs that can be viewed as the parallel of split graphs that admit disjoint clique-neighborhood and nested clique-neighborhood. For Cluster Deletion we provide polynomial-time algorithms on both classes of graphs. In particular, for the former case, a polynomial-time algorithm is already known and is achieved through computing a minimizer of submodular functions [3]. Here we provide a simpler and faster (linear-time) algorithm for Cluster Deletion on such graphs that avoids the usage of submodular functions minimization.

2 Preliminaries

All graphs considered here are simple and undirected. We refer to Diestel's classical book [8] for standard graph terminology that is undefined here. Two adjacent vertices u and v are called *true twins* if $N[u] = N[v]$, whereas two non-adjacent vertices x and y are called *false twins* if $N(u) = N(v)$. For a set of finite graphs $\mathcal{H}$, we say that a graph G is $\mathcal{H}$-free if G does not contain an induced subgraph isomorphic to any of the graphs of $\mathcal{H}$.

The problem of Cluster Deletion is formally defined as follows: given a graph $G = (V, E)$, the goal is to compute the minimum set $F \subseteq E(G)$ of edges such that every connected component of $G - F$ is a clique. A *cluster graph* is a P_3-free graph, or equivalently, any of its connected components is a clique. Thus, the task of Cluster Deletion is to turn the input graph G into a cluster graph by deleting the minimum number of edges. Let $S = C_1, \ldots, C_k$ be a solution of Cluster Deletion such that $G[C_i]$ is a clique. In such terms, the problem can be viewed as a vertex partition problem into $C_1, \ldots, C_k$. Each C_i is simple called *cluster*. Edgeless clusters, i.e., clusters containing exactly one vertex, are called *trivial clusters*. The edges of G are partitioned into *internal* and *external* edges: an internal edge uv has both its endpoints $u, v \in C_i$ in the same cluster C_i, whereas an external edge uv has its endpoints in different clusters $u \in C_i$ and $v \in C_j$, for $i \neq j$. Then, the goal of Cluster Deletion is to minimize the number of external edges which is equivalent to maximize the number of internal edges. We write $S(G)$ to denote an optimal solution for Cluster Deletion of the graph G, that is, a cluster subgraph of G having the maximum number of edges. Given a solution $S(G)$, the number of edges incident only to the same cluster, that is the number of internal edges, is denoted by $|S(G)|$.

For a clique C, we say that a vertex x is *C-compatible* if $C \setminus \{x\} \subseteq N(x)$. We start with few preliminary observations regarding twin vertices. Notice that for true twins x and y, if x belongs to any cluster C then y is C-compatible.

▶ **Lemma 1** ([3]). *Let x and y be true twins in G. Then, in any optimal solution x and y belong to the same cluster.*

The above lemma shows that we can contract true twins and look for a solution on a vertex-weighted graph that does not contain true twins. Notice, however, that the weights on the vertices imply weights on the edges of the graph, as they contribute to the total cost of external and internal edges in a solution. Even though false twins cannot be grouped into the same cluster as they are non-adjacent, we can actually disregard one of the false twins whenever their neighborhood forms a clique.

▶ **Lemma 2.** *Let x and y be false twins in G such that $N(x) = N(y)$ is a clique. Then, there is an optimal solution such that y constitutes a trivial cluster.*

Proof. Let C_x and C_y be the clusters of x and y, respectively, in an optimal solution such that $|C_x| \geq 2$ and $|C_y| \geq 2$. We construct another solution by replacing both clusters by $C_x \cup C_y \setminus \{y\}$ and $\{y\}$, respectively. To see that this indeed a solution, first observe that x is adjacent to all the vertices of $C_y \setminus \{y\}$ because $N(x) = N(y)$, and $C_x \cup C_y \setminus \{y\} \subseteq N[x]$ forms a clique by the assumption. Moreover, since $|C_x| \geq 2$ and $|C_y| \geq 2$, we know that $|C_x| + |C_y| \leq |C_x||C_y|$, implying that the number of internal edges in the constructed solution is at least as large as the number of internal edges of the optimal solution. ◀

Moreover, we prove the following generalization of Lemma 1.

▶ **Lemma 3.** *Let C and C' be two clusters of an optimal solution and let $x \in C$ and $y \in C'$. If y is C-compatible then x is not C'-compatible.*

Proof. Let S be an optimal solution such that $C, C' \in S$. Assume for contradiction that x is C'-compatible. We show that S is not optimal. Since y is C-compatible, we can move y to C and obtain a solution S_y that contains the clusters $C \cup \{y\}$ and $C' \setminus \{y\}$. Similarly, we construct a solution S_x from S, by moving x to C' so that $C \setminus \{x\}, C' \cup \{x\} \in S_x$. Notice that the S_x forms a clustering, since x is C'-compatible. We distinguish between the following cases, according to the values $|C|$ and $|C'|$.

- If $|C| \geq |C'|$ then $|S_y| > |S|$, because $\binom{|C|+1}{2} + \binom{|C'|-1}{2} > \binom{|C|}{2} + \binom{|C'|}{2}$.
- If $|C| < |C'|$ then $|S_x| > |S|$, because $\binom{|C|-1}{2} + \binom{|C'|+1}{2} > \binom{|C|}{2} + \binom{|C'|}{2}$.

In both cases we reach a contradiction to the optimality of S. Therefore, x is not C'-compatible. ◀

▶ **Corollary 4.** *Let C be a cluster of an optimal solution and let $x \in C$. If there is a vertex y that is C-compatible and $N[y] \subseteq N[x]$, then y belongs to C.*

3 Polynomial-time algorithm on interval graphs

Here we present a polynomial-time algorithm for the Cluster Deletion problem on interval graphs. A graph is an *interval graph* if there is a bijection between its vertices and a family of closed intervals of the real line such that two vertices are adjacent if and only if the two corresponding intervals intersect. Such a bijection is called an *interval representation* of the graph. We identify the intervals of the given representation with the vertices of the graph, interchanging these notions appropriately. Whether a given graph is an interval graph can be decided in linear time and if so, an interval representation can be generated in linear time [10, 13]. Notice that every induced subgraph of an interval graph is an interval graph.

Let G be an interval graph. Instead of working with the interval representation of G, we consider its sequence of maximal cliques. It is known that a graph G with p maximal cliques is an interval graph if and only if there is an ordering $K_1, \ldots, K_p$ of the maximal cliques of G, such that for each vertex v of G, the maximal cliques containing v appear consecutively in the ordering (see e.g., [10, 13]). A path $\mathcal{P} = K_1 \cdots K_p$ following such an ordering is called a *clique path* of G. Notice that a clique path is not necessarily unique for an interval graph. Also note that an interval graph with n vertices contains at most n maximal cliques. By definition, for every vertex v of G, the maximal cliques containing v form a connected subpath in $\mathcal{P}$.

Given a vertex v, we denote by $K_{a(v)}, \ldots, K_{b(v)}$ the maximal cliques containing v with respect to $\mathcal{P}$, where $K_{a(v)}$ and $K_{b(v)}$ are the *first* (*leftmost*) and *last* (*rightmost*) maximal cliques containing v. Notice that $a(v) \leq b(v)$ holds. Moreover, for every edge of G there is a maximal clique K_i of $\mathcal{P}$ that contains both endpoints of the edge. Thus, two vertices u and v are adjacent if and only if $a(v) \leq a(u) \leq b(v)$ or $a(v) \leq b(u) \leq b(v)$.

For a set of vertices $U \subseteq V$, we write $a\text{-}\min U$ and $a\text{-}\max U$ to denote the minimum and maximum value, respectively, among all $a(u)$ with $u \in U$. Similarly, $b\text{-}\min U$ and $b\text{-}\max U$ correspond to the minimum and maximum value, respectively, with respect to $b(u)$.

With respect to the Cluster Deletion problem, observe that for any cluster C of a solution, we know that $C \subseteq K_i$ where $K_i \in \mathcal{P}$, as C forms a clique. A vertex y is said to be *guarded* by two vertices x and z if $\min\{a(x), a(z)\} \leq a(y)$ and $b(y) \leq \max\{b(x), b(z)\}$ hold. For a clique C, observe that y is C-compatible if and only if there exists a maximal clique K_i such that $C \subseteq K_i$ with $a(y) \leq i \leq b(y)$. Observe that the following statement generalizes Corollary 4, in the sense that the neighborhood of the guarded vertex y is not necessarily contained in the neighborhood of x or z.

▶ **Lemma 5.** *Let x, y, z be three vertices of G such that y is guarded by x and z. If x and z belong to the same cluster C of an optimal solution and y is C-compatible then $y \in C$.*

Let $v_1, \ldots, v_n$ be an ordering of the vertices such that $b(v_1) \leq \cdots \leq b(v_n)$. For every v_i, v_j with $b(v_i) \leq b(v_j)$, we define the following set of vertices:

$$V_{i,j} = \{v \in V(G) : \min\{a(v_i), a(v_j)\} \leq a(v) \text{ and } b(v) \leq b(v_j)\}\,.$$

That is, $V_{i,j}$ contains all vertices that are guarded by v_i and v_j. We write $a(i,j)$ to denote the value of $\min\{a(v_i), a(v_j)\}$ and we simple write $K_{a(j)}$ and $K_{b(j)}$ instead of $K_{a(v_j)}$ and $K_{b(v_j)}$. Notice that for a neighbor u of v_j with $u \in V_{i,j}$, we have either $a(v_j) \leq a(u)$ or $a(v_i) \leq a(u) \leq a(v_j)$. This means that all neighbors of v_j that are totally included (i.e., all vertices u such that $a(v_j) \leq a(u) \leq b(u) \leq b(v_j)$) belong to $V_{i,j}$ for any v_i with $b(v_i) \leq b(v_j)$. To distinguish such neighbors of v_j, we define the following sets (see also Figure 1):

- $U(j)$ contains the neighbors $u \in V_{i,j}$ of v_j such that $a(u) < a(v_j) \leq b(u) \leq b(v_j)$ (neighbors of v_j in $V_{i,j}$ that partially overlap v_j).
- $M(j)$ contains the neighbors $w \in V_{i,j}$ of v_j such that $a(v_j) \leq a(w) \leq b(w) \leq b(v_j)$ (neighbors of v_j that are totally included within v_j).

In the forthcoming arguments, we restrict ourselves to the graph induced by $V_{i,j}$. It is clear that the first maximal clique that contains a vertex of $V_{i,j}$ is $K_{a(i,j)}$, whereas the last maximal clique is $K_{b(j)}$. For two vertices v_i, v_j with $b(v_i) \leq b(v_j)$, we define the following:

- $A_{i,j}$ is the value of an optimal solution for Cluster Deletion of the graph $G[V_{i,j}]$.

To ease the notation, when we say a cluster of $A_{i,j}$ we mean a cluster of an optimal solution of $G[V_{i,j}]$. Notice that $A_{1,n}$ is the desired value for the whole graph G, since $V_{1,n} = V(G)$.

Our task is to construct the values for $A_{i,j}$ by taking into account all possible clusters that contain v_j. To do so, we show that (i) the number of candidate clusters containing v_j in $A_{i,j}$ is polynomial and (ii) each such candidate cluster containing v_j separates the graph in a recursive way with respect to optimal subsolutions.

Observe that if $v_iv_j \in E(G)$ then $v_i \in U(j)$ if and only if $a(v_i) < a(v_j)$, whereas $v_i \in M(j)$ if and only if $a(v_j) \leq a(v_i)$; in the latter case, it is not difficult to see that $V_{i,j} = M(j) \cup \{v_j\}$. Thus, whenever $v_i \in M(j)$ holds, we have $V_{i,j} = V_{j,j}$. The candidates of a cluster of $A_{i,j}$ containing v_j lie among $U(j)$ and $M(j)$. Let us show with the next two lemmas that we can restrict ourselves into a polynomial number of such candidates. To avoid repeating ourselves, in the forthcoming statements we let v_i, v_j be two vertices with $b(v_i) \leq b(v_j)$.

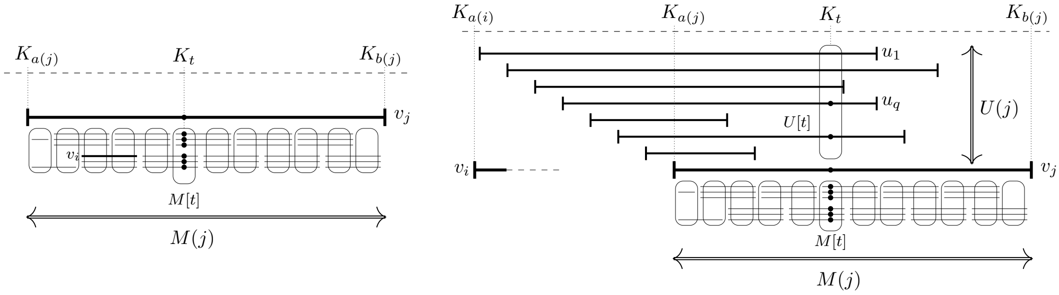


Figure 1 Illustrating the sets $M(j)$ and $U(j)$ for v_j. The left part shows the case in which $v_i \in M(j)$ (or, equivalently, $V_{i,j} = V_{j,j}$), whereas the right part corresponds to the case in which $a(v_i) < a(v_j)$.

▶ **Lemma 6.** *Let C be a cluster of $A_{i,j}$ containing v_j. If there is a vertex $w \in M(j)$ such that $w \in C$ then there is a maximal clique K_t with $a(v_j) \le t \le b(v_j)$ such that $K_t \cap M(j) \subseteq C$ and $C \cap M(j) \subseteq K_t$.*

Proof. Observe that $w \in M(j)$ implies $a(v_j) \le a(w) \le b(w) \le b(v_j)$. Since $v_j, w \in C$, we know that there is a maximal clique K_t for which $C \subseteq K_t$ with $a(v_j) \le a(w) \le t \le b(w) \le b(v_j)$. We show that all other vertices of $K_t \cap M(j)$ are guarded by v_j and w. Notice that for every vertex $y \in M(j)$ we already know that $a(v_j) \le a(y)$ and $b(y) \le b(v_j)$. Thus, for every vertex $y \in M(j)$ we have $a(v_j) = \min\{a(v_j), a(w)\} \le a(y)$ and $b(y) \le b(v_j) = \max\{b(v_j), b(w)\}$. This means that all vertices of $K_t \cap M(j) \setminus \{w\}$ are guarded by v_j and w. Moreover, since $C \subseteq K_t$, we know that all vertices of $K_t \cap M(j)$ are C-compatible. Therefore, we apply Lemma 5 to every vertex of $K_t \cap M(j)$, showing that $K_t \cap M(j) \subseteq C$. Furthermore, there is no vertex of $M(j) \setminus K_t$ that belongs to C, because $C \subseteq K_t$. ◀

By Lemma 6, we know that we have to pick the entire set $K_t \cap M(j)$ for constructing candidates to form a cluster that contains v_j and some vertices of $M(j)$. As there are at most n choices for K_t, we get a polynomial number of such candidate sets. We next show that we can construct a polynomial number of candidate sets that contain v_j and vertices of $U(j)$. For doing so, we consider the vertices of $U(j)$ increasingly ordered with respect to their first maximal clique. More precisely, let $U(j)_{\le a} = (u_1, \ldots, u_{|U(j)|})$ be an increasingly order of the vertices of $U(j)$ such that $a(u_1) \le \cdots \le a(u_{|U(j)|})$ (see the right part of Figure 1).

▶ **Lemma 7.** *Let C be a cluster of $A_{i,j}$ containing v_j and let $u_q \in U(j)_{\le a}$.If $u_q \in C$ then every vertex of $\{u_{q+1}, \ldots, u_{|U(j)|}\}$ that is C-compatible belongs to C.*

Proof. Let u be a vertex of $\{u_{q+1}, \ldots, u_{|U(j)|}\}$. We show that u is guarded by u_q and v_j. By the definition of $U(j)_{\le a}$, we know that $a(u_q) < a(u) < a(v_j)$. Moreover, observe that $b(u) \le b(v_j)$ holds by the fact that $u \in V_{i,j}$ and $b(u_q) \le b(v_j)$. Thus, we apply Lemma 5 to u, because $u_q, v_j \in C$ and u is C-compatible, showing that $u \in C$ as desired. ◀

For $a(v_j) \le t \le b(v_j)$, let $M[t] = K_t \cap M(j)$. Observe that each $M[t]$ may be an empty set. On the part $M(j)$, all vertices are grouped into the sets $M[a(v_j)], \ldots, M[b(v_j)]$. Similar to the group $M[t]$, let $U[t] = U(j) \cap K_t$. Then, all vertices of $U[t]$ are $\{v_j, M[t]\}$-compatible and all vertices of $M[t]$ are $\{v_j, U[t]\}$-compatible. Figure 1 depicts the corresponding sets.

▶ **Lemma 8.** *Let C be a cluster of $A_{i,j}$ containing v_j. Then, there is $a(v_j) \le t \le b(v_j)$ such that $M[t] \subseteq C$.*

All vertices of a cluster C containing v_j belong to $U(j) \cup M(j)$. Thus, $C \setminus \{v_j\}$ can be partitioned into $C \cap U(j)$ and $C \cap M(j)$. Also notice that $C \subseteq K_t$ for some $a(v_j) \leq t \leq b(v_j)$. Combined with the previous lemmas, we enumerate all such subsets C of $U(j) \cup M(j)$ in polynomial-time. In particular, we first build all candidates for $C \cap M(j)$, which are exactly the sets $M[t]$ by Lemmas 6 and 8. Then, for each of such candidate $M[t]$, we apply Lemma 7 to construct all subsets containing the last q vertices of $U[t]_{\leq a}$. Thus, there are at most n^2 candidate sets from the vertices of $U(j) \cup M(j)$ that belong to the same cluster with v_j.

3.1 Splitting into partial solutions

We further partition the vertices of $M(j)$. Given a pivot group $M[t]$, we consider the vertices that lie on the right part of $M[t]$. More formally, for $a(v_j) \leq t < b(v_j)$, we define the set $B_j(t) = \big((K_{t+1} \cup \cdots \cup K_{b(j)}) \setminus K_t\big) \cap M(j)$. The reason of breaking the vertices of the part $M(j)$ into sets $B_j(t)$ is the following.

▶ **Lemma 9.** *Let C be a cluster of $A_{i,j}$ such that $\{v_j\} \cup M[t] \subseteq C$, for $a(v_j) \leq t \leq b(v_j)$. Then, for any two vertices $x \in V_{i,j} \setminus B_j(t)$ and $y \in B_j(t)$, there is no cluster of $A_{i,j}$ that contains both of them.*

Proof. First observe that $y \in (M[t+1] \cup \cdots \cup M[b(j)]) \setminus M[t]$. We consider two cases for x, depending on whether $x \in M(j)$ or not. Assume that $x \in M(j)$. If $x \in M[t]$, then $x \in C$ by Lemma 6, which implies that $y \notin C$. If $x \in (M[a(v_j)] \cup \cdots \cup M[t-1]) \setminus M[t]$ then $xy \notin E(G)$. Now assume that $x \in U(j)$. If $x \in C$, then y does not belong to K_t, so that $y \notin C$. If $x \notin C$, then we show that x does not belong to a cluster with any vertex of $B_j(t)$. Assume for contradiction that x belongs to a cluster C' such that $C' \cap B_j(t) \neq \varnothing$. This means that $x \in K_{i'}$ with $t < i' \leq b(v_j)$ and $C' \subseteq K_{i'}$. Then v_j is C'-compatible and x is C-compatible, as both x and v_j belong to $K_t \cap K_{i'}$. Therefore, by Lemma 3 we reach a contradiction to x and v_j belonging to different clusters. ◀

For a non-empty set $S \subseteq V(G)$, we write $A(S)$ to denote the following solutions:

- $A(S) = A_{i',j'}$, where $v_{i'}$ is the vertex of S having the smallest $a(v_{i'})$ and $v_{j'}$ is the vertex of S having the largest $b(v_{j'})$.

Having this notation, observe that $A_{i,j} = A(V_{i,j})$, for any v_i, v_j with $b(v_i) \leq b(v_j)$. However, it is important to notice that $A(S)$ does not necessarily represent the optimal solution of $G[S]$, since the vertices of S may not be consecutive with respect to $V_{i',j'}$, so that S is only a subset of $V_{i',j'}$ in the corresponding solution $A_{i',j'}$ for $A(S)$. Under the following assumptions, with the next result we show that for the chosen sets we have $S = V_{i',j'}$.

▶ **Observation 10.** *Let $V_t = K_t \cap V_{i,j}$, for every $\min\{a(v_i), a(v_j)\} \leq t \leq b(v_j)$. If $S_L = \big(V_{a(i,j)} \cup \cdots \cup V_{t-1}\big) \setminus V_t$ then $S_L = V_{i',j'}$, where $i' = a\text{-}\min(S_L)$ and $j' = b\text{-}\max(S_L)$.*

Given the clique path $\mathcal{P} = K_1 \cdots K_p$, a *clique-index* t is an integer $1 \leq t \leq p$. Let $\ell(j), r(j)$ be two clique-indices such that $a(i,j) \leq \ell(j) \leq a(v_j)$ and $a(v_j) \leq r(j) \leq b(v_j)$. We denote by $\ell_r(j)$ the minimum value of $a(v)$ among all vertices of $v \in K_{r(j)} \cap V_{i,j}$ having $\ell(j) \leq a(v)$. Clearly, $\ell(j) \leq \ell_r(j) \leq r(j)$ holds. A pair of clique-indices $(\ell(j), r(j))$ is called *admissible pair* for a vertex v_j, if both $a(i,j) \leq \ell(j) \leq a(v_j)$ and $a(v_j) \leq r(j) \leq b(v_j)$ hold. Given an admissible pair $(\ell(j), r(j))$, we define the following set of vertices:

- $C(\ell(j), r(j)) = \{z \in V_{i,j} : \ell_r(j) \leq a(z) \text{ and } r(j) \leq b(z)\}$.

Observe that all vertices of $C(\ell(j), r(j))$ induce a clique in G, because $C(\ell(j), r(j)) \subseteq K_{r(j)}$. We say that a vertex u *crosses* the pair $(\ell(j), r(j))$ if $a(u) < \ell_r(j)$ and $r(j) \leq b(u)$. It is not difficult to see that for a vertex u that crosses $(\ell(j), r(j))$, we have $u \notin C(\ell(j), r(j))$. We prove the following properties of $C(\ell(j), r(j))$.

▶ **Lemma 11.** *Let $v_{i'}, v_{j'}$ be two vertices with $b(v_{i'}) \leq b(v_{j'})$ and let (ℓ, r) be an admissible pair for $v_{j'}$. Moreover, let v_i, v_j be the vertices of $V_{i',j'} \setminus C(\ell, r)$ having the smallest $a(v_i)$ and largest $b(v_j)$, respectively. If the vertices of $C(\ell, r)$ form a cluster in $A_{i',j'}$ then the following statements hold:*

1. *$V_{i,j} = V_{i',j'} \setminus C(\ell, r)$.*
2. *If $a(x) \leq r \leq b(x)$ holds for a vertex $x \in V_{i,j}$, then x crosses (ℓ, r).*
3. *Every vertex of $B_j(r)$ does not belong to the same cluster with any vertex of $V_{i,j} \setminus B_j(r)$.*
4. *Every vertex that crosses (ℓ, r) does not belong to the same cluster with any vertex $y \in V_{i,j}$ having $\ell_r \leq a(y)$.*

Notice that the number of admissible pairs $(\ell(j), r(j))$ for v_j is polynomial because there are at most n choices for each clique-index. Moreover, if $v_i \in M(j)$ then $\ell(j) = a(v_j)$. A pair of clique-indices (ℓ, r) with $\ell \leq r$ is called *bounding pair for v_j* if either $b(v_j) < r$ holds, or v_j crosses (ℓ, r). Given a bounding pair (ℓ, r) for v_j, we write $(\ell(j), r(j)) < (\ell, r)$ to denote the set of admissible pairs $(\ell(j), r(j))$ for v_j with the following restriction on $r(j)$:

- $r(j) \leq b(v_j)$, whenever $b(v_j) < r$ holds, and $r(j) < \ell$, whenever $b(v_j) \geq r$ holds.

Observe that if $b(v_j) < r$ holds, then $(\ell(j), r(j)) < (\ell, r)$ describes all admissible pairs for v_j with no restriction, regardless of ℓ. On the other hand, if $\ell < a(v_j)$ and $r \leq b(v_j)$ hold, then (ℓ, r) is not a bounding pair for v_j. In fact, we will show that the latter case will not be considered in our partial subsolutions. Intuitively, an admissible pair $(\ell(j), r(j))$ corresponds to the cluster containing v_j, whereas a bounding pair (ℓ, r) forbids v_j to select certain vertices as they have already formed a cluster that does not contain v_j (observe that $v_j \in C(\ell(j), r(j))$ and $v_j \notin C(\ell, r)$).

Our task is to construct subsolutions over all admissible pairs for v_j with the property that the vertices of $C(\ell(j), r(j))$ form a cluster. To do so, we consider a vertex $v_{j'}$ with $b(v_j) \leq b(v_{j'})$ and a cluster containing $v_{j'}$. Let (ℓ, r) be an admissible pair for $v_{j'}$ such that $a(v_j) \leq r \leq b(v_j)$. The previous results suggest to consider solutions in which the vertices of $C(\ell, r)$ form a cluster in an optimal solution. It is clear that if $\ell \leq a(v_j)$ then $v_j \in C(\ell, r)$. Moreover, if $b(v_j) < r$, then no vertex of $V_{i,j}$ belongs to $C(\ell, r)$. Thus, we need to construct solutions for $A_{i,j}$, whenever (ℓ, r) is a bounding pair for v_j and the vertices of $C(\ell, r)$ form a cluster. Such an idea is formally described as follows: Let (ℓ, r) be a bounding pair for v_j.

- $A_{i,j}[\ell, r]$ is the value of an optimal solution for Cluster Deletion of the graph $G[V_{i,j}] - (C(\ell, r) \cup B_j(r))$ such that the vertices of $C(\ell, r)$ form a cluster.

Hereafter, we assume that $B_j(t)$ with $t \geq b(v_j)$ corresponds to an empty set. Figure 2 illustrates a partition of the vertices with respect to $A_{i,j}[\ell, r]$. Notice that an optimal solution $A_{i,j}$ without any restriction is described in terms of $A_{i,j}[\ell, r]$ by $A_{i,j}[1, b(v_j) + 1]$, since no vertex of $V_{i,j}$ belongs to $C(1, b(v_j) + 1)$. Therefore, $A_{1,n}[1, n + 1]$ corresponds to the optimal solution of the whole graph G. As base cases, observe that if $V_{i,j}$ contains at most one vertex then $A_{i,j}[\ell, r] = 0$ for all bounding pairs (ℓ, r). For a set C, we write $|C|_2$ to denote the number $\binom{|C|}{2}$. With the following result, we describe a recursive formulation for the optimal solution $A_{i,j}[\ell, r]$, which is our central tool for our dynamic programming algorithm.

▶ **Lemma 12.** *Let (ℓ, r) be a bounding pair for v_j. Then,*

$$A_{i,j}[\ell, r] = \max_{(\ell(j), r(j)) < (\ell, r)} \left(A(V_L)[\ell(j), r(j)] + |C(\ell(j), r(j))|_2 + A(V_R)[\ell, r]\right),$$

where $V_L = V_{i,j} \setminus (C(\ell(j), r(j)) \cup B_j(r(j)))$ and $V_R = B_j(r(j)) \setminus (C(\ell, r) \cup B_j(r))$.

▶ **Theorem 13.** Cluster Deletion *is polynomial-time solvable on interval graphs.*

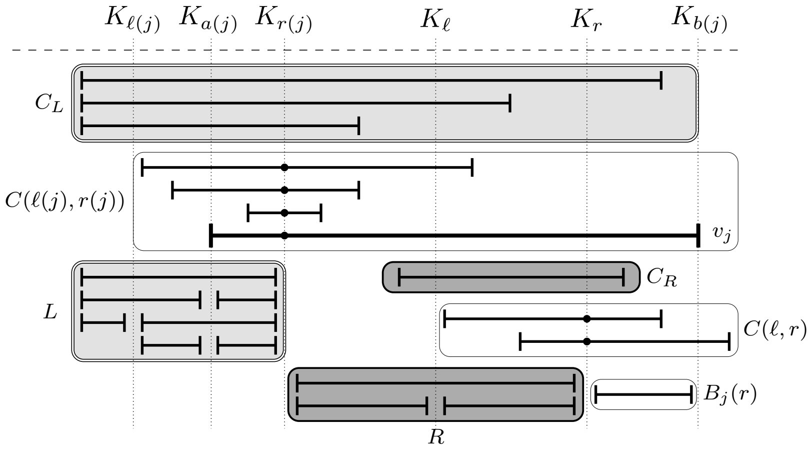


Figure 2 A partition of the set of vertices given in $A_{i,j}[\ell, r]$, where $V_L = C_L \cup L$ and $V_R = C_R \cup R$. Observe that $B_j(r(j)) = R \cup C_R \cup (C(\ell, r) \cap V_{i,j}) \cup B_j(r)$.

4 Cluster Deletion on a generalization of split graphs

A graph $G = (V, E)$ is a *split graph* if V can be partitioned into a clique C and an independent set I, where (C, I) is called a *split partition* of G. Split graphs are characterized as $(2K_2, C_4, C_5)$-free graphs [9]. They form a subclass of the larger and widely known graph class of *chordal graphs*, which are the graphs that do not contain induced cycles of length 4 or more as induced subgraphs. In general, a split graph can have more than one split partition and computing such a partition can be done in linear time [17].

Hereafter, for a split graph G, we denote by (C, I) a split partition of G in which C is a maximal clique. It is known that Cluster Deletion is polynomial-time solvable on split graphs [3]. In fact, the algorithm given in [3] is characterized by its simplicity due to the following elegant characterization of an optimal solution: if there is a vertex $v \in I$ such that $N(v) = C \setminus \{w\}$ and w has a neighbor v' in I then the non-trivial clusters of an optimal solution are $C \setminus \{w\} \cup \{v\}$ and $\{w, v'\}$; otherwise, the only non-trivial cluster of an optimal solution is C [3]. Here we study whether such a simple characterization can be extended into more general classes of split graphs. Due to Lemma 1, it is natural to consider true twins at the independent set, as they are grouped together in an optimal solution and they are expected not to influence the solution characterization. Surprisingly, we show that Cluster Deletion remains NP-complete even on such a slight generalization of split graphs. Before presenting our NP-completeness proof, let us first show that such graphs form a proper subclass of P_5-free chordal graphs. We start by giving the formal definition of such graphs.

▶ **Definition 14.** *A graph $G = (V, E)$ is called split-twin graph if its vertex set can be partitioned into C and I such that $G[C]$ is a clique and the vertices of each connected component of $G[I]$ form true twins in G.*

It is clear that in a split-twin graph G the following holds: (i) each connected component of $G[I]$ is a clique and forms a true-twin set in G, and (ii) contracting the connected components of $G[I]$ results in a split graph, denoted by G^*. Figure 3 illustrates the induced subgraphs that are forbidden in a split-twin graph.

▶ **Proposition 15.** *A graph G is split-twin if and only if it does not contain any of the graphs $C_4, C_5, P_5, 2P_3, \bar{A}, X$ as induced subgraphs.*

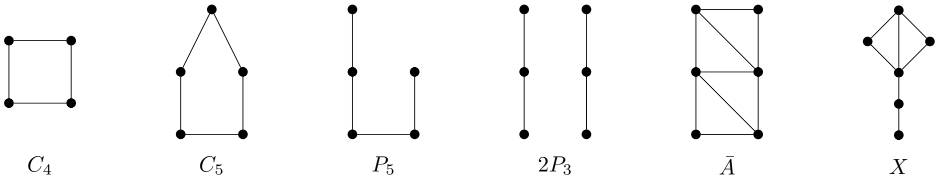


Figure 3 The list of forbidden induced subgraph characterization for split-twin graphs.

Thus by Proposition 15, split-twin graphs form a proper subclass of P_5-free chordal graphs, i.e., of (C_4, C_5, P_5)-free graphs. Now let us show that decision version of Cluster Deletion is NP-complete on split-twin graphs. This is achieved by observing that the constructed graphs given in the reduction for P_5-free graphs [3], constitute such split-related graphs. In particular, the reduction shown in [3] comes from the X3C problem: given a universe X of $3q$ elements and a collection $C = \{C_1, \ldots, C_{|C|}\}$ of 3-element subsets of X, asks whether there is a subset $C' \subseteq C$ such that every element of X occurs in exactly one member of C'. The constructed graph G is obtained by identifying the elements of X as a clique K_X and there are $|C|$ disjoint cliques $K_1, \ldots, K_{|C|}$ each of size $3q$ corresponding to the subsets of C and a vertex x of K_X is adjacent to all the vertices of K_i if and only if x belongs to the corresponding subset C_i of K_i. Then, it is not difficult to see that the vertices of each K_i are true twins and the contracted graph G^* is a split graph, showing that G is indeed a split-twin graph. Therefore, by the NP-completeness given in [3], we have:

▶ **Theorem 16.** Cluster Deletion *is NP-complete on split-twin graphs.*

However, here we give a different reduction that highlights an interesting connection between edge-weighted and vertex-weighted split graphs. In the Edge Weighted Cluster Deletion problem, each edge of the input graph is associated with a weight and the objective is to construct a clustered graph having the maximum total (cumulative) weight of edges. As already explained, we can contract true twins and obtain a vertex-weighted graph as input for the corresponding Cluster Deletion. Similarly, it is known that for edge-weighted graphs the corresponding Edge Weighted Cluster Deletion remains NP-hard even when restricted to particular variations on special families of graphs [3]. In fact, it is known that Edge Weighted Cluster Deletion remains NP-hard on split graphs even when (i) all edges inside the clique have weight one, (ii) all edges incident to a vertex $w \in I$ have the same weight q, and (iii) $q = |C|$ [3]. We abbreviate the latter problem by EWCD and denote by (C, I, k) an instance of the problem where (C, I) is a split partition of the vertices of G and k is the total weight of the edges in a cluster solution for G. With the following result, we show an interesting connection between the two variations of the problem when restricted to split-twin graphs.

▶ **Theorem 17.** *There exists a polynomial time algorithm that, given an instance (C, I, k) for EWCD, produces an equivalent instance for* Cluster Deletion *on split-twin graphs.*

Proof. From G, we build a split-twin graph $G' = (C' \cup I', E')$ by keeping the same clique $C' = C$, and for every vertex $w_j \in I$ we apply the following:

- We replace w_j by $q = |C|$ true twin vertices I'_j (i.e., by a q-clique) such that for any vertex $w' \in I'_j$ we have $N_{G'}(w') = N_G(w_j) \cup (I'_j \setminus \{w'\})$. That is, their neighbors outside I'_j are exactly $N_G(w_j)$. Moreover, the set of vertices $I'_1, \ldots, I'_{|I|}$ form I'.

By the above construction, it is not difficult to see that G' is a split-twin graph, since the graph induced by I' is a disjoint union of cliques and two adjacent vertices of I' are true twins in G'. Also observe that the construction takes polynomial time because q is at most $n = |V(G)|$. We claim that there is an edge weighted cluster solution for G with total weight at least k if and only if there is a cluster solution for G' having at least $k + |I| \cdot \binom{q}{2}$ edges.

Assume that there is a cluster solution S for G with total weight at least k. From S, we construct a solution S' for G'. There are three types of clusters in S:

(a) Cluster formed only by vertices of the clique C, i.e., $Y \in S$, where $Y \subseteq C$. We keep such clusters in S'. We denote by t_a the total weight of clusters of type (a). Notice that since the weight of edges having both endpoints in C are all equal to one, t_a corresponds to the number of edges in Y.

(b) Cluster formed only by one vertex $w_j \in I$, i.e., $\{w_j\} \in S$. In S' we replace such cluster by the corresponding clique I'_j having exactly $\binom{q}{2}$ edges. It is clear that the total weight of such clusters do not contribute to the value of S.

(c) Cluster formed by the vertices $y_1, \ldots, y_p, w_j$, where $y_i \in C$ and $w_j \in I$. As the weights of the edges between the vertices of y_i is one, the total number of weights in such a cluster is $\binom{p}{2} + p \cdot q$. Let t_c be the total weight of clusters of type (c). In S' we replace w_j by the vertices of I'_j and obtain a cluster S' having $\binom{p}{2} + p \cdot q + \binom{q}{2}$ number of edges.

Now observe that in S we have $t_a + t_c$ total weight, which implies $t_a + t_c \geq k$. Thus, in S' we have at least $t_a + t_c + |I| \cdot \binom{q}{2}$ edges, giving the desired bound.

For the opposite direction, assume that there is a solution S' in G' having at least $k + |I| \cdot \binom{q}{2}$ edges. All vertices of I'_j are true twins and, by Lemma 1, they belong to the same cluster in S'. Thus, any cluster of S' has one of the following forms: (i) Y', where $Y' \subseteq C'$, (ii) I'_j, (iii) $I'_j \cup \{y'_1, \ldots, y'_p\}$, where $y'_i \in C'$. This means that all internal edges having both endpoints in I' contribute to the value of S' by $|I| \cdot \binom{q}{2}$. Moreover, observe that for any internal edge of S' of the form $y'w'$ with $y' \in C'$ and $w' \in I'_j$, we know that there are exactly q internal edges incident to y' and the q vertices of I'_j. Thus, internal edges $y'w'$ of S' correspond to exactly one internal edge yw_j of S having weight q, where $y = y'$ (recall that $C = C'$) and w_j is the vertex of I associated with I_j. Hence, all internal edges outside each I'_j in S' correspond to either a weighted internal edge in S or to the same unweighted edge of C in S. Therefore, there is an edge weighted solution S having weight at least k. ◀

4.1 Polynomial-time algorithms on subclasses of split-twin graphs

Due to the hardness result given in Theorems 16 and 17, it is natural to consider subclasses of split-twin graphs related to their analogue subclasses of split graphs. We consider two such subclasses. The first one corresponds to the split-twin graphs such that the vertices of I have no common neighbor in the clique, unless they are true or false twins. The second subclass corresponds to threshold graphs (i.e., split graphs in which the vertices of the independent set have nested neighborhood) and form the split-twin graphs in which the vertices of I have a nested neighborhood. We formally define such graphs and give polynomial-time algorithms for Cluster Deletion. For a vertex $x \in I$ we write $N_C(x)$ to denote the set $N(x) \cap C$.

▶ **Definition 18.** *A split-twin graph G with partition (C, I) on its vertices is called 1-split-twin graph if for any two vertices $x, y \in I$, either $N_C(x) \cap N_C(y) = \varnothing$ or $N_C(x) = N_C(y)$.*

It is not difficult to see that in a 1-split-twin graph, any two vertices of I having a common neighbor in C have the same neighborhood in C. Close related to 1-split-twin graphs, are the *1-split graphs* which are the edge-weighted split graphs in which every vertex of the independent set is adjacent to exactly one vertex of the clique. It is known that the (edge-weighted) Cluster Deletion is solved in polynomial-time on 1-split graphs [3]. Let us explain how to use the algorithm on a 1-split graph to obtain a polynomial-time algorithm on a 1-split-twin graph $G = (C, I)$. Observe that the contracted graph G^* is 1-split. Let xy be an edge of G^*. Denote by $w(x)$ and $w(y)$ the weights assigned to x and y that correspond to the sizes of their true twins classes in G. From the vertex-weighted 1-split graph G^*, construct an edge-weighted 1-split graph H^* by removing the vertex weights and for each edge xy assign weight $w(x) \cdot w(y)$. Then, given a solution of the edge-weighted H^* taken from the algorithm of [3], we obtain a solution for Cluster Deletion on G by adding the internal edges corresponding to each contracted vertex.

Notice, however, that the running time is bounded by the polynomial-time algorithm of 1-split graphs. In particular the described algorithm of 1-split graphs is accomplished through a minimizer of a general submodular function provided a given oracle for evaluating the function value [3]. This means that through such an approach it is unlikely to achieve a better running time for 1-split-twin graphs, unless there is a faster algorithm with less number of oracle calls for finding a minimizer of a general submodular function. With our next result we provide a simpler and faster (linear-time) algorithm for Cluster Deletion on 1-split-twin graphs that avoids the usage of submodular functions minimization.

▶ **Theorem 19.** Cluster Deletion *is linear-time solvable on 1-split-twin graphs.*

Proof. Let G be a 1-split-twin graph with partition (C, I). First observe that if G is disconnected then I contains isolated cliques, i.e., true twins having no neighbor in C. Thus we can restrict ourselves to a connected graph G, since by Lemma 1 each isolated clique is contained in exactly one cluster of an optimal solution. We now show that all vertices of C that have a common neighbor in I are true twins. Let u and v be two vertices of C such that $x \in N(u) \cap N(v) \cap I$. All vertices of $C \setminus \{u, v\}$ are adjacent to both u and v. Assume that there is a vertex $y \in I$ that is adjacent to u and non-adjacent to v. If $xy \in E(G)$ then by the definition of split-twin graphs x and y are true twins which contradicts the assumption of $xv \in E(G)$ and $yv \notin E(G)$. Otherwise, x and y are non-adjacent and since $N_C(x) \cap N_C(y) \neq \varnothing$ we reach a contradiction to the definition of 1-split-twin graphs. Thus, all vertices of C that have a common neighbor in I are true twins.

We partition the vertices of C into true twin classes $C_1, \ldots, C_k$, such that each C_i contains true twins of C. From the previous discussion, we know that any vertex of I is adjacent to all the vertices of exactly one class C_i; otherwise, there are vertices of different classes in C that have common neighbor. For a class C_i, we partition the vertices of $N(C_i) \cap I$ into true twin classes $I_i^1, \ldots, I_i^q$ such that $|I_i^1| \geq \cdots \geq |I_i^q|$.

We claim that in an optimal solution S, the vertices of each class I_i^j with $j \geq 2$ constitute a cluster. To see this, observe first that the vertices of I_i^j, $1 \leq j \leq q$, are true twins, and by Lemma 1 they all belong to the same cluster of S. Also, by Lemma 1 we know that all the vertices of C_i belong to the same cluster of S. Moreover, all vertices between different classes I_i^j,$I_i^{j'}$ are non-adjacent and are C_i-compatible. Since every vertex of I_i^j is non-adjacent to all the vertices of $V(G) \setminus \{I_i^j \cup C_i\}$, we know that any cluster of S that contains I_i^j is of the form either $\{I_i^j \cup C_i\}$ or I_i^j. Assume that there is a cluster that contains $\{I_i^j \cup C_i\}$ with $j \geq 2$. Then, we substitute the vertices of I_i^j by the vertices of I_i^1 and obtain a solution of at least the same size, because $|I_i^1| \geq |I_i^j|$ implies $\binom{|C_i|+|I_i^1|}{2} \geq \binom{|C_i|+|I_i^j|}{2}$. Thus, all vertices of each class I_i^j with $j \geq 2$ constitute a cluster in an optimal solution S.

This means that we can safely remove the vertices of I_i^j with $j \geq 2$, by constructing a cluster that contains only I_i^j. Hence, we construct a graph G^* from G, in which there are only matched pair of k classes (C_i, I_i) such that (i) all sets C_i, I_i are non-empty except possibly the set I_k, (ii) $N(C_i) \cap I = I_i$, (iii) $N(I_i) = C_i$, (iv) $G^*[C_i \cup I_i]$ is a clique, and (v) $G^*[C_1 \cup \cdots \cup C_k]$ is a clique. Our task is to solve Cluster Deletion on G^*, since for the rest of the vertices we have determined their cluster. By Lemma 1, if the vertices of $C_i \cup C_j$ belong to the same cluster then the vertices of each I_i and I_j constitute two clusters. Thus, for each set of vertices I_i we know that either one of $C_i \cup I_i$ or I_i constitutes a cluster in S. This boils down to compute a set M of matched pairs (C_i, I_i), having the maximum value

$$\sum_{(C_i,I_i)\in M} \binom{|C_i|+|I_i|}{2} + \binom{\sum_{C_j \notin M} |C_j|}{2} + \sum_{I_j \notin M} \binom{|I_j|}{2}.$$

Let (C_i, I_i) and (C_j, I_j) be two pairs of classes such that $|C_i| + |I_i| \leq |C_j| + |I_j|$. We show that if $(C_j, I_j) \notin M$ then $(C_i, I_i) \notin M$. Assume for contradiction that $(C_j, I_j) \notin M$ and $(C_i, I_i) \in M$. Observe that $|I_j| < \sum_{C_t \notin M \setminus C_j} |C_t|$, because I_j is C_j-compatible. Similarly, we know that $\sum_{C_t \notin M \setminus C_j} |C_t| + |C_j| \leq |I_i|$. This however, shows that $|C_j| + |I_j| < |I_i|$, contradicting the fact that $|C_i| + |I_i| \leq |C_j| + |I_j|$. Thus $(C_j, I_j) \notin M$ implies $(C_i, I_i) \notin M$.

This means that we can consider the k pair of classes (C_i, I_i) in a decreasing order according to their number of vertices $|C_i| + |I_i|$. With a simple dynamic programming algorithm, starting from the largest ordered pair (C_1, I_1) we know that either (C_1, I_1) belongs to M or not. In the former, we add $\binom{|C_1|+|I_1|}{2}$ to the optimal value of $(C_2, I_2), \ldots, (C_k, I_k)$ and in the latter we know that no pair belongs to M giving a total value of $\binom{\sum |C_i|}{2} + \sum \binom{|I_i|}{2}$. By choosing the maximum between the two values, we construct a table of size k needed for the dynamic programming. Computing the twin classes and the partition (C, I) takes linear time in the size of G and sorting the pair of classes can be done $O(n)$ time, since $\sum(|C_i| + |I_i|)$ is bounded by n. Thus, the total running time is $O(n + m)$, as the dynamic programming for computing M requires $O(n)$ time. Therefore, all steps can be carried out in linear time for a 1-split-twin graph G. ◀

▶ **Definition 20.** *A split-twin graph G with partition (C, I) on its vertices is called threshold-twin graph if the vertices of I can be ordered $w_1, \ldots, w_{|I|}$ such that for any $w_i, w_j \in I$ with $i < j$, we have $N_C(w_i) \subseteq N_C(w_j)$.*

For the next result, we prove that there is no P_4 in a threshold-twin graph (P_4-free graphs are closed under true twins addition). Thus, by the algorithm given in [11], we have:

▶ **Theorem 21.** Cluster Deletion *is polynomial-time solvable on threshold-twin graphs.*

5 Concluding remarks

It is notable that our algorithm for interval graphs, heavily relies on the linear structure obtained from their clique paths. Such an observation, leads us to consider few open questions regarding two main directions. On the one hand, it seems tempting to adjust our algorithm for other vertex partitioning problems on interval graphs within a more general framework, as already have been studied for particular graph properties [4, 12, 20, 21, 25]. On the other hand, it is reasonable to ask whether our approach works for Cluster Deletion on graphs admitting similar linear structure such as permutation graphs, or graphs having bounded linear related parameter. Towards the latter direction, observe that Cluster Deletion as a vertex partitioning problem can be solved in linear time on graphs of bounded treewidth by using Courcelle's machinery [6].

Although for other structural parameters it seems rather difficult to obtain similar result, it is still interesting to settle the complexity of Cluster Deletion on distance hereditary graphs that admit constant clique-width [15]. In fact, we would like to settle the case in which from a given cograph we can append degree-one vertices. This comes in conjunction with the 1-split-twin graphs, as they can be seen as a degree-one extension of a clique.

References

1 N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004.

2 F. Bonomo, G. Durán, A. Napoli, and M. Valencia-Pabon. A one-to-one correspondence between potential solutions of the cluster deletion problem and the minimum sum coloring problem, and its application to P_4-sparse graphs. *Inf. Proc. Lett.*, 115:600–603, 2015.

3 F. Bonomo, G. Durán, and M. Valencia-Pabon. Complexity of the cluster deletion problem on subclasses of chordal graphs. *Theor. Comp. Science*, 600:59–69, 2015.

4 B. Bui-Xuan, J. A. Telle, and M. Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theor. Comput. Sci.*, 511:66–76, 2013.

5 M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. In *Proceedings of FOCS 2003*, pages 524–533, 2003.

6 B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

7 A. Dessmark, J. Jansson, A. Lingas, E.-M. Lundell, and M. Persson. On the approximability of maximum and minimum edge clique partition problems. *Int. J. Found. Comput. Sci.*, 18:217–226, 2007.

8 R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2012.

9 S. Földes and P. L. Hammer. Split graphs. *Congressus Numerantium*, 19:311–315, 1977.

10 D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965.

11 Y. Gao, D. R. Hare, and J. Nastos. The cluster deletion problem for cographs. *Discrete Mathematics*, 313:2763–2771, 2013.

12 M. U. Gerber and D. Kobler. Algorithms for vertex-partitioning problems on graphs with fixed clique-width. *Theor. Comp. Science*, 299:719–734, 2003.

13 P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539–548, 1964.

14 P. A. Golovach, P. Heggernes, A. L. Konstantinidis, P. T. Lima, and C. Papadopoulos. Parameterized aspects of strong subgraph closure. In *Proceedings of SWAT 2018*, pages 23:1–23:13, 2018.

15 M. C. Golumbic and U. Rotics. On the clique-width of some perfect graph classes. *Int. J. Found. Comput. Sci.*, 11:423–443, 2000.

16 N. Grüttemeier and C. Komusiewicz. On the relation of strong triadic closure and cluster deletion. In *Proceedings of WG 2018*, pages 239–251, 2018.

17 P. L. Hammer and B. Simeone. The splittance of a graph. *Combinatorica*, 1:275–284, 1981.

18 P. Hansen and B. Jaumard. Cluster analysis and mathematical programming. *Math. Programming*, 79:191–215, 1997.

19 J. Hartigan. *Clustering Algorithms*. Wiley, New York, 1975.

20 P. Heggernes, D. Lokshtanov, J. Nederlof, C. Paul, and J. A. Telle. Generalized graph clustering: recognizing (p, q)-cluster graphs. In *Proceedings of WG 2010*, pages 171–183, 2010.

21 I. A. Kanj, C. Komusiewicz, M. Sorge, and E. Jan van Leeuwen. Solving partition problems almost always requires pushing many vertices around. In *Proceedings of ESA 2018*, pages 51:1–51:14, 2018.

22 C. Komusiewicz and J. Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160:2259–2270, 2012.

23 S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

24 R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144:173–182, 2004.

25 J. A. Telle and A. Proskurowski. Algorithms for Vertex Partitioning Problems on Partial k-Trees. *SIAM J. Discrete Math.*, 10:529–550, 1997.

Constrained Representations of Map Graphs and Half-Squares

Hoang-Oanh Le
Berlin, Germany
LeHoangOanh@web.de

Van Bang Le
Universität Rostock, Institut für Informatik, Rostock, Germany
van-bang.le@uni-rostock.de

Abstract

The square of a graph H, denoted H^2, is obtained from H by adding new edges between two distinct vertices whenever their distance in H is two. The half-squares of a bipartite graph $B = (X, Y, E_B)$ are the subgraphs of B^2 induced by the color classes X and Y, $B^2[X]$ and $B^2[Y]$. For a given graph $G = (V, E_G)$, if $G = B^2[V]$ for some bipartite graph $B = (V, W, E_B)$, then B is a representation of G and W is the set of points in B. If in addition B is planar, then G is also called a map graph and B is a witness of G [Chen, Grigni, Papadimitriou. Map graphs. *J. ACM*, 49 (2) (2002) 127-138].

While Chen, Grigni, Papadimitriou proved that any map graph $G = (V, E_G)$ has a witness with at most $3|V| - 6$ points, we show that, given a map graph G and an integer k, deciding if G admits a witness with at most k points is NP-complete. As a by-product, we obtain NP-completeness of EDGE CLIQUE PARTITION on planar graphs; until this present paper, the complexity status of EDGE CLIQUE PARTITION for planar graphs was previously unknown.

We also consider half-squares of tree-convex bipartite graphs and prove the following complexity dichotomy: Given a graph $G = (V, E_G)$ and an integer k, deciding if $G = B^2[V]$ for some tree-convex bipartite graph $B = (V, W, E_B)$ with $|W| \leq k$ points is NP-complete if G is non-chordal dually chordal and solvable in linear time otherwise. Our proof relies on a characterization of half-squares of tree-convex bipartite graphs, saying that these are precisely the chordal and dually chordal graphs.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases map graph, half-square, edge clique cover, edge clique partition, graph classes

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.13

1 Introduction

Map graphs, introduced and investigated in [5, 6], are intersection graphs of simply-connected and interior-disjoint regions of the Euclidean plane; each region is homeomorphic to a closed disc. More precisely, a *map* of a graph $G = (V, E_G)$ is a function $\mathcal{M}$ taking each vertex $v \in V$ to a region $\mathcal{M}(v)$ in the plane, such that all $\mathcal{M}(v)$, $v \in V$, are interior-disjoint, and two distinct vertices v and v' of G are adjacent if and only if the boundaries of $\mathcal{M}(v)$ and $\mathcal{M}(v')$ intersect, even in a point. A *map graph* is one having a map. Map graphs are interesting as they generalize planar graphs in a very natural way. Some applications of map graphs have been addressed in [8]. Papers dealing with hard problems in map graphs include [4, 9, 10, 11, 13, 14].

In [5, 6], the notion of *half-squares* of bipartite graphs has been also introduced in order to give a combinatorial representation of map graphs. The square of a graph H, denoted H^2, is obtained from H by adding new edges between any two vertices at distance two in H. For a bipartite graph $B = (X, Y, E_B)$, the subgraphs of the square B^2 induced by the color classes X and Y, $B^2[X]$ and $B^2[Y]$, are called the two *half-squares* of B. For a given graph $G = (V, E_G)$, if $G = B^2[V]$ for some bipartite graph $B = (V, W, E_B)$, then B is a *representation* or a *half-root* of G and W is the set of *points* in B. While every graph is a

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 13; pp. 13:1–13:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

half-square of some bipartite graph, it turns out that map graphs are exactly half-squares of *planar* bipartite graphs [5, 6]. If $G = (V, E_G)$ is a map graph and $B = (V, W, E_B)$ is a planar representation of G, then B is called a *witness* of G. See Figure 1 for an example.

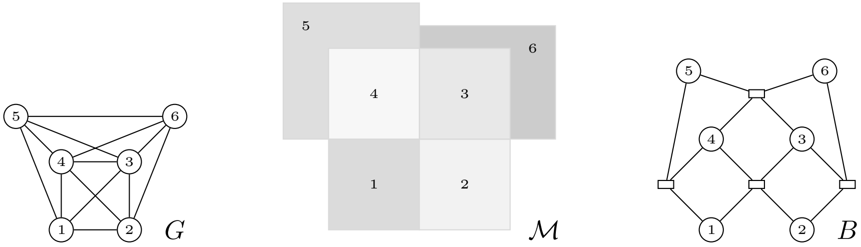


Figure 1 A map graph G, a map $\mathcal{M}$, and a witness B of G.

It is perhaps important to note at this place that one of the difficulties in recognizing map graphs is that we do not know the set of points of a witness we are looking for. It is shown in [5, 6] that an n-vertex graph $G = (V, E_G)$ is a map graph if and only if it has a witness $B = (V, W, E_B)$ with $|W| \leq 3n - 6$ points, implying that recognizing map graphs is in NP. Subsequently, Thorup [28] announced that recognizing map graphs is in P by giving an $\Omega(n^{120})$-time algorithm for n-vertex input graphs.[1] Thorup's algorithm is very complex and highly non-combinatorial. Given the very high polynomial degree in Thorup's running time, the most discussed problem concerning map graphs is whether there is a faster recognition algorithm with simpler arguments for map graphs.

One direction in attacking this problem is to consider map graphs with restricted witness. Structural results and more efficient recognition algorithms for map graphs with restricted witness will enhance our understanding on map graphs in whole. More generally, let $\mathcal{B}$ be a class of (not necessarily planar) bipartite graphs, the following problem has been discussed first by Le and Le [18].

$\mathcal{B}$ REPRESENTATION
Instance: A graph $G = (V, E_G)$.
Question: Does there exist a bipartite graph $B = (V, W, E_B)$ in $\mathcal{B}$ with $G = B^2[V]$?

Recall that in case $\mathcal{B}$ is the class of all planar bipartite graphs, PLANAR REPRESENTATION is the problem of recognizing map graphs, which admits an $\Omega(n^{120})$-time algorithm due to Thorup. Recall also that every map graph has a witness $B = (V, W, E_B)$ with $|W| \leq 3|V| - 6$ due to Chen *et al.* [5, 6]. This motivates considering the following problem.

POINT MINIMAL $\mathcal{B}$ REPRESENTATION
Instance: A graph $G = (V, E_G)$ and an integer k.
Question: Does there exist a bipartite graph $B = (V, W, E_B)$ in $\mathcal{B}$ with $G = B^2[V]$ and $|W| \leq k$?

In case $\mathcal{B}$ is the class of all bipartite graphs, we simply denote the problem by POINT MINIMAL REPRESENTATION.

This paper considers the cases where $\mathcal{B}$ is one of the classes of (planar) bipartite graphs of a given girth, of tree-convex bipartite graphs, and of tree-biconvex bipartite graphs. All terms used are given in the next section.

[1] Thorup did not give the running time explicitly, but it is estimated to be roughly $\Omega(n^{120})$ with n being the vertex number of the input graph; cf. [6].

Our contributions. We first consider map graphs with witness of large girth and, more generally, half-squares of bipartite graphs of large girth, and prove the following complexity dichotomy for MINIMAL POINT (PLANAR) GIRTH-g REPRESENTATION: Given a (map) graph $G = (V, E_G)$ and an integer k, deciding if $G = B^2[V]$ for some (planar) bipartite graph $B = (V, W, E_B)$ of girth at least g with $|W| \leq k$ points is NP-complete for $g \leq 6$ and polynomially solvable otherwise. The case $g \geq 8$ is based on our previous paper [19], and the case $g \leq 6$ is based on a close connection to the well-known NP-complete problems EDGE CLIQUE COVER and EDGE CLIQUE PARTITION. It is perhaps interesting to note that, while recognizing map graphs is in P due to Thorup, our hardness result in case $g = 4$ says that the problem becomes intractable if we ask for a witness with few points. In case $g = 6$, our result implies that EDGE CLIQUE PARTITION is NP-complete for planar graphs. (The complexity status of this problem for planar graphs was previously unknown.) We then consider half-squares of tree-convex bipartite graphs, and prove the following complexity dichotomy for MINIMAL POINT TREE-CONVEX REPRESENTATION: Given a graph $G = (V, E_G)$ and an integer k, deciding if $G = B^2[V]$ for some tree-convex bipartite graph $B = (V, W, E_B)$ with $|W| \leq k$ points is NP-complete for non-chordal dually chordal graphs G and solvable in linear time otherwise. We obtain this result by proving that half-squares of tree-convex bipartite graphs are exactly the chordal and dually chordal graphs. We also show that MINIMAL POINT TREE-BICONVEX REPRESENTATION can be solved in linear time by proving that half-squares of tree-biconvex bipartite graphs are precisely the double chordal graphs. Our results on half-squares of tree-(bi)convex bipartite graphs settle the question left open in [18].

Related work. The first restricted representations of map graphs have been considered in [5, 6] which lead to the so-called k-map graphs; k-map graphs are map graphs having a witness in which every point has at most k neighbors. It turns out that, for $k \leq 3$, k-map graphs are precisely the planar graphs. 4-map graphs can be recognized in cubic time [1], and are related to 1-planar graphs [1, 7], a relevant topic in graph drawing. Recognizing k-map graphs, $k \geq 5$, in polynomial time still remains open. (We remark that Thorup's algorithm cannot be used to recognize map graphs having witness with additional properties.)

Mnich et al. [25] considered map graphs with outerplanar witness and tree witness, and showed that such map graphs can be recognized in linear time. Map graphs with witness of a given girth and, more generally, half-squares of bipartite graphs of a given girth have been considered in the recent paper [19]. It is shown in that paper that half-squares of (planar) bipartite graphs of girth at least 8 admit good characterizations, leading to cubic time recognition algorithms. In [18], half-squares of classical bipartite graphs, such as biconvex, convex, and chordal bipartite graphs, have been studied. It turns out that half-squares of biconvex, convex, and chordal bipartite graphs (all are subclasses of tree-(bi)convex bipartite graphs) are exactly the proper interval, interval, and strongly chordal graphs, respectively.

The paper is organized as follows. All definitions and notion needed are provided in the next section. Section 3 first collects known results on half-squares of (planar) bipartite graphs of large girth, and then provides a dichotomy theorem for POINT MINIMAL GIRTH-g (PLANAR) REPRESENTATION. Section 4 deals with half-squares of tree-convex and tree-biconvex bipartite graphs, and provides a dichotomy theorem for POINT MINIMAL TREE-CONVEX REPRESENTATION. Section 5 concludes the paper with some open problems for future work.

2 Preliminaries

All graphs considered are simple and connected. Let $G = (V, E_G)$ be a graph with vertex set $V(G) = V$ and edge set $E(G) = E_G$. A *stable set* (a *clique*) in G is a set of pairwise non-adjacent (adjacent) vertices. The complete graph on n vertices and the cycle with n vertices are denoted K_n and C_n, respectively. A K_3 is also called a *triangle*. The *diamond*, denoted $K_4 - e$, is the graph obtained from the K_4 by deleting an edge.

The neighborhood of a vertex v in G, denoted $N_G(v)$, is the set of all vertices in G adjacent to v; if the context is clear, we simply write $N(v)$. A *universal vertex* v in G is one with $N(v) = V \setminus \{v\}$, i.e., v is adjacent to all other vertices in G.

Let F be a graph. *F-free graphs* are those having no induced subgraphs isomorphic to F. *Chordal graphs* are precisely the C_k-free graphs, $k \geq 4$. A *dually chordal graph* G is one in which every connected component H of G admits a spanning tree T such that every maximal clique of H induces a subtree in T.[2] Graphs that are both chordal and dually chordal are called *double chordal*. While chordal graphs are closed under taking induced subgraphs, dually chordal graphs and double chordal graphs are not. *Strongly chordal graphs* are those graphs G such that every induced subgraph of G is double chordal. See [15, 24, 27] for more information on these graph classes. Additional information on dually chordal graphs can be found in [2]. We will use the well-known facts that chordal and dually chordal graphs, hence double chordal graphs, can be recognized in linear time [15, 27, 2], and that any n-vertex chordal graph has at most n maximal cliques and all of them can be listed in linear time [15, 27].

For a subset $W \subseteq V$, $G[W]$ is the subgraph of G induced by W, and $G - W$ stands for $G[V \setminus W]$. For a vertex v, $G - v$ stands for $G - \{v\}$. We will consider map graphs with large-girth witness and, more generally, half-squares of bipartite graphs of large girth. Here, the *girth* of a graph is the minimum length of a cycle in that graph. (Thus, a graph has girth at least g if and only if it is C_k-free for all $k < g$.) We will also consider half-squares of tree-convex bipartite graphs, a problem left open in [18]. A bipartite graph $B = (X, Y, E_B)$ is *tree-convex on X* if there exists a tree $T = (X, E_T)$ such that, for each $y \in Y$, $N(y)$ induces a subtree in T. Being *tree-convex on Y* is defined similarly. B is *tree-convex* if it is tree-convex on X or tree-convex on Y. B is *tree-biconvex* if it is both tree-convex on X and tree-convex on Y. A well-known subclass of tree-biconvex bipartite graphs consists of the *chordal bipartite graphs*, i.e., bipartite graphs containing no induced cycle of length at least six. Liu [21] discusses relationships between tree-convex bipartite graphs and other classical classes of bipartite graphs.

POINT MINIMAL $\mathcal{B}$ REPRESENTATION is related to two well-studied problems. An *edge clique cover* of a graph G is a family of cliques $\mathcal{C}$ in G such that every edge of G is contained in one or more cliques in $\mathcal{C}$. An *edge clique partition* of G is an edge clique cover $\mathcal{C}$ of G such that every edge of G is contained in exactly one clique in $\mathcal{C}$. The two well-studied problems are:

EDGE CLIQUE COVER

Instance: A graph $G = (V, E_G)$ and an integer k.

Question: Does G have an edge clique cover of size k or less?

[2] Dually chordal graphs haven been studied under different names and admit various characterizations. The chosen definition is dependent on our purpose.

EDGE CLIQUE PARTITION
Instance: A graph $G = (V, E_G)$ and an integer k.
Question: Does G have an edge clique partition of size k or less?

EDGE CLIQUE COVER is NP-complete [26, 17], and remains NP-complete on planar graphs [3] and on complements of bipartite graphs [20]. We will use the fact that EDGE CLIQUE COVER is solvable in linear time on chordal graphs [23]. The monograph [24] provides more information on edge clique covers.

EDGE CLIQUE PARTITION is NP-complete [26, 16], and remains NP-complete on K_4-free graphs [22]. In contrast to EDGE CLIQUE COVER, EDGE CLIQUE PARTITION is NP-complete on chordal graphs, even on split graphs [30]. EDGE CLIQUE PARTITION on planar graphs has been considered in [12]; the complexity status of this problem on planar graphs was unknown until our present work.

Let $\mathcal{C}$ be an edge clique cover of a graph $G = (V, E_G)$. The *vertex-$\mathcal{C}$ incidence bipartite graph* of G is $B_{\mathcal{C}} = (V, \mathcal{C}, E_B)$ with $E_B = \{vC \mid v \in V, C \in \mathcal{C}, v \in C\}$. If $\mathcal{C} = \mathcal{C}(G)$, the set of all maximal cliques in G, then $B_{\mathcal{C}}$ is called the *vertex-clique incidence bipartite graph* of G and usually denoted by B_G.

▶ **Fact 1.**
(i) *For any graph $G = (V, E_G)$ and any edge clique cover $\mathcal{C}$ of G, $G = B^2_{\mathcal{C}}[V]$. If $\mathcal{C}$ is in addition an edge clique partition, then $B_{\mathcal{C}}$ is C_4-free.*
(ii) *For any graph $G = (V, E_G)$ and any bipartite graph $B = (V, W, E_B)$ with $G = B^2[V]$, $\mathcal{C} = \{N_B(w) \mid w \in W\}$ is an edge clique cover of G. If B is in addition C_4-free, then $\mathcal{C}$ is an edge clique partition of G.*

Proof. (i): For all vertices $x, y \in V$ we have $xy \in E_G \Leftrightarrow x, y \in C$ for some $C \in \mathcal{C} \Leftrightarrow xC$ and yC are edges of $B_{\mathcal{C}}$ for some $C \in \mathcal{C} \Leftrightarrow xy \in E(B^2_{\mathcal{C}}[V])$. If $\mathcal{C}$ is an edge clique partition, then, for any two distinct cliques $C, C' \in \mathcal{C}$, C and C' have at most one vertex in common. Hence, $B_{\mathcal{C}}$ is a C_4-free.

(ii): If $G = B^2[V]$ for some bipartite graph $B = (V, W, E_B)$, then any edge of G is in a clique $N_B(w)$ for some $w \in W$, hence $\{N_B(w) \mid w \in W\}$ is an edge clique cover of G. If, in addition, B is C_4-free, then $|N_B(w) \cap N_B(w')| \leq 1$ for any two distinct points $w, w' \in W$. Hence $\{N_B(w) \mid w \in W\}$ is an edge clique partition of G. ◀

Thus,

- POINT MINIMAL REPRESENTATION and EDGE CLIQUE COVER, and
- POINT MINIMAL C_4-FREE REPRESENTATION and EDGE CLIQUE PARTITION

are computationally equivalent.

3 Girth-constrained representations

This section deals with half-squares of (planar) bipartite graphs of large girth. In [18], the following useful fact has been observed, and used in [19] in discussing half-squares of bipartite graphs with girth constraints.

▶ **Lemma 1** ([18]). *Let $G = B^2[V]$ for some bipartite graph $B = (V, W, E_B)$. If B has no induced cycle of length six, then every maximal clique Q in G stems from a star in B, i.e., there is a point $w \in W$ such that $Q = N_B(w)$.*

We will also use this fact when considering point minimal representations of half-squares and map graphs.

3.1 Half-squares of girth-constrained bipartite graphs

Recall that *every* graph is a half-square of a girth-six bipartite graph (take its subdivision). Half-squares of bipartite graphs of large girth have been fully characterized as follows.

▶ **Theorem 2** ([19]). *Let $t \geq 4$ be an integer. The following statements are equivalent for every graph $G = (V, E_G)$.*

(i) *G is a half-square of a bipartite graph with girth at least $2t$;*
(ii) *G is diamond-free and C_ℓ-free for every $4 \leq \ell \leq t-1$;*
(iii) *The vertex-clique incidence bipartite graph B_G of G has girth at least $2t$.*

Theorem 2 implies that half-squares of bipartite graphs of large girth can be recognized in cubic time (cf. [19]).

By definition, *every* map graph is a half-square of a planar bipartite graph. Though map graphs can be recognized in polynomial time due to Thorup, no good characterization for map graphs is known so far. Map graphs of planar bipartite graphs of large girth have been fully characterized as follows.

▶ **Theorem 3** ([19]). *Let $t \geq 4$ be an integer. The following statements are equivalent for every graph $G = (V, E_G)$.*

(i) *G is a map graph having a witness of girth at least $2t$;*
(ii) *G is diamond-free and C_ℓ-free for every $4 \leq \ell \leq t-1$, and the vertex-clique incidence bipartite graph B_G of G is planar;*
(iii) *The vertex-clique incidence bipartite graph B_G of G is planar and has girth at least $2t$.*

Theorem 3 implies that map graphs with witness of large girth can be recognized in time $O(n^2m)$ (cf. [19]). No good characterization of map graphs with girth-six witness is known so far. It is also not known whether these map graphs can be recognized efficiently. Note that every planar graph has a girth-six witness, e.g., its subdivision.

3.2 Point minimal girth-constrained representations

This subsection deals with half-squares of (planar) bipartite graphs with girth constraints. We first consider the non-planar case.

Recall that, by Fact 1, POINT MINIMAL REPRESENTATION is equivalent to EDGE CLIQUE COVER, and thus is NP-complete. Also by Fact 1, POINT MINIMAL C_4-FREE REPRESENTATION is equivalent to EDGE CLIQUE PARTITION, and thus is NP-complete. Notice that C_4-free bipartite graphs and bipartite graphs of girth at least six coincide.

Now, let $t \geq 4$ be an integer and assume $G = B^2[V]$ for some bipartite graph $B = (V, W, E_B)$ of girth at least $2t$. By Lemma 1, any maximal clique in G is a neighborhood of some $w \in W$, implying $|W| \geq |\mathcal{C}(G)|$. Thus, by Theorem 2, B_G is a minimal point girth-$2t$ representation for G. Note that, in this case, $\mathcal{C}(G)$ can be computed in polynomial time (cf. [19]), hence we obtain:

▶ **Theorem 4.** POINT MINIMAL GIRTH-AT-LEAST-$2t$ REPRESENTATION *is* NP-*complete for $t \leq 3$ and solvable in polynomial time otherwise.*

In the remainder of this subsection, we deal with the planar case. We first consider the girth-four witness case, i.e., no girth condition is made. Recall that any map graph with n vertices has a witness with at most $3n-6$ points. We are going to show that finding a witness with minimal number of points is hard. We will use the fact that EDGE CLIQUE COVER remains NP-complete on maximal planar graphs without triangle-separators. More precisely, it was shown in [3], that EDGE CLIQUE COVER remains NP-complete for plane triangulations in which every triangle is a face. Observe that such a triangulation does not contain any 4-clique K_4, unless the whole graph is a K_4. Thus, we may further assume that all plane triangulations considered are K_4-free.

▶ **Theorem 5.** POINT MINIMAL PLANAR REPRESENTATION *is* NP-*complete, even when restricted to planar graphs.*

Proof. Let $G = (V, E_G)$ be a plane triangulation in which every triangle is a face, and let k be an integer. We will argue that G has an edge clique cover of size k or less if and only if G has a witness $B = (V, W, E_B)$ with $|W| \leq k$.

First, assume G has an edge clique cover $\mathcal{C}$ with $|\mathcal{C}| \leq k$. Note that we can assume that every clique in $\mathcal{C}$ is a triangle. Then, as any triangle in G is a face of the plane triangulation G, the vertex-$\mathcal{C}$ incidence bipartite graph $B_{\mathcal{C}} = (V, \mathcal{C}, E_B)$ is planar. Indeed, $B_{\mathcal{C}}$ is obtained from G by

- inserting a point w_T in the face T, $T \in \mathcal{C}$, and
- connecting w_T with the three vertices of the triangle T, and
- deleting all edges of G.

See also Figure 2. By Fact 1, as $\mathcal{C}$ is an edge clique cover of G, $G = B^2_{\mathcal{C}}[V]$, and by construction, $B_{\mathcal{C}}$ has $|\mathcal{C}| \leq k$ points.

Next, assume that $G = H^2[V]$ for some (planar) bipartite graph $H = (V, W, E_H)$ with $|W| \leq k$. Then, by Fact 1, $N_H(w)$, $w \in W$, form an edge clique cover of G with $|W| \leq k$ cliques. (Notice that, in this direction, we do not use the fact that H is planar. Any half-root of G with at most k points works.) ◀

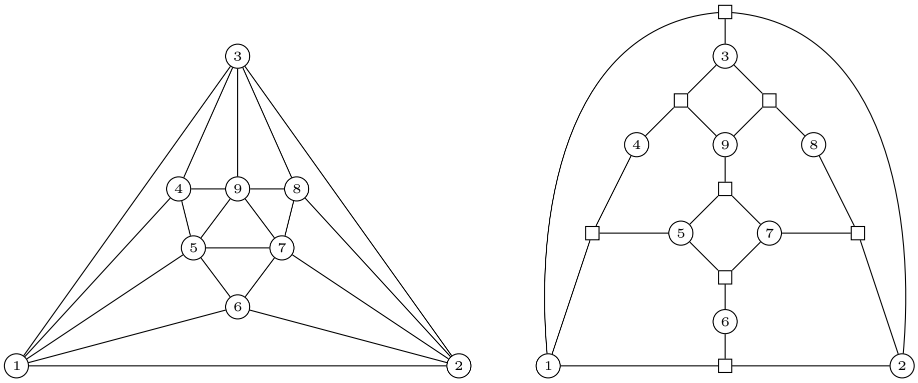


Figure 2 A triangulation G with the edge clique cover $\mathcal{C}$ consisting of the eight triangles 126, 278, 389, 349, 145, 567, 579, and 123, and the planar bipartite graph $B_{\mathcal{C}}$ obtained from G and $\mathcal{C}$.

We next consider the girth-six witness case. Recall that every planar graph has a witness of girth six. We are going to show that finding a witness of girth six with minimal number of points is hard. In a graph, a set of pairwise edge-disjoint triangles is called an independent triangle set. In [29], Uehara considered the following problem:

INDEPENDENT TRIANGLE SET
Instance: A graph $G = (V, E_G)$ and an integer k.
Question: Does G have k or more pairwise edge-disjoint triangles?

Uehara [29] proved that the INDEPENDENT TRIANGLE SET, restricted to plane triangulations, is NP-complete. Chang and Müller [3] observed that INDEPENDENT TRIANGLE SET is NP-complete even on plane triangulations in which every triangle is a face. (We leave the details to the full version.) Recall that we may further assume that all such plane triangulations are K_4-free.

▶ **Theorem 6.** POINT MINIMAL PLANAR GIRTH-6 REPRESENTATION *is* NP*-complete, even when restricted to planar graphs.*

Proof. Let $G = (V, E_G)$ be a plane triangulation without K_4 in which every triangle is a face, and let k be an integer. We will argue that G has k edge-disjoint triangles if and only if there is a C_4-free planar bipartite graph $B = (V, W, E_B)$ with $G = B^2[V]$ and $|W| \le m - 2k$. (As usual, m denotes the edge number of G.)

First, assume G has k edge-disjoint triangles $T_1, \ldots, T_k$. We construct a bipartite graph $B = (V, W, E_B)$ as follows; let F be the set of all edges of G not belonging to any triangle T_i, $1 \le i \le k$. For each $1 \le i \le k$, w_i is a point in W corresponding to T_i, and for each edge $e \in F$, w_e is a point in W corresponding to e. Then, B is the V-W incidence bipartite graph. Thus, $W = \{w_1, \ldots, w_k\} \cup \{w_e \mid e \in F\}$, and $E_B = \{vw_i \mid v \in V, v \in T_i, 1 \le i \le k\} \cup \{vw_e \mid v \in V, v \in e \in F\}$. Obviously, B is planar. Indeed, B is obtained from the plane triangulation G by

- inserting a point w_i in the face T_i, $1 \le i \le k$, and connecting w_i with the three vertices of T_i,
- subdividing each edge $e \in F$ by a point w_e, and
- deleting all edges in $T_1 \cup \ldots \cup T_k$.

Now, as each of the triangles T_i is a face of the plane triangulation G, B is clearly planar.

Since $\{T_1, \ldots, T_k\} \cup F$ is an edge clique cover of G, $G = B^2[V]$. Since the triangles T_i are edge-disjoint, $|N_B(w_i) \cap N_B(w_j)| \le 1$ for $1 \le i, j \le k$, $i \ne j$, and by definition of F, $|N_B(w_e) \cap N_B(w)| \le 1$ for all $e \in F$, $w \in W \setminus \{w_e\}$. That is, B is C_4-free. Moreover, B has $|W| = k + |F| = m - 2k$ points.

Next, assume that $G = H^2[V]$ for some (planar) C_4-free bipartite graph $H = (V, W, E_H)$ with $|W| \le m-2k$. Among all such bipartite graphs, let H have minimal number of points $|W|$. Since G is K_4-free, $|N_H(w)| \le 3$ for all $w \in W$. Since $|W|$ is minimal, $|N_H(w)| \ge 2$, and every two points have distinct neighborhoods. Let $w_1, \ldots, w_{k'}$ be the degree-3 points in W. Then, as H is C_4-free, $N_H(w_i)$, $1 \le i \le k'$, are k' edges-disjoint triangles in G. Since $G = H^2[V]$, G has $m = 3k' + (|W| - k') = |W| + 2k' \le m - 2k + 2k'$ edges. Therefore, $k' \ge k$. That is, G has at least k edges-disjoint triangles. ◀

Since, by Fact 1, POINT MINIMAL GIRTH-6 REPRESENTATION and EDGE CLIQUE PARTITION are equivalent, Theorem 6 implies:

▶ **Corollary 7.** EDGE CLIQUE PARTITION *is* NP*-complete on planar graphs.*

We remark that the complexity of EDGE CLIQUE PARTITION on planar graphs was previously unknown until this work (cf. [12]). Actually, the proof of Theorem 6 implies that EDGE CLIQUE PARTITION is NP-complete even for K_4-free maximal plane graphs in which every triangle is a face.

Now, let $t \ge 4$ be an integer, and assume that $G = B^2[V]$ for some planar bipartite graph $B = (V, W, E_B)$ of girth at least $2t$. By Lemma 1, any maximal clique in G is a neighborhood of some point $w \in W$, implying $|W| \ge |\mathcal{C}(G)|$. Thus, by Theorem 3, B_G is a minimal point planar girth-$2t$ representation for G. Note that, in this case, $\mathcal{C}(G)$ can be computed in polynomial time (cf. [19]), hence, by Theorems 5 and 6 we obtain:

▶ **Theorem 8.** POINT MINIMAL PLANAR GIRTH-AT-LEAST-$2t$ REPRESENTATION *is* NP*-complete for* $t \leq 3$ *and solvable in polynomial time otherwise.*

4 Tree-convex representations

Recall that a bipartite graph $B = (X, Y, E_B)$ is tree-convex on X (resp. Y) if there is a tree T on vertex set X (resp. Y) such that the neighborhood of any vertex $y \in Y$ (resp. $x \in X$) forms a subtree in T. B is tree-biconvex if it is both tree-convex on X and on Y. In this section we first characterize and recognize half-squares of tree-convex bipartite graphs and half-squares of tree-biconvex bipartite graphs. Characterizing and recognizing these half-squares have been left open in [18]. We then discuss the problem of determining such a representation with minimal number of points.

4.1 Half-squares of tree-convex bipartite graphs

In this section, we characterize half-squares of tree-convex bipartite graphs and of tree-biconvex bipartite graphs. It turns out that these are precisely the chordal and dually chordal graphs, and the double chordal graphs, respectively. We will use the following well known characterizations of chordal graphs which are classical by now (cf. [15, 24]).

▶ **Theorem 9.** *The following statements are equivalent for any graph* $G = (V, E)$*:*

(i) G *is chordal;*

(ii) G *is the vertex-intersection graph of subtrees in a tree: There are subtrees* T_v*,* $v \in V$*, of a tree* T *such that, for any pair of vertices* u, v *of* G*,* $uv \in E$ *if and only if* T_u *and* T_v *have a vertex in common;*

(iii) G *has a clique tree: There is a tree* T *on the maximal cliques of* G *with the property that, for any vertex* v *of* G*, the cliques containing* v *form a subtree of* T*.*

▶ **Lemma 10.** *Let* $B = (X, Y, E_B)$ *be a bipartite graph. If* B *is tree-convex on* Y*, then* $B^2[X]$ *is chordal and* $B^2[Y]$ *is dually chordal.*

Proof. Let B be tree-convex with an associated tree $T = (Y, E_T)$ such that, for each $v \in X$, $N_B(v)$ induces a subtree in T.

Then, by Theorem 9 (ii), $B^2[X]$ is chordal. We now show that $G = B^2[Y]$ is dually chordal. Note that we may assume that G is connected. Then T is a spanning tree of G. Indeed, consider an edge y_1y_2 of T, and let T_1 and T_2 be the two subtrees of $T - y_1y_2$ containing y_1 and y_2, respectively. Since $G = B^2[Y]$ is connected, some vertex $x \in X$ must have neighbors, in B, in both T_1 and T_2. Since $N_B(x)$ is a subtree of T, both y_1 and y_2 must be neighbors of such a vertex x. Therefore, y_1y_2 is an edge of $B^2[V] = G$, and hence T is a spanning tree of G as claimed. Now, consider an arbitrary maximal clique C of G, and suppose that $T[C]$ is not connected. Let $T_1, \ldots, T_q$ be the connected components of $T[C]$. Let $y \notin T[C]$ such that there is a connected component of $T - y$ contains only one of $T_1, \ldots, T_q$, say T_1. (As T is a tree, such vertex y exists.) Now, since C is a clique in G, for each $w \in T_1$ and $w' \in T_i$, $2 \leq i \leq q$, there is some $v \in X$ adjacent in B to both w and w'. Since $T[N(v)]$ is a subtree, v therefore must be adjacent in B to y. Thus, y is adjacent in G to every $w \in T_1$ and every $w' \in T_i$, $2 \leq i \leq q$. This contradicts the fact that C is a maximal clique in G. Thus, for any maximal clique C of G, $T[C]$ is a subtree in T, hence G is dually chordal. ◀

It follows from Lemma 10 that half-squares of tree-biconvex bipartite graphs are double chordal. The following lemma characterizes chordal graphs, dually chordal graphs and double chordal graphs in terms of their vertex-clique bipartite graphs.

▶ **Lemma 11.** *Let $G = (V, E_G)$ be a graph and let $B_G = (V, \mathcal{C}(G), E_B)$ the vertex-clique incidence bipartite graph of G. Then:*

(i) *G is chordal if and only if B_G is tree-convex on $\mathcal{C}(G)$.*

(ii) *G is dually chordal if and only if B_G is tree-convex on V.*

(iii) *G is double chordal if and only if B_G is tree-biconvex.*

Proof. (i): Let G be a chordal graph. By Theorem 9 (iii), G has a clique tree T. By definition of B_G and of T, for each $v \in V$, $N_{B_G}(v) = \{C \in \mathcal{C}(G) \mid v \in C\}$ induces a subtree in T. That is, B_G is tree-convex on $\mathcal{C}(G)$. The conserve follows from Lemma 10 with $B = B_G$, where $X = V$ and $Y = \mathcal{C}(G)$. (Recall that $G = B_G^2[V] = B^2[X]$.)

(ii): Let G be a (connected) dually chordal graph, and let $T = (V, E_T)$ be a spanning tree of G such that, for each maximal clique C of G, $T[C]$ is a subtree of T. Then, by definition of B_G, for each $w \in W$, $N_{B_G}(w)$ is a maximal clique in G, hence $N_{B_G}(w)$ induces a subtree in T. The converse follows from Lemma 10 with $B = B_G$, where $X = \mathcal{C}(G)$ and $Y = V$.

(iii): This part immediately follows from (i) and (ii). ◀

By Lemmas 10 and 11 we obtain:

▶ **Theorem 12.**

(i) *A graph is a half-square of a tree-convex bipartite graph if and only if it is chordal or dually chordal.*

(ii) *Half-squares of tree-biconvex bipartite graphs are exactly the double chordal graphs.*

Since chordal and dually chordal graphs, hence double chordal graphs, can be recognized in linear time, we obtain from Theorem 12:

▶ **Corollary 13.** *Deciding if a given graph is a half-square of a tree-(bi)convex bipartite graph can be done in linear time.*

4.2 Point minimal tree-convex representations

In the remainder of this section, we first show that POINT MINIMAL TREE-BICONVEX REPRESENTATION is solvable in linear time and then prove a complexity dichotomy theorem for POINT MINIMAL TREE-CONVEX REPRESENTATION.

We will show that, in fact, $\langle G, k \rangle$ is a yes-instance for POINT MINIMAL TREE-BICONVEX REPRESENTATION if and only if G is double chordal and k is at least the number of maximal cliques of G, $k \geq |\mathcal{C}(G)|$. If we ask for a tree-convex (not necessarily tree-biconvex) representation, k may be much smaller than $|\mathcal{C}(G)|$. See Figure 3 for an example.

The following fact will be useful in later discussions:

▶ **Lemma 14.** *Let $B = (X, Y, E_B)$ be tree-convex on Y. Then, for each maximal clique C of $B^2[X]$, there is some $y \in Y$ with $C = N_B(y)$.*

Proof. Let $T = (Y, E_T)$ be a tree such that, for any $x \in X$, $T[N_B(x)]$ is a subtree of T. Let C be a maximal clique in $B^2[X]$. Let $y \in T$ be a vertex with maximum $|N_B(y) \cap C|$. Suppose there is some $x \in C \setminus N_B(y)$. Let $w \in T$ be a neighbor of x in B that is closest to y in T, and let T_w be the connected component of $T - wy'$ containing w, where wy' is the w-edge on the w,y-path in T (possibly $y' = y$). Since B is tree-convex on Y with

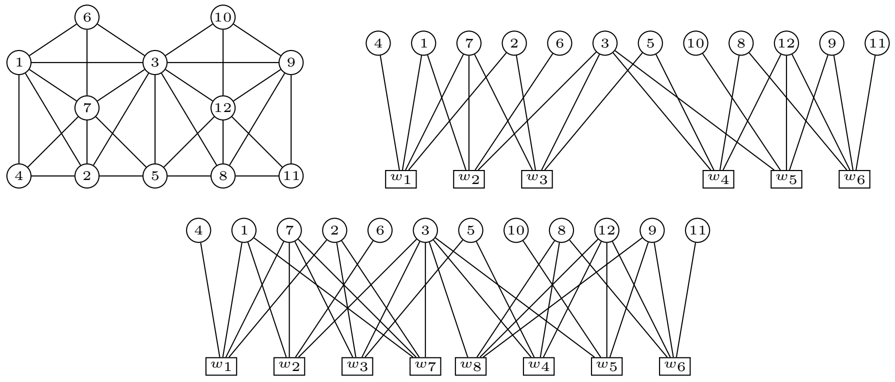


Figure 3 A double chordal graph G (top left), a point minimal tree-convex, not tree-biconvex, half-square root B (top right) and a point minimal tree-biconvex half-square root B_G of G (bottom).

tree T, $N_B(x) \subseteq T_w$. By the choice of y, there is some $x' \in N_B(y) \cap C \setminus N_B(w)$. As B is tree-convex on Y with tree T, we have $N_B(x') \subseteq T_y$ with T_y is the connected component of $T - yy''$ containing y, where yy'' is the y-edge on the y, w-path in T (possibly $y'' = w$). Since $T_w \cap T_y = \emptyset$, we have $N_B(x) \cap N_B(x') = \emptyset$. This contradicts the fact that x and x' are adjacent in $B^2[V]$.

Thus, $C \subseteq N_B(y)$, and by the maximality of the clique C, $C = N_B(y)$. ◀

Notice that tree-convex bipartite graphs need not be C_6-free, and C_6-free bipartite graphs need not be tree-convex. So, Lemma 14 and Lemma 1 are independent to each other.

▶ **Theorem 15.** POINT MINIMAL TREE-BICONVEX REPRESENTATION *is solvable in linear time.*

Proof. Let $\langle G, k \rangle$ be an instance for POINT MINIMAL TREE-BICONVEX REPRESENTATION. By Theorem 12 (ii) we may assume that $G = (V, E_G)$ is double chordal. Let $B = (V, W, E_G)$ be an arbitrary tree-biconvex bipartite graph with $G = B^2[V]$. By Lemma 14, every maximal clique C of G is the neighborhood $N_B(w)$ for some $w \in W$, implying $|W| \geq |\mathcal{C}(G)|$. Therefore, the vertex-clique incidence bipartite graph B_G of G (which is tree-biconvex by Lemma 11 (iii)) is a point optimal tree-biconvex half-root of G. That is, $\langle G, k \rangle$ is a yes-instance if and only if G is double chordal and $k \geq |\mathcal{C}(G)|$.

Finally, recall that double chordal graphs can be recognized in linear time, and that all maximal cliques of an n-vertex chordal graph (there are at most n) can be computed in linear time (and so B_G can be constructed in linear time, too). ◀

We now are providing a dichotomy for POINT MINIMAL TREE-CONVEX REPRESENTATION. We first begin with the hardness case.

▶ **Lemma 16.** POINT MINIMAL TREE-CONVEX REPRESENTATION *is* NP*-complete, when restricted to non-chordal dually chordal input graphs.*

Proof. Given an instance $\langle G = (V, E_G), k \rangle$ of POINT MINIMAL REPRESENTATION, construct an instance $\langle G', k' \rangle$ for POINT MINIMAL TREE-CONVEX REPRESENTATION as follows.

- G' is obtained from G by adding a new vertex u and all edges between u and all vertices of G, i.e., u is a universal vertex of G';
- $k' := k$.

Suppose that $G = B^2[V]$ for some bipartite graph $B = (V, W, E_B)$ with $|W| \leq k$. Consider $B' = (V', W', E_{B'})$ with $V' = V \cup \{u\}$, $W' = W$ and $E_{B'} = E_B \cup \{uw \mid w \in W\}$. Then it is easy to see that $G' = B'^2[V']$. Notice, moreover, that G' is dually chordal (as u is a universal vertex of G'), and B' is tree-convex (with a star $T = (V, \{uv \mid v \in V\})$).

Conversely, if $G' = H^2[V']$ for some bipartite graph $H = (V', W, E_H)$ with $|W| \leq k'$, regardless tree-convex or not, then clearly $G = B^2[V]$, where $B = H - u$ with at most $k = k'$ points.

Since POINT MINIMAL REPRESENTATION (viz., CLIQUE EDGE COVER) is NP-complete on non-chordal graphs, POINT MINIMAL TREE-CONVEX REPRESENTATION is NP-complete, when restricted to non-chordal dually chordal graphs. ◀

For the efficient solvable cases, we need the following characterization of dually chordal graphs which is slightly more flexible than the one stated in Lemma 11 (ii), and can be proved along the same line. (Notice that the characterization of chordal and double chordal graphs stated in Lemma 11 (i) and (iii), respectively, does not admit this flexibility.)

▶ **Lemma 17.** *Let $G = (V, E_G)$ be a graph and let $\mathcal{C}$ be an edge clique cover of G in which every member is a maximal clique. Let $B_{\mathcal{C}} = (V, \mathcal{C}, E_B)$ be the vertex-$\mathcal{C}$ incidence bipartite graph of G. Then G is dually chordal if and only if $B_{\mathcal{C}}$ is tree-convex on V.*

Notice that any edge clique cover can be modified in an obvious way to another one of the same size that consists of maximal cliques only.

▶ **Theorem 18.** POINT MINIMAL TREE-CONVEX REPRESENTATION *is* NP*-complete for non-chordal dually chordal inputs, and solvable in linear time otherwise.*

Proof. Let $\langle G, k\rangle$ be an instance for POINT MINIMAL TREE-CONVEX REPRESENTATION. By Theorem 12, we may assume that $G = (V, E_G)$ is chordal or dually chordal (otherwise, the output is 'no' as G does not have a tree-convex representation). Recall that chordal, as well as dually chordal graphs can be recognized in linear time.

By Lemma 16, it remains to consider the case in which G is chordal. The following procedure decides in linear time if G has a tree-convex representation with at most k points and, if so, outputs such one.

```
(1)  if $G$ is double chordal then
(2)      compute an optimal edge clique cover $\mathcal{C}$ of $G$ that consists of
(3)      maximal cliques only
(4)      if $k < |\mathcal{C}|$ then return 'no'
(5)      return the vertex-$\mathcal{C}$ incidence bipartite graph $B_{\mathcal{C}}$
(6)  else  // $G$ is chordal but not dually chordal
(7)      if $k < |\mathcal{C}(G)|$ then return 'no'
(8)      return the vertex-clique incidence bipartite graph $B_G$
```

Since G is chordal, an optimal edge clique cover $\mathcal{C}$ can be computed in linear time [23]. In fact, the optimal edge clique cover of a chordal graph computed in [23] consists of maximal cliques only. Also, recall that for any chordal graph $G = (V, E_G)$, $\mathcal{C}(G)$ consists of at most $|V|$ maximal cliques and all maximal cliques can be listed in linear time.

We now argue that the output of the procedure is a tree-convex representation with at most k points (if exists). Assume first that G is dually chordal (and hence G is double chordal). Then $B_{\mathcal{C}}$ is tree-convex (on V) by Lemma 17. Thus, by Fact 1, $B_{\mathcal{C}}$ is a point optimal tree-convex representation of G. So, the outputs at lines (4) and (5) are correct.

In the second case, let us assume that G is not dually chordal. Then, by Lemma 10, for any tree-convex representation $B = (V, W, E_B)$ of G, B must be tree-convex on W. Hence, by Lemma 14, every maximal clique of $G = B^2[V]$ is the neighborhood $N_B(w)$ for some $w \in W$, implying $|W| \geq |\mathcal{C}(G)|$. Therefore, the vertex-clique incidence bipartite graph B_G of G (which is, by Lemma 11 (i), tree-convex on $\mathcal{C}(G)$; recall that G is chordal) is a point optimal tree-convex half-root of G. Thus, the outputs at lines (7) and (8) are correct. ◀

5 Conclusion

Though the computational complexity of MINIMAL POINT PLANAR GIRTH-g REPRESENTATION is completely determined (Theorem 8), the problem of characterizing and recognizing map graphs with girth-6 witness is still open. Recall that, by definition, all map graphs have a witness of girth at least 4, and that all map graphs with witness of girth $g \geq 8$ admit good characterizations which lead to simple cubic time recognition algorithms [19]. Since any planar graph has a girth-six witness (e.g., its subdivision), it is natural to study map graphs with girth-six witness. Thus, recognizing and characterizing map graphs with girth-six witness are two interesting open problems.

In contrast to large-girth witnesses, maximal witnesses (i.e., maximal planar bipartite graphs) have girth four. Recognizing and characterizing map graphs with maximal witness are two other interesting open problems for further research.

Perhaps, another way to look for a simpler and more efficient algorithm than the one of Thorup is to consider restricted input graphs (rather than restricted witnesses). So, recognizing map graphs is trivial if the input graphs are planar. But it is not obvious for other restricted graph classes; especially for graphs with arbitrary large cliques. In particular, it seems that it is not easy to recognize chordal map graphs in polynomial time without using Thorup's algorithm.

References

1 Franz J. Brandenburg. Characterizing and Recognizing 4-Map Graphs. *Algorithmica*, 81(5):1818–1843, 2019. doi:10.1007/s00453-018-0510-x.

2 Andreas Brandstädt, Feodor F. Dragan, Victor Chepoi, and Vitaly I. Voloshin. Dually Chordal Graphs. *SIAM J. Discrete Math.*, 11(3):437–455, 1998. doi:10.1137/S0895480193253415.

3 Maw-Shang Chang and Haiko Müller. On the Tree-Degree of Graphs. In *Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001, Boltenhagen, Germany, June 14-16, 2001, Proceedings*, pages 44–54, 2001. doi:10.1007/3-540-45477-2_6.

4 Zhi-Zhong Chen. Approximation Algorithms for Independent Sets in Map Graphs. *J. Algorithms*, 41(1):20–40, 2001. doi:10.1006/jagm.2001.1178.

5 Zhi-Zhong Chen, Michelangelo Grigni, and Christos H. Papadimitriou. Planar Map Graphs. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 514–523, 1998. doi:10.1145/276698.276865.

6 Zhi-Zhong Chen, Michelangelo Grigni, and Christos H. Papadimitriou. Map graphs. *J. ACM*, 49(2):127–138, 2002. doi:10.1145/506147.506148.

7 Zhi-Zhong Chen, Michelangelo Grigni, and Christos H. Papadimitriou. Recognizing Hole-Free 4-Map Graphs in Cubic Time. *Algorithmica*, 45(2):227–262, 2006. doi:10.1007/s00453-005-1184-8.

8 Zhi-Zhong Chen, Xin He, and Ming-Yang Kao. Nonplanar Topological Inference and Political-Map Graphs. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA.*, pages 195–204, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.314558.

9 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Trans. Algorithms*, 1(1):33–47, 2005. doi:10.1145/1077464.1077468.

10 Erik D. Demaine and MohammadTaghi Hajiaghayi. The Bidimensionality Theory and Its Algorithmic Applications. *Comput. J.*, 51(3):292–302, 2008. doi:10.1093/comjnl/bxm033.

11 Kord Eickmeyer and Ken-ichi Kawarabayashi. FO model checking on map graphs. In *Fundamentals of Computation Theory - 21st International Symposium, FCT 2017, Bordeaux, France, September 11-13, 2017, Proceedings*, pages 204–216, 2017. doi:10.1007/978-3-662-55751-8_17.

12 Rudolf Fleischer and Xiaotian Wu. Edge Clique Partition of K_4-Free and Planar Graphs. In *Computational Geometry, Graphs and Applications - 9th International Conference, CGGA 2010, Dalian, China, November 3-6, 2010, Revised Selected Papers*, pages 84–95, 2010. doi:10.1007/978-3-642-24983-9_9.

13 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Decomposition of Map Graphs with Applications. *CoRR*, abs/1903.01291, 2019. arXiv:1903.01291.

14 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Bidimensionality and geometric graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1563–1575, 2012. doi:10.1137/1.9781611973099.124.

15 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, 1980. doi:10.1016/C2013-0-10739-8.

16 Ian Holyer. The NP-Completeness of Some Edge-Partition Problems. *SIAM J. Comput.*, 10(4):713–717, 1981. doi:10.1137/0210054.

17 Lawrence T. Kou, Larry J. Stockmeyer, and C. K. Wong. Covering Edges by Cliques with Regard to Keyword Conflicts and Intersection Graphs. *Commun. ACM*, 21(2):135–139, 1978. doi:10.1145/359340.359346.

18 Hoàng-Oanh Le and Van Bang Le. Hardness and structural results for half-squares of restricted tree-convex bipartite graphs. *Algorithmica, in press*, 2019. doi:10.1007/s00453-018-0440-7.

19 Hoàng-Oanh Le and Van Bang Le. Map graphs having witnesses of large girth. *Theor. Comput. Sci.*, 772:143–148, 2019. doi:10.1016/j.tcs.2018.12.010.

20 Van Bang Le and Sheng-Lung Peng. On the complete width and edge clique cover problems. *J. Comb. Optim.*, 36(2):532–548, 2018. doi:10.1007/s10878-016-0106-9.

21 Tian Liu. Restricted Bipartite Graphs: Comparison and Hardness Results. In *Algorithmic Aspects in Information and Management - 10th International Conference, AAIM 2014, Vancouver, BC, Canada, July 8-11, 2014. Proceedings*, pages 241–252, 2014. doi:10.1007/978-3-319-07956-1_22.

22 S. Ma, Walter D. Wallis, and Julin Wu. On the complexity of the clique partition problem. *Congressus Numerantium*, 67:59–66, 1988.

23 S. Ma, Walter D. Wallis, and Julin Wu. Clique Covering of Chordal Graphs. *Utilitas Mathematica*, 36:151–152, 1989.

24 Terry A. McKee and F. R. McMorris. *Topics in Intersection Graph Theory.* SIAM, 1999. URL: https://epubs.siam.org/doi/book/10.1137/1.9780898719802.

25 Matthias Mnich, Ignaz Rutter, and Jens M. Schmidt. Linear-time recognition of map graphs with outerplanar witness. *Discrete Optimization*, 28:63–77, 2018. doi:10.1016/j.disopt.2017.12.002.

26 James Orlin. Contentment in graph theory: covering graphs with cliques. *Indagationes Mathematicae*, 80:406–424, 1977. doi:10.1016/1385-7258(77)90055-5.

27 Jeremy P. Spinrad. *Efficient Graph Representations.* Fields Institute Monographs, 2003.

28 Mikkel Thorup. Map Graphs in Polynomial Time. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 396–405, 1998. doi:10.1109/SFCS.1998.743490.

29 Ryuhei Uehara. NP-complete problems on a 3-connected cubic planar graph and their applications. *Tokyo Woman's Christian University*, Technical Report TWCU-M-0004, 1996/9. URL: `http://www.jaist.ac.jp/~uehara/pdf/triangle.pdf`.

30 W. D. Wallis and Julin Wu. On clique partitions of split graphs. *Discrete Mathematics*, 92(1-3):427–429, 1991. `doi:10.1016/0012-365X(91)90297-F`.

Colouring H-Free Graphs of Bounded Diameter

Barnaby Martin
Department of Computer Science, Durham University, United Kingdom
barnaby.d.martin@durham.ac.uk

Daniël Paulusma
Department of Computer Science, Durham University, United Kingdom
daniel.paulusma@durham.ac.uk

Siani Smith
Department of Computer Science, Durham University, United Kingdom
siani.smith@durham.ac.uk

Abstract

The Colouring problem is to decide if the vertices of a graph can be coloured with at most k colours for an integer k, such that no two adjacent vertices are coloured alike. A graph G is H-free if G does not contain H as an induced subgraph. It is known that Colouring is NP-complete for H-free graphs if H contains a cycle or claw, even for fixed $k \geq 3$. We examine to what extent the situation may change if in addition the input graph has bounded diameter.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases vertex colouring, H-free graph, diameter

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.14

Funding *Daniël Paulusma*: supported by the Leverhulme Trust (RPG-2016-258).

1 Introduction

Graph colouring is one of the best studied concepts in Computer Science and Mathematics. This is mainly due to its many practical and theoretical applications and its many natural variants and generalizations. Over the years, numerous surveys and books on graph colouring were published (see, for example, [1, 4, 18, 21, 26, 28, 31]).

A *(vertex) colouring* of a graph $G = (V, E)$ is a mapping $c : V \to \{1, 2, \ldots\}$ that assigns each vertex $u \in V$ a *colour* $c(u)$ in such a way that $c(u) \neq c(v)$ whenever $uv \in E$. If $1 \leq c(u) \leq k$, then c is said to be a *k-colouring* of G and G is said to be *k-colourable*. The Colouring problem is to decide if a given graph G has a k-colouring for some given integer k. If k is *fixed*, that is, k is not part of the input, we denote the problem by k-Colouring. It is well known that even 3-Colouring is NP-complete [23].

In this paper we aim to increase our understanding of the computational hardness of Colouring. One way to do this is to consider inputs from families of graphs to learn more about the kind of graph structure that causes the hardness. This led to a highly extensive study of Colouring and k-Colouring for many special graph classes. The best-known result in this direction is due to Grötschel, Lovász, and Schrijver, who proved that Colouring is polynomial-time solvable for perfect graphs [13].

Perfect graphs form an example of a graph class that is closed under vertex deletion. Such graph classes are also called *hereditary*. Hereditary graph classes are ideally suited for a *systematic* study in the computational complexity of graph problems. Not only do they capture a very large collection of many well-studied graph classes, but they are also exactly the graph classes that can be characterized by a unique set $\mathcal{H}$ of minimal forbidden induced subgraphs. When solving an NP-hard problem under input restrictions, it is standard practice to consider, for example, first the case where $\mathcal{H}$ has small size, or where each $H \in \mathcal{H}$ has small size.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 14; pp. 14:1–14:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We note that the set $\mathcal{H}$ defined above may be infinite. If not, say $\mathcal{H} = \{H_1, \ldots, H_p\}$ for some positive integer p, then the corresponding hereditary graph class $\mathcal{G}$ is said to be *finitely defined*. Formally, a graph G is $(H_1, \ldots, H_p)$*-free* if for each $i \in \{1, \ldots, p\}$, G is H_i*-free*, where the latter means that G does not contain an induced subgraph isomorphic to H_i.

We emphasize that the borderline between NP-hardness and tractability is often far from clear beforehand and jumps in computational complexity can be extreme. In order to illustrate this behaviour of graph problems, we present the following example of a (somewhat artificial) graph problem related to vertex colouring.

Colouring-or-Subgraph
Instance: an n-vertex graph G
Question: is G $\lceil\sqrt{\log n}\rceil$-colourable or H-free for some graph H with $|V(H)| \leq \lceil\sqrt{\log n}\rceil$?

▶ **Theorem 1.** *The* Colouring-or-Subgraph *problem is* NP*-hard, but constant-time solvable for every hereditary graph class not equal to the class of all graphs.*

Proof. We reduce from 3-Colouring, which we recall is NP-complete [23]. Let G be an n-vertex graph. Set $k = \lceil\sqrt{\log n}\rceil$. Add $k - 3$ pairwise adjacent vertices to G. Make the new vertices also adjacent to every vertex of G. Add each possible graph on k vertices as a connected component to G. The resulting graph G' has $n + (k-3) + k \cdot 2^{\frac{k(k-1)}{2}} < 3n^2$ vertices. By construction, G' contains every graph on at most k vertices as an induced subgraph. Hence, G' is a yes-instance of Colouring-or-Subgraph if and only if G' is k-colourable, and the latter holds if and only if G is 3-colourable.

Now let $\mathcal{G}$ be a hereditary graph class for which there exist at least one graph H such that every graph $G \in \mathcal{G}$ is H-free. Let $\ell = |V(H)|$. We claim that Colouring-or-Subgraph is constant-time solvable for $\mathcal{G}$. Let $G \in \mathcal{G}$ be an n-vertex graph. If $n \leq 2^{|\ell|^2}$, then G has constant size and the problem is constant-time solvable. If $n > 2^{|\ell|^2}$, then $\ell = |V(H)| < \sqrt{\log n} \leq \lceil\sqrt{\log n}\rceil$. Hence G is a yes-instance of Colouring-or-Subgraph, as G is H-free and H has less than $\lceil\sqrt{\log n}\rceil$ vertices. ◀

In this paper, we consider the problems Colouring and k-Colouring. In order to describe known results and our new results we first give some terminology and notation.

1.1 Terminology and Notation

The *disjoint union* of two vertex-disjoint graphs F and G is the graph $G + F = (V(F) \cup V(G), E(F) \cup E(G))$. The disjoint union of s copies of a graph G is denoted sG. A *linear forest* is the disjoint union of paths. The *length* of a path or a cycle is the number of its edges. The *distance* $\text{dist}(u, v)$ between two vertices u, v in a graph G is the length of a shortest induced path between them. The *diameter* of a graph G is the maximum distance over all pairs of vertices in G. The *girth* of a graph G is the length of a shortest induced cycle of G. The graphs C_r, P_r and K_r denote the cycle, path and complete graph on r vertices, respectively.

A *polyad* is a tree where exactly one vertex has degree at least 3. We will use several special polyads in our paper. The graph $K_{1,r}$ denotes the $(r+1)$-vertex *star*, that is, the graph with vertices $x, y_1, \ldots, y_r$ and edges xy_i for $i = 1, \ldots, r$. The graph $K_{1,3}$ is also called the *claw*. The *subdivision* of an edge uw in a graph removes uw and replaces it with a new vertex v and edges uv, vw. We let $K_{1,r}^{\ell}$ denote the *ℓ-subdivided star*, which is the graph obtained from a star $K_{1,r}$ by subdividing one edge of $K_{1,r}$ exactly ℓ times. The graph $S_{h,i,j}$, for $1 \leq h \leq i \leq j$, denotes the *subdivided claw*, which is the tree with one vertex x of degree 3 and exactly three leaves, which are of distance h, i and j from x, respectively. Note that $S_{1,1,1} = K_{1,3}$. The graph $S_{1,1,2} = K_{1,3}^1$ is also known as the *chair*.

1.2 Known Results

The computational complexity of Colouring has been fully classified for H-free graphs: if H is an induced subgraph of $P_1 + P_3$ or of P_4, then Colouring for H-free graphs is polynomial-time solvable, and otherwise it is NP-complete [20]. In contrast, the complexity classification for k-Colouring restricted to H-free graphs is still incomplete. It is known that for every $k \geq 3$, k-Colouring for H-free graphs is NP-complete if H contains a cycle [10] or an induced claw [16, 22]. However, the remaining case where H is a linear forest has not been settled yet even if H consists of a single path. For P_t-free graphs, the cases $k \leq 2$, $t \geq 1$ (trivial), $k \geq 3$, $t \leq 5$ [14], $k = 3$, $6 \leq t \leq 7$ [2] and $k = 4$, $t = 6$ [6] are polynomial-time solvable and the cases $k = 4$, $t \geq 7$ [17] and $k \geq 5$, $t \geq 6$ [17] are NP-complete. The cases where $k = 3$ and $t \geq 8$ are still open. For further details, including for linear forests H of more than one connected component, see the survey paper [11] or some recent papers [5, 12, 19].

1.3 Our Focus

We consider H-free graphs where H contains a cycle or claw. In this case, k-Colouring restricted to H-free graphs is NP-compete for every $k \geq 3$, as mentioned above. However, we re-examine the situation after adding a diameter constraint to our input graphs. If the diameter is 1, then G is a complete graph, and Colouring becomes trivial. As such, our research question is:

To what extent does bounding the diameter help making Colouring *and* k-Colouring *tractable on H-free graphs?*

We remark that H-free graphs of diameter at most d for some integer d are no longer hereditary, which requires some care in the proof of our results. We also note that by a straightforward reduction from 3-Colouring one can show that k-Colouring is NP-complete for graphs of diameter d for all pairs (k, d) with $k \geq 3$ and $d \geq 2$ except for two cases, namely $(k, d) \in \{(3, 2), (3, 3)\}$. Mertzios and Spirakis [24] settled the case $(k, d) = (3, 3)$ by proving that 3-Colouring is NP-complete even for C_3-free graphs of diameter 3. The case $(k, d) = (3, 2)$ is still open.

1.4 Our Results

We complement the bounded diameter results of Mertzios and Spirakis [24] by presenting a set of new results for Colouring and k-Colouring for H-free graphs of bounded diameter when H contains a claw or a cycle. Results for the case where H has a cycle usually follow from stronger results for graphs of girth at least g for some fixed integer g. In particular, Emden-Weinert, Hougardy and Kreuter [10] proved that for all integers $k \geq 3$ and $g \geq 3$, k-Colouring is NP-complete for graphs with girth at least g and with maximum degree at most $6k^{13}$ (for more results on Colouring for graphs of maximum degree, see [3, 7, 25]).

First, in Section 3 we research the effect on bounding the diameter of k-Colouring and Colouring restricted to polyad-free graphs for various polyads. Our first result, which formed together with the result of [24] the starting point of our investigation, is that k-Colouring is constant-time solvable for $K_{1,r}$-free graphs of diameter d for any fixed integers $d \geq 1$, $k \geq 1$ and $r \geq 1$. We also show that this does not hold for Colouring (when k is part of the input). We then extend these results for larger polyads; see also Figure 1.

Second, in Section 4 we perform a similar study for graphs of bounded diameter and girth. We provide new polynomial-time and NP-hardness results for k-Colouring, identifying and

Colours	Diameter	H-free	Complexity	Theorem
fixed k	d	$K_{1,r}$	P	9
input k	d	$K_{1,4}$	NP-c	10
3	d	$K_{1,3}^1$	P	12(1)
3	2	$K_{1,r}^2$	P	12(2)
3	4	$K_{1,4}^3$	NP-c	12(3)
4	2	$K_{1,3}^1$	NP-c	12(4)
3	2	$S_{1,2,2}$	P	13

Figure 1 Our polynomial-time (P) and NP-complete (NP-c) results for polyad-free graphs.

narrowing the gap between tractability and intractability, in particular for the case where $k = 3$ (see also Figure 2). Section 5 contains some open questions and directions for future work.

girth diameter	≥ 3	≥ 4	≥ 5	≥ 6	≥ 7	≥ 8	≥ 9	≥ 10	≥ 11	≥ 12
≤ 1	P	P	P	P	P	P	P	P	P	P
≤ 2	?	?	P	P	P	P	P	P	P	P
≤ 3	NP-c	NP-c	?	?	P	P	P	P	P	P
≤ 4	NP-c	NP-c	NP-c	NP-c	?	?	P	P	P	P
≤ 5	NP-c	NP-c	NP-c	NP-c	?	?	?	?	?	P

Figure 2 The complexity of 3-Colouring for graphs of diameter at most d and girth at least g.

2 Preliminaries

In this section we complement Section 1.1 by giving some additional terminology and notation. We also recall some useful results from the literature.

Let $G = (V, E)$ be a graph. A vertex $u \in V$ is *dominating* if u is adjacent to every other vertex of G. For a set $S \subseteq V$, the graph $G[S]$ denotes the subgraph of G induced by S. The *neighbourhood* of a vertex $u \in V$ is the set $N(u) = \{v \mid uv \in E\}$ and the *degree* of u is the size of $N(u)$. For a set $U \subseteq V$, we write $N(U) = \bigcup_{u \in U} N(u) \setminus U$. For a set $U \subseteq V$ and a vertex $u \in U$, the *private neighbourhood* of u with respect to U is the set $N(u) \setminus (N(U \setminus \{u\}) \cup U)$ of *private neighbours* of u with respect to U, which is the set of neighbours of u outside U that are not a neighbour of any other vertex of U. If every vertex of G has degree p, then G is *(p)-regular*.

We will use the aforementioned results of Král' et al.; Holyer; Leven and Galil; Emden-Weinert, Hougardy and Kreuter; and Mertzios and Spirakis.

▶ **Theorem 2** ([20]). *Let H be a graph. If $H \subseteq_i P_4$ or $H \subseteq_i P_1 + P_3$, then* Colouring *restricted to H-free graphs is polynomial-time solvable, otherwise it is* NP*-complete.*

▶ **Theorem 3** ([16, 22]). *For every integer $k \geq 3$, k-*Colouring *is* NP*-complete for claw-free graphs.*

▶ **Theorem 4** ([10]). *For all integers $k \geq 3$ and $g \geq 3$, k-*Colouring *is* NP*-complete for graphs with girth at least g (and with maximum degree at most $6k^{13}$).*

▶ **Theorem 5** ([24]). 3-Colouring *is NP-complete for C_3-free graphs of diameter* 3.

A *list assignment* of a graph $G = (V, E)$ is a function L that prescribes a *list of admissible colours* $L(u) \subseteq \{1, 2, \ldots\}$ to each $u \in V$. A colouring c *respects* L if $c(u) \in L(u)$ for every $u \in V$. If $|L(u)| \leq 2$ for each $u \in V$, then L is also called a 2-*list assignment*. The 2-List Colouring problem is to decide if a graph G with a 2-list assignment L has a colouring that respects G. Our strategy for obtaining a polynomial-time algorithm for 3-Colouring is often to reduce the input to a polynomial number of instances of 2-List Colouring. The reason is that we can then apply the following well-known result of Edwards.

▶ **Theorem 6** ([9]). *The* 2-List Colouring *problem is linear-time solvable.*

We will also use the following result, which includes the Hoffman-Singleton Theorem, which provides a description of regular graphs of diameter 2 and girth 5.

▶ **Theorem 7** ([8, 15, 30]). *For every $d \geq 1$, every graph of diameter d and girth $2d + 1$ is p-regular for some integer p. Moreover, if $d = 2$, then there are only four such graphs (with $p = 2, 3, 7, 57$, respectively) and if $d \geq 3$, then such graphs are cycles (of length $2d + 1$).*

A *clique* in a graph is a set of pairwise adjacent vertices, and an *independent set* is a set of pairwise non-adjacent vertices. By Ramsey's Theorem [27], there exists a constant, which we denote by $R(k, r)$, such that any graph on at least $R(k, r)$ vertices contains either a clique of size k or an independent set of size r.

3 Polyad-Free Graphs of Bounded Diameter

In this section we prove, among other things, our results on Colouring and k-Colouring for polyad-free graphs of bounded diameter; see also Figure 1. We first make an observation.

▶ **Lemma 8.** *If G is a graph of diameter d that is not a tree, then G contains an induced cycle of length at most $2d + 1$.*

Proof. As G is not a tree and G is connected, G must contain a cycle C. Suppose that C has length at least $2d + 2$. Since G has diameter d, there exists a path of length at most d in G between any two vertices u and v at distance $d + 1$ in C. The vertices of this path, together with the vertices of the path of length $d + 1$ between u and v on C, induce a subgraph of G that contains an induced cycle C' of length at most $2d + 1$. ◀

We now state our first result, which forms the starting point of the research in this section.

▶ **Theorem 9.** *For all integers $d, k, r \geq 1$, k-*Colouring *is constant-time solvable for $K_{1,r}$-free graphs of diameter d.*

Proof. Let $G = (V, E)$ be a $K_{1,r}$-free graph of diameter d. We prove that if G has size larger than some constant $\beta(k, r)$, which we determine below, then G is not k-colourable. If $|V(G)| \leq \beta(k, r)$, we can solve k-Colouring in constant time.

As G is $K_{1,r}$-free, Ramsey's Theorem tells us that the neighbourhood of every vertex $u \in V$ with degree at least $R(k, r)$ contains a clique of size k. In that case $N(u) \cup \{u\}$ is a clique of size $k + 1$. Hence, to be k-colourable, every vertex of G must have degree less than $R(k, r)$, so G must have at most $\beta(k, r) = 1 + R(k, r) + R(k, r)^2 + \ldots + R(k, r)^d$ vertices. ◀

If k is not part of the input, Theorem 9 no longer holds. This is shown by the following more general theorem. In this theorem we assume that $H \not\subseteq_i P_1 + P_3$ and $H \not\subseteq_i P_4$, as in those cases Colouring is polynomial-time solvable for all H-free graphs due to Theorem 2. Note that Theorem 10 covers all remaining cases except the case where $H = K_{1,3}$.

▶ **Theorem 10.** *Let H be a graph with $H \not\subseteq_i P_1 + P_3$ and $H \not\subseteq_i P_4$ and d be an integer. Then* Colouring *for H-free graphs of diameter at most d is*

1. *NP-complete if H has no dominating vertex u such that $H - u \subseteq_i P_1 + P_3$ or $H - u \subseteq_i P_4$ and $d \geq 2$;*
2. *NP-complete if $H \neq K_{1,3}$ and H has a dominating vertex u such that $H - u \subseteq_i P_1 + P_3$ or $H - u \subseteq_i P_4$ and $d \geq 3$.*

Proof.

1. Let H have no dominating vertex u such that $H-u \subseteq_i P_1+P_3$ or $H-u \subseteq_i P_4$. We define H' as $H - u$ if H has a dominating vertex u and as H itself otherwise. By construction, $H' \not\subseteq_i P_1 + P_3$ and $H' \not\subseteq_i P_4$. Hence, Colouring is NP-complete for H'-free graphs due to Theorem 2. Let G be an H'-free graph. Add a dominating vertex to G. The new graph G' has diameter 2 and is H-free. Moreover, G is k-colourable if and only if G' is $(k + 1)$-colourable.
2. Let $H \neq K_{1,3}$ have a dominating vertex u such that $H - u \subseteq_i P_1 + P_3$ or $H - u \subseteq_i P_4$. Then H cannot be a forest, as in that case H would be in $\{P_1, P_2, P_3, K_{1,3}\}$. Hence, H has an induced cycle C_r for some $r \geq 3$. If $r = 3$, then 3-Colouring is NP-complete for H-free graphs of diameter 3, as it is so for C_3-free graphs of diameter 3 due to Theorem 5. If $r \geq 4$, then Colouring is NP-complete even for H-free graphs of diameter 2, as it is so for C_r-free graphs of diameter 2 due to **1**. ◀

It is a natural question whether we can extend Theorem 9 to H-free graphs of diameter d, where H is a slightly larger tree than a star. The first interesting case is where H is an ℓ-subdivided star $K_{1,r}^{\ell}$ for some integer $\ell \geq 1$ and $r \geq 3$. We prove a number of results for various values of d,k,ℓ. For one of our proofs and also for the proof of our next result we need the following theorem.

▶ **Theorem 11.** 3-Colouring *can be solved in polynomial time for C_5-free graphs of diameter at most* 2.

Proof. If G is bipartite, then G is 3-colourable. If G contains a K_4, then G is not 3-colourable. We check these properties in polynomial time, and from now on we assume that G is K_4-free and non-bipartite. The latter implies that G must have an odd induced cycle C_r for some odd integer r. As G has diameter 2, we find that $r \leq 5$ due to Lemma 8. As G is C_5-free, it follows that $r = 3$.

Let C be a triangle in G. We write $N_0 = V(C) = \{x_1, x_2, x_3\}$, $N_1 = N(V(C))$ and $N_2 = V(G) \setminus (N_0 \cup N_1)$. As G has diameter 2, for every $i \in \{1, 2, 3\}$, it holds that every vertex in N_2 has a neighbour in N_1 that is adjacent to x_i.

We let T consist of all vertices of N_2 that have a neighbour in N_1 that is adjacent to exactly two vertices of N_0. We claim that $N_2 = T$. In order to see this, let $u \in N_2$. If u has a neighbour $y \in N_1$ adjacent to every x_i, then G contains a K_4, a contradiction. Hence, u must have three distinct neighbour y_1, y_2, y_3, such that for $i \in \{1, 2, 3\}$, it holds that $N(y_i) \cap N_0 = \{x_i\}$. If $\{y_1, y_2, y_3\}$ is a clique, then G has a K_4 on vertices u, y_1, y_2, y_3, a contradiction. Hence, we may assume without loss of generality that y_1 and y_2 are non-adjacent. However, then $\{u, y_1, x_1, x_2, y_2\}$ induces a C_5 in G, another contradiction. We conclude that $T = N_2$.

If G has a 3-colouring c, then we may assume without loss of generality that $c(x_i) = i$ for $i \in \{1, 2, 3\}$. Hence, our algorithm assigns colours 1, 2, 3 to x_1, x_2, x_3, respectively. This reduces the list of admissible colours of the vertices of N_1 by at least one colour. In particular, vertices in N_1 that have two neighbours in N_0 can be coloured with only one colour. Our algorithm assigns this colour to such vertices. This means that any of their neighbours in $T = N_2$ can be coloured with at most two colours. So, after propagation, we have obtained either two adjacent vertices that are coloured alike, in which case G is not 3-colourable, or we have constructed an instance of 2-List Colouring. We can solve such an instance in linear time due to Theorem 6. ◀

We are now ready to state our results for $K_{1,r}^{\ell}$, where we exclude the cases that are tractable in general, namely where $d = 1$, or $k \leq 2$, or $r \leq 2$ (the latter case corresponds to the case where $H = K_{1,2}^{+} = P_4$, so we can use Theorem 2). Note that for $k \geq 4$ all interesting cases are NP-complete, whereas for $k = 3$ the situation is less clear.

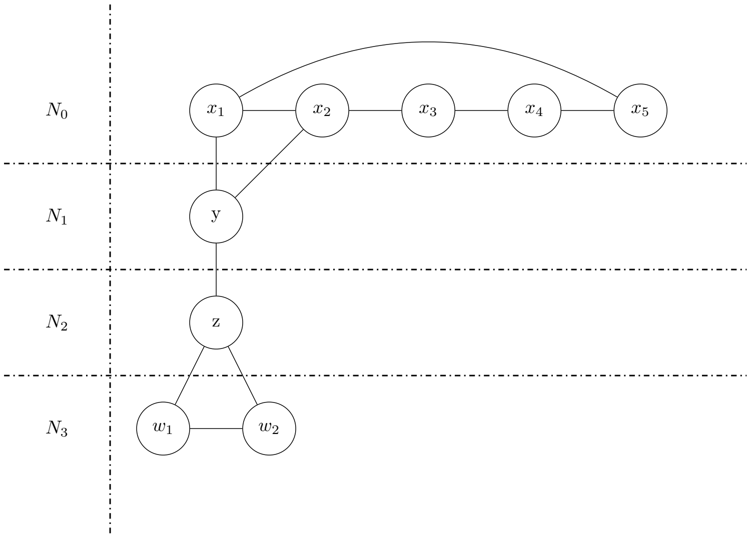


Figure 3 An example of a decomposition of a chair-free graph of diameter 3 into sets $N_0, \ldots, N_3$ where $p = 5$ and $y \in N_1$ has two “descendants” in N_3. To prevent an induced chair, y must be adjacent to exactly two (adjacent) vertices of N_0, and w_1 and w_2 must be adjacent to each other.

▶ **Theorem 12.** *Let d, k, ℓ, r be four integers with $d \geq 2$, $k \geq 3$, $\ell \geq 1$ and $r \geq 3$. Then k-Colouring for $K_{1,r}^{\ell}$-free graphs of diameter at most d is:*

1. *polynomial-time solvable if $d \geq 2$, $k = 3$, $\ell = 1$ and $r = 3$*
2. *polynomial-time solvable if $d = 2$, $k = 3$, $\ell = 2$ and $r \geq 3$*
3. *NP-complete if $d \geq 4$, $k = 3$, $\ell \geq 3$ and $r \geq 4$*
4. *NP-complete if $d \geq 2$, $k \geq 4$, $\ell \geq 1$ and $r \geq 3$.*

Proof.

1. Recall that $K_{1,3}^{1}$ is the chair $S_{1,1,2}$. Let G be a chair-free graph of diameter d. If G is a tree, then G is even 2-colourable. We check in $O(n^4)$ time if G has a K_4. If so, then G is not 3-colourable. From now on we assume that G is not a tree and that G is K_4-free. As G is not a tree and G is connected, G contains an induced cycle of length at most $2d + 1$ by Lemma 8. We can find a largest induced cycle C of length at most $2d + 1$ in $O(n^{2d+1})$ time. Let $|V(C)| = p$. We write $N_0 = V(C) = \{x_1, x_2, \ldots, x_p\}$ and for $i \geq 1$, $N_i = N(N_{i-1}) \setminus N_{i-2}$. So the sets N_i partition $V(G)$, and the distance of a vertex $u \in N_i$ to N_0 is i.

Case 1. $4 \leq p \leq 2d + 1$.

This case is illustrated in Figure 3. We consider every possible 3-colouring of C. Let c be such a 3-colouring. Every vertex with two differently coloured neighbours can only be coloured with one remaining colour. We assign this unique colour to such a vertex and apply this rule as long as possible. This takes polynomial time. The remaining vertices have a list of admissible colours that either consists of two or three colours, and vertices in the latter case belong to $V(G) \setminus (N_0 \cup N_1)$ (as $N(N_0) = N_1$).

If $N_2 = \emptyset$, then $V(G) = N_0 \cup N_1$. Then, we obtained an instance of 2-List Colouring, which we can solve in linear time due to Theorem 6. Now assume that $N_2 \neq \emptyset$. Let $z \in N_2$. Then z has a neighbour $y \in N_1$, which in turn has a neighbour $x \in N_0$. If y is adjacent to neither neighbour of x on N_0, then z, y, x and these two neighbours induce a chair in G, a contradiction. Hence, y must be adjacent to at least one neighbour of x on N_0, meaning that y must have received a colour by our algorithm. Consequently, z must have a list of admissible colours of size at most 2.

From the above we deduce that every vertex in N_2 has only two available colours in its list. We now consider the vertices of N_3. Let $z' \in N_3$. Then z' has a neighbour $z \in N_2$, which in turn has a neighbour $y \in N_1$, which in turn has a neighbour $x \in N_0$, say $x = x_1$. If y has two non-adjacent neighbours in N_0, then z', z, y and these two non-adjacent neighbours of y induce a chair in G, a contradiction. Combined with the fact deduced above, we conclude that y must have exactly two neighbours in N_0 and these two neighbours must be adjacent, say x_2 is the other neighbour of y in N_0.

Suppose x_1 and x_2 are both adjacent to a vertex $y' \in N_1 \setminus \{y\}$ that is adjacent to a vertex in N_2 that has a neighbour in N_3. Then, just as in the case of vertex y, the two vertices x_1 and x_2 are the only two neighbours of y' in N_0. If y and y' are not adjacent, this means that x_2, x_3, x_4, y, y' induce a chair in G, a contradiction. Hence y and y' must be adjacent. However, then x_1, x_2, y, y' form a K_4, a contradiction. This means that every pair of adjacent vertices of N_0 can have at most one common neighbour in N_1 that is adjacent to a vertex in N_2 with a neighbour in N_3. We already deduced that every vertex of N_1 with a "descendant" in N_3 has exactly two neighbours in N_0, which are adjacent. Hence, we conclude that the number of such vertices of N_1 is at most p.

We now observe that for $i \geq 2$, every vertex in N_i has at most two neighbours in N_{i+1}. This can be seen as follows. If $v \in N_i$ has two non-adjacent neighbours w_1, w_2 in N_{i+1}, then we pick a neighbour u of v in N_{i-1}, which has a neighbour t in N_{i-2}. Then v, u, t, w_1, w_2 induce a chair in G, a contradiction. Hence , the neighbourhood of every vertex in N_i in N_{i+1} is a clique, which must have size at most 2 due to the K_4-freeness of G. As the number of vertices in N_1 with a "descendant" in N_3 is at most p, this means that there are at most $2^{i-1}p$ vertices in N_i with a neighbour in N_{i+1}. Therefore the total number of vertices not belonging to any of the sets N_0, N_1 or N_2 is at most $\sum_{i=3}^{d} 2^{i-1}p$. This means the total number of vertices not belonging to N_1 or N_2 is at most $\beta(d) = \sum_{i=3}^{d} 2^{i-1}p + p \leq \sum_{i=3}^{d} 2^{i-1}(2d+1) + 2d + 1$. Let T_c be this set. We consider every possible 3-colouring of $G[T_c]$. As we already deduced that the vertices in $N_1 \cup N_2$ have a list of size at most 2, for each case we obtain an instance of 2-List Colouring, which we can solve in linear time due to Theorem 6. As the total number of instances we need to consider is at most $3^p \times 3^{\beta(d)} \leq 3^{2d+1} \times 3^{\beta(d)}$, our algorithm runs in polynomial time.

Case 2. $p = 3$.

As p was the size of a largest induced cycle of length at most $2d+1$ and $2d+1 \geq 5$, we find that G is C_4-free. As G is K_4-free, each vertex of N_1 is adjacent to at most two vertices of N_0. If a vertex $x \in N_0$ has two independent private neighbours u and v in N_1 with respect to N_0, then every neighbour w of u in N_2 must also be a neighbour of v and vice versa, since G is chair-free. However, this is not possible, as x, u, w, v induce a C_4.

We conclude that u and v must be adjacent. Therefore, as G is K_4-free, every vertex of N_0 has at most two private neighbours in N_1, with respect to N_0, that have a neighbour in N_2.

By the same arguments as above we deduce that every two vertices of N_0 have at most one common neighbour in N_1 that is adjacent to a vertex in N_2. Combined with the above, we find that there at most $6+3=9$ vertices in N_1 that have a neighbour in N_2. If a vertex in N_1 has two independent neighbours in N_2, then G contains an induced chair, which is not possible. Hence the neighbourhood of a vertex in N_1 in N_2 is a clique, which has size at most 2 due to the K_4-freeness of G. We conclude that $|N_2| \leq 9 \times 2 = 18$. Similarly, every vertex in N_i for $i \geq 3$ has at most two neighbours in N_{i+1}. Therefore the number of vertices in N_i for $i \geq 3$ is at most $18 \times 2^{i-2}$. This means that the total number of vertices outside $N_0 \cup N_1 \cup N_2$ is at most $\beta(d) = \sum_{i=3}^{d} 18 \times 2^{i-2}$. Let T be this set. We consider every possible 3-colouring of $G[T]$ and every possible 3-colouring of C. For each case we obtain an instance of 2-List Colouring, which we can solve in linear time due to Theorem 6. As the total number of instances we need to consider is at most $3^d \times 3^{\beta(d)}$, our algorithm runs in polynomial time.

2. Let G be a $K_{1,r}^2$-free graph of diameter at most 2. We first check in $O(n^4)$ time if G is K_4-free. If not, then G is not 3-colourable. We then check in $O(n^5)$ time if G has an induced C_5. If G is C_5-free, then we use Theorem 11. From now on, suppose that G is K_4-free and that G contains an induced cycle C of length 5, say on vertices $x_1, \ldots, x_5$ in that order. We write $N_0 = V(C) = \{x_1, \ldots, x_5\}$, $N_1 = N(V(C))$ and $N_2 = V(G) \setminus (N_0 \cup N_1)$. Let N_2' be the set of vertices in N_2 that are adjacent to some vertex in N_1 that is a private neighbour of some vertex in N_0 with respect to N_0. As G is K_4-free, the private neighbourhood $P(x_i)$ of each vertex $x_i \in N_0$ with respect to N_0 does not contain a clique of size 3. Moreover, if $P(x_i)$ contains an independent set I of size $r-1$ for some $i \in \{1, \ldots, 5\}$, then $I \cup \{x_i, x_{i+1}, x_{i+2}, x_{i+3}\}$ induces a $K_{1,r}^2$, which is not possible. Now let $v \in P(x_i)$ for some $i \in \{1, \ldots 5\}$, say $i = 1$. As G is K_4-free, the set $N(v) \cap N_2$ does not contain a clique of size 3. Moreover, if $N(v) \cap N_2$ contains an independent set I' of size $r-1$, then $I' \cup \{v, x_1, x_2, x_3, \}$ induces a $K_{1,r}^2$, which is not possible. Hence, $|N(v) \cap N_2| \leq R(3, r-1)$ by Ramsey's Theorem. We conclude that $|N_2'| \leq 5R(3, r-1)^2$.
We now consider all possible 3-colourings of C. Let c be such a 3-colouring. We assume without loss of generality that $c(x_1) = c(x_3) = 1$, $c(x_2) = c(x_4) = 2$ and $c(x_5) = 3$. Moreover, every vertex that has two differently coloured neighbours can only be coloured with one remaining colour. We assign this unique colour to such a vertex and apply this rule as long as possible. This takes polynomial time. The remaining vertices have a list of admissible colours that either consists of two or three colours, and vertices in the latter case must belong to N_2 (as $N(N_0) = N_1$).
Let T_c be the set of vertices in N_2 that still have a list of size 3. We will prove that $T_c \subseteq N_2'$. Let $u \in T_c$. As G has diameter 2, we find that u has a neighbour v adjacent to x_5. Then v cannot be adjacent to any of $x_1, \ldots, x_4$, as otherwise v would have a unique colour and u would not be in T_c. Hence, v is a private neighbour of x_5 with respect to N_0. We conclude that all vertices in T_c belong to N_2', which implies that $|T_c| \leq |N_2'| \leq 5R(3, r-1)^2$.
We now consider every possible 3-colouring of $G[T_c]$. Then all uncoloured vertices have a list of size at most 2. In other words, we created an instance of 2-List Colouring, which we solve in linear time using Theorem 6. As the number of 3-colourings of C is at most 3^5 and for each 3-colouring c of C the number of 3-colourings of $G[T_c]$ is at most $3^{5R(3,r-1)^2}$, the total running time of our algorithm is polynomial.

3. We consider the standard reduction from the NP-complete problem NAE 3-SAT [29], where each variable appears in at most three clauses and each literal appears in at most two. Given a CNF formula ϕ, we construct the graph G as follows:
   - Add a vertex v_{x_i} for each literal x_i.
   - Add an edge between each literal and its negation.
   - Add a vertex z adjacent to every literal vertex.
   - For each clause C_i add a triangle T_i with vertices $c_{i_1}, c_{i_2}, c_{i_3}$.
   - Fix an arbitrary order of the literals of C_i, $x_{i_1}, x_{i_2}, x_{i_3}$ and add an edge $x_{i_j}c_{i_j}$.

   Given a 3-colouring of G, assume z is assigned colour 1. Then each literal vertex is assigned either colour 2 or colour 3. If, for some clause C_i, the vertices x_{i_1}, x_{i_2} and $x_{i,3}$ are all assigned the same colour, then T_i cannot be coloured. Therefore, if we set literals whose vertices are coloured with colour 2 to be true and those coloured with colour 3 to be false, each clause must contain at least one true literal and at least one false literal.
   If ϕ is satisfiable then we can colour vertex z with colour 1, each true literal with colour 2 and each false literal with colour 3. Then, since each clause has at least one true literal and at least one false literal, each triangle has neighbours in two different colours. This implies that each triangle is 3-colourable. Therefore G is 3-colourable if and only if ϕ is satisfiable.
   We next show that G has diameter at most 4. First note that any literal vertex is adjacent to z and any clause vertex is adjacent to some literal vertex so any vertex is at distance at most 2 from z. Therefore any two vertices are at distance at most 4.
   Finally we show that G is $K_{1,4}^3$-free. Any literal vertex has degree at most 4 since it appears in at most two clauses. However it has at most 3 independent neighbours since its negation is adjacent to z. Each clause vertex has at most 3 neighbours so the only vertex with four independent neighbours is d. The longest induced path including z has length at most 4 since any such path contains at most one literal and at most two vertices of any triangle. Therefore G is $K_{1,4}^3$-free.
4. This follows from Theorem 3. Let $k^* \geq 3$. We take a claw-free graph G and add a dominating vertex to it. The new graph G' has diameter at most 2 and is $K_{1,3}^1$-free. Let $k = k^* + 1 \geq 4$. Then G is k^*-colourable if and only if G' is k-colourable. ◀

Subdividing two edges of the claw yields another interesting case, namely where $H = S_{1,2,2}$. For $k \geq 4$, Theorem 12 tells us that k-Colouring is NP-complete for $S_{1,2,2}$-free graphs of diameter 2. For $k = 3$, we could only prove polynomial-time solvability if $d = 2$.

▶ **Theorem 13.** 3-Colouring *can be solved in polynomial time for $S_{1,2,2}$-free graphs of diameter at most* 2.

Proof. Let G be an $S_{1,2,2}$-free graph of diameter at most 2. We first check in $O(n^5)$ time if G has an induced C_5. If G is C_5-free, then we use Theorem 11. Suppose G contains an induced cycle C of length 5, say on vertices $x_1, \ldots, x_5$ in that order. We write $N_0 = V(C) = \{x_1, \ldots, x_5\}$, $N_1 = N(V(C))$ and $N_2 = V(G) \setminus (N_0 \cup N_1)$. As G has diameter 2, for every $i \in \{1, 2, 3\}$, every vertex in N_2 has a neighbour in N_1 that is adjacent to x_i.

We let T consist of all vertices of N_2 that have a neighbour in N_1 that is adjacent to two adjacent vertices of N_0. So the colour of any vertex of T will be fixed in any 3-colouring after colouring the five vertices of N_0. We claim that $N_2 = T$. In order to see this, let $u \in N_2$. As G has diameter 2, we find that u must have a neighbour $v \in N_1$ adjacent to a vertex of N_0, say x_1. Then v is not adjacent to x_5 or x_2. If v is not adjacent to x_3 either, then the vertices x_1, x_5, x_2, x_3, v, u induce a $S_{1,2,2}$ with center x_1, a contradiction. So v must be adjacent to x_3, meaning v is not adjacent to x_4. However, now x_3, x_2, x_4, x_5, v, u induce a $S_{1,2,2}$ with center x_3, another contradiction.

We now "guess" the 3-colouring of C by considering all 3^5 possibilities if necessary. We then proceed as in the proof of Theorem 11. That is, we observe that every vertex of N_1 can only be coloured with two possible colours and that after propagation, every uncoloured vertex of N_2 can only be coloured with two possible colours as well (as $T = N_2$). Then it remains to solve an instance of 2-List Colouring, which takes linear time by Theorem 6. As we need to do this at most 3^5 times, the total running time of our algorithm is polynomial. ◀

4 Graphs of Bounded Diameter and Girth

In this section we will examine the trade-offs for k-Colouring between diameter and girth. Recall that Mertzios and Spirakis [24] proved that 3-Colouring is NP-complete for graphs of diameter 3 and girth 4 (Theorem 5). We extend their result in our next theorem, partially displayed in Figure 2. This theorem shows that there is still a large gap for which we do not know the computational complexity of 3-Colouring for graphs of diameter d and girth g.

▶ **Theorem 14.** *Let d, g, k be three integers with $d \geq 2$, $g \geq 3$ and $k \geq 3$. Then k-Colouring for graphs of diameter at most d and girth at least g is*

1. *polynomial-time solvable if $g \geq 2d + 1$*
2. *NP-complete if $d = 3$ and $g \leq 4$ and $k = 3$*
3. *NP-complete if $4p \leq d \leq 4p + 3$ and $g \leq 4p + 2$ for some integer $p \geq 1$ and $k = 3$.*

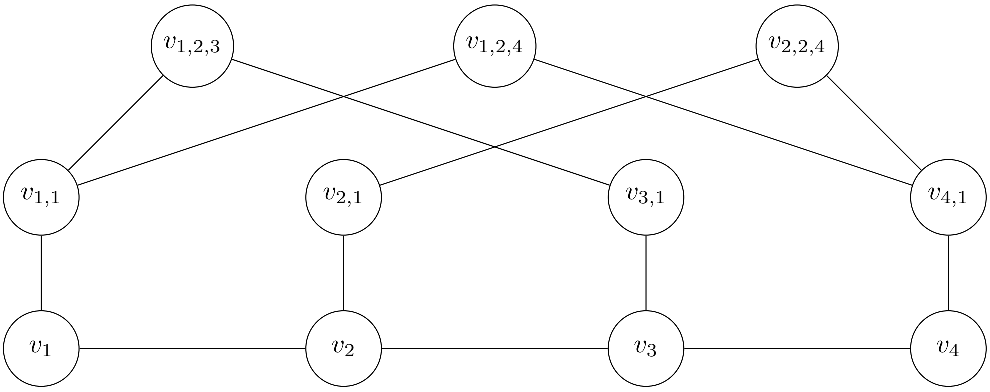


Figure 4 An example of a graph G', constructed in the proof of Theorem 14(3), for $p = 1$.

Proof.

1. This case follows from Theorem 7.
2. This case is Theorem 5 (proven in [24]).
3. We reduce 3-Colouring for graphs of girth at least $8p - 3$, which is NP-complete by Theorem 4, to 3-Colouring for graphs of diameter at most $4p$ and girth at least $4p + 2$. Construct the graph G' as follows (see Figure 4 for an example):
   - label the vertices of G v_1 to v_n;
   - for each vertex of G, add a new neighbour $v_{i,1}$;
   - for every two vertices v_i and v_j such that $\text{dist}(v_i, v_j) > l = 2p - 1$ add new vertices to form the path $v_{i,1}v_{i,2,j}...v_{i,p+1,j}v_{j,p,i}...v_{j,1}$.

First we show that G' has diameter at most $4p$. For any two vertices v_i and v_j of G either $\text{dist}(v_i, v_j) \leq l$ or we have the path $v_{i,1}v_{i,2,j}...v_{i,p+1,j}v_{j,p,i}...v_{j,1}$ and $\text{dist}(v_i, v_j) \leq 2p+2$. Similarly, $\text{dist}(v_i, v_{j,1}) \leq 2p+1$ and $\text{dist}(v_{i,1}, v_{j,1}) \leq 2p+1$. Now consider two vertices $v_{a,r,b}$ and $v_{c,q,d}$ for $2 \leq r \leq p+1$, $2 \leq q \leq p+1$. If $\text{dist}(v_a, v_c) \leq l$ then $\text{dist}(v_{a,r,b}, v_{c,q,d}) \leq r+q+l \leq (p+1)+(p+1)+(2p-1) \leq 4p+1$. Otherwise we have the path $v_{a,r,b}..v_{a,1}v_{a,2,c}...v_{a,p+1,c}v_{c,p,a}...v_{c,1}v_{c,2,d}...v_{c,q,d}$. This gives $\text{dist}(v_{a,r,b}, v_{c,q,d}) \leq (r-1)+p+p+(q-1) \leq 4p$. In fact, if $\text{dist}(v_{a,r,b}, v_{c,q,d}) = 4p+1$, then we must have $r = q = p+1$ and $\text{dist}(v_a, v_c) = \text{dist}(v_a, v_d) = \text{dist}(v_b, v_c) = \text{dist}(v_b, v_d) = 2p-1$. In this case we have two paths of length at most $4p-2$ between v_a and v_b, one containing v_c and the other containing v_d. These paths must be distinct since the existence of the vertex $v_{c,p+1,d}$ implies that $\text{dist}(v_c, v_d) > 2p-1$. Therefore we have a cycle in G of length at most $8p-4$ which contradicts the assumption that G has girth at least $8p-3$. This implies that the diameter of G' is at most $4p$.

Since G has girth at least $8p-3$, every cycle in G' of length less than $4p+2$ must contain at least one vertex of $V(G') \setminus V(G)$. Since all the vertices of $V(G') \setminus V(G)$ except the vertices $v_{i,1}$ have degree 2, any such cycle C must contain the path $v_{i,1}..v_{i,p+1,j}...v_j$ for some v_i, v_j at distance greater than l. This path has length $2p+1$. If C contains $v_{i,2,m}$ for some m different from j then it contains the path $v_{i,2,m}...v_{m,1}$ and has length at least $4p+2$. Similarly, this is the case if C contains $v_{j,2,m}$ for m different from i. Otherwise C contains v_i and v_j which are at distance at least l and has length at least $(2p+1)+2+(2p-1) = 4p+2$.

Finally, we show that G is 3-colourable if and only if G' is 3-colourable. The latter holds if and only if the subgraph G'' of G' induced by $V(G) \cup \{v_{i,1} \mid 1 \leq i \leq n\}$ is 3-colourable, since every other vertex of G' has degree 2. The graph G is 3-colourable if and only if G'' is 3-colourable, since G is an induced subgraph of G'' and each vertex of $V(G'') \setminus V(G)$ has degree 1. Therefore, G is 3-colourable if and only if G' is 3-colourable. ◀

5 Conclusions

We proved a number of new results for Colouring and k-Colouring for polyad-free graphs of bounded diameter and for graphs of bounded diameter and girth. In particular we identified and narrowed a number of complexity gaps. This leads us to some natural open problems. Our first two open problems follow from Theorem 10. The third open problem comes from Theorem 12; note that $K_{1,3}^2 = S_{1,1,3}$. Our fourth open problem stems from Theorem 13. Recall that determining the complexity of 3-Colouring for graphs of diameter 2 is still wide open. This question is covered by the fifth open problem.

▶ **Open Problem 1.** *Does there exist an integer d such that* Colouring *is* NP*-complete for $K_{1,3}$-free graphs of diameter d?*

▶ **Open Problem 2.** *What is the complexity of* Colouring *for C_3-free graphs of diameter 2, or equivalently, graphs of diameter 2 and girth 4?*

▶ **Open Problem 3.** *What are the complexities of* 3-Colouring *for $K_{1,4}^1$-free graphs of diameter 3 and for $K_{1,3}^2$-free graphs of diameter 3?*

▶ **Open Problem 4.** *Do there exist integers d, h, i, j such that* 3-Colouring *is* NP*-complete for $S_{h,i,j}$-free graphs of diameter d?*

▶ **Open Problem 5.** *What is the complexity of the open cases in Figure 2 and in particular of* 3-Colouring *for graphs of diameter 2 and for graphs of diameter 2 and girth 4?*

References

1 Noga Alon. Restricted colorings of graphs. *Surveys in combinatorics, London Mathematical Society Lecture Note Series*, 187:1–33, 1993.

2 Flavia Bonomo, Maria Chudnovsky, Peter Maceli, Oliver Schaudt, Maya Stein, and Mingxian Zhong. Three-Coloring and List Three-Coloring of Graphs Without Induced Paths on Seven Vertices. *Combinatorica*, 38(4):779–801, 2018.

3 Miroslav Chlebík and Janka Chlebíková. Hard coloring problems in low degree planar bipartite graphs. *Discrete Applied Mathematics*, 154(14):1960–1965, 2006.

4 Maria Chudnovsky. Coloring graphs with forbidden induced subgraphs. *Proc. ICM 2014*, IV:291–302, 2014.

5 Maria Chudnovsky, Shenwei Huang, Sophie Spirkl, and Mingxian Zhong. List-three-coloring graphs with no induced $P_6 + rP_3$. *CoRR*, abs/1806.11196, 2018.

6 Maria Chudnovsky, Sophie Spirkl, and Mingxian Zhong. Four-coloring P_6-free graphs. *Proc. SODA 2019*, pages 1239–1256, 2019.

7 Konrad Kazimierz Dabrowski, François Dross, Matthew Johnson, and Daniël Paulusma. Filling the Complexity Gaps for Colouring Planar and Bounded Degree Graphs. *Journal of Graph Theory*, to appear, 2015.

8 R. Mark Damerell. On Moore Graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 74:227–236, 1973.

9 Keith Edwards. The complexity of colouring problems on dense graphs. *TCS*, 43:337–343, 1986.

10 Thomas Emden-Weinert, Stefan Hougardy, and Bernd Kreuter. Uniquely Colourable Graphs and the Hardness of Colouring Graphs of Large Girth. *Combinatorics, Probability and Computing*, 7(04):375–386, 1998.

11 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.

12 Carla Groenland, Karolina Okrasa, Pawel Rzążewski, Alex Scott, Paul Seymour, and Sophie Spirkl. H-colouring P_t-free graphs in subexponential time. *CoRR*, 1803.05396, 2018.

13 Martin Grötschel, László Lovász, and Alexander Schrijver. Polynomial Algorithms for Perfect Graphs. *Annals of Discrete Mathematics*, 21:325–356, 1984.

14 Chính T. Hoàng, Marcin Kamiński, Vadim V. Lozin, Joe Sawada, and Xiao Shu. Deciding k-Colorability of P_5-Free Graphs in Polynomial Time. *Algorithmica*, 57(1):74–81, 2010.

15 Alan J. Hoffman and Robert R. Singleton. On Moore graphs with diameter 2 and 3. *IBM Journal of Research and Development*, 5:497–504, 1960.

16 Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981.

17 Shenwei Huang. Improved complexity results on k-coloring P_t-free graphs. *European Journal of Combinatorics*, 51:336–346, 2016.

18 Tommy R. Jensen and Bjarne Toft. *Graph coloring problems*. John Wiley & Sons, 1995.

19 Tereza Klimošová, Josef Malík, Tomáš Masařík, Jana Novotná, Daniël Paulusma, and Veronika Slívová. Colouring $(P_r + P_s)$-Free Graphs. *Proc. ISAAC 2018, LIPIcs*, 123:5:1–5:13, 2018.

20 Daniel Král', Jan Kratochvíl, Zsolt Tuza, and Gerhard J. Woeginger. Complexity of Coloring Graphs without Forbidden Induced Subgraphs. *Proceedings of WG 2001, LNCS*, 2204:254–262, 2001.

21 Jan Kratochvíl, Zsolt Tuza, and Margit Voigt. New trends in the theory of graph colorings: choosability and list coloring. *Proc. DIMATIA-DIMACS Conference*, 49:183–197, 1999.

22 Daniel Leven and Zvi Galil. NP completeness of finding the chromatic index of regular graphs. *Journal of Algorithms*, 4(1):35–44, 1983.

23 László Lovász. Coverings and coloring of hypergraphs. *Congr. Numer.*, VIII:3–12, 1973.

24 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.

25 Michael Molloy and Bruce A. Reed. Colouring graphs when the number of colours is almost the maximum degree. *Journal of Combinatorial Theory, Series B*, 109:134–195, 2014.

26 Daniël Paulusma. Open problems on graph coloring for special graph classes. *Proc. WG 2015, LNCS*, 9224:16–30, 2015.

27 Frank P. Ramsey. On a Problem of Formal Logic. *Proceedings of the London Mathematical Society*, s2-30(1):264–286, 1930.

28 Bert Randerath and Ingo Schiermeyer. Vertex Colouring and Forbidden Subgraphs – A Survey. *Graphs and Combinatorics*, 20(1):1–40, 2004.

29 Thomas J. Schaefer. The Complexity of Satisfiability Problems. *STOC*, pages 216–226, 1978.

30 Robert R. Singleton. There is no irregular Moore Graph. *Amererican Mathematical Monthly*, 75:42–43, 1968.

31 Zsolt Tuza. Graph colorings with local constraints - a survey. *Discussiones Mathematicae Graph Theory*, 17(2):161–228, 1997.

Distance Labeling Schemes for Cube-Free Median Graphs

Victor Chepoi
Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
victor.chepoi@lis-lab.fr

Arnaud Labourel
Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
arnaud.labourel@lis-lab.fr

Sébastien Ratel
Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
sebastien.ratel@lis-lab.fr

Abstract

Distance labeling schemes are schemes that label the vertices of a graph with short labels in such a way that the distance between any two vertices u and v can be determined efficiently by merely inspecting the labels of u and v, without using any other information. One of the important problems is finding natural classes of graphs admitting distance labeling schemes with labels of polylogarithmic size. In this paper, we show that the class of cube-free median graphs on n nodes enjoys distance labeling scheme with labels of $O(\log^3 n)$ bits.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases median graphs, labeling schemes, distributed distance computation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.15

Related Version A full version of the paper is available at `https://arxiv.org/abs/1809.10508`.

Funding The work on this paper was supported by ANR project DISTANCIA (ANR-17-CE40-0015).

1 Introduction

Classical network representations are usually global in nature. In order to derive a useful piece of information, one must access to a global data structure representing the entire network even if the needed information only concerns few nodes. Nowadays, with networks getting bigger and bigger, the need for locality is more important than ever. Indeed, in several cases, global representations are impractical and network representation must be distributed. The notion of (distributed) labeling scheme has been introduced [12, 32, 38, 39, 27] in order to meet this need. A *(distributed) labeling scheme* is a scheme maintaining global information on a network using local data structures (or labels) assigned to nodes of the network. Their goal is to locally store some useful information about the network in order to answer a specific query concerning a pair of nodes by only inspecting the labels of the two nodes. Motivation for such localized data structure in distributed computing is surveyed and widely discussed in [38]. The predefined queries can be of various types such as distance, adjacency, or routing. The quality of a labeling scheme is measured by the size of the labels of nodes and the time required to answer queries. Trees with n vertices admit adjacency and routing labeling schemes with size of labels and query time $O(\log n)$[1] and distance labeling schemes with size of labels and query time $O(\log^2 n)$, and this is asymptotically optimal. Finding natural classes of graphs admitting distance labeling schemes with labels of polylogarithmic size is an important and challenging problem.

[1] All logarithms in this paper are in base 2.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 15; pp. 15:1–15:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A connected graph G is *median* if any triplet of vertices x, y, z contains a unique vertex simultaneously lying on shortest (x, y)-, (y, z)-, and (z, x)-paths. Median graphs constitute the most important class in metric graph theory [5]. This importance is explained by the bijections between median graphs and discrete structures arising and playing important roles in completely different areas of research in mathematics and theoretical computer science: in fact, median graphs, 1-skeletons of CAT(0) cube complexes from geometric group theory [30, 41], domains of event structures from concurrency [44], median algebras from universal algebra [7], and solution sets of 2-SAT formulae from complexity theory [36, 42] are all the same. In this paper, we design a distance labeling scheme for median graphs containing no cubes. In our scheme, the labels have $O(\log^3 n)$ bits and $O(1)$ query time. Our constant query time assumes the standard word-RAM model with word size $\Omega(\log n)$.

We continue with the idea of the labeling scheme. Let $G = (V, E)$ be a cube-free median graph with n vertices. First, the algorithm computes a median (centroid) vertex m of G. and the star $\mathrm{St}(m)$ of m (the union of all edges and squares of G incident to m). The star $\mathrm{St}(m)$ is gated, i.e., each vertex of G has an unique projection (nearest vertex) in $\mathrm{St}(m)$. Therefore, with respect to the projection function, the vertex-set of G is partitioned into fibers: the fiber $F(x)$ of $x \in \mathrm{St}(m)$ consists of all vertices $v \in V$ having x as the projection in $\mathrm{St}(m)$. Since m is a median of G, each fiber contains at most $\frac{n}{2}$ vertices. The fibers are also gated and are classified into panels and cones depending to the distance between their projections and m (one for panels and two for cones). Each cone has at most two neighboring panels however a panel may have an unbounded number of neighboring cones. Given two arbitrary vertices u and v of G, we show that $d_G(u, v) = d_G(u, m) + d_G(m, v)$ for all locations of u and v in the fibers of $\mathrm{St}(m)$ except the cases when u and v belong to neighboring cones and panels, or u and v belong to two cones neighboring the same panel, or u and v belong to the same fiber. If $d_G(u, v) = d_G(u, m) + d_G(m, v)$, then $d_G(u, v)$ can be retrieved by keeping $d_G(u, m)$ in the label of u and $d_G(v, m)$ in the label of v. If u and v belong to the same fiber $F(x)$, the computation of $d_G(u, v)$ is done by recursively partitioning the cube-free median graph $F(x)$ at a later stage of the recursion. In the two other cases, we show that $d_G(u, v)$ can be retrieved by keeping in the labels of vertices in all cones the distances to their projections on the two neighboring panels. It turns out (and this is the main technical contribution of the paper), that for each panel $F(x)$, the union of all projections of vertices from neighboring cones on $F(x)$ is included in an isometric tree of G and that the vertices of the panel $F(x)$ contain one or two projections in this tree. All such outward and inward projections are kept in the labels of respective vertices. Therefore, one can use distance labeling schemes for trees to deal with vertices u and v lying in neighboring fibers or in cones having a common neighboring panel. Consequently, the size of the label of a vertex u on each recursion level is $O(\log^2 n)$. Since the recursion depth is $O(\log n)$, the vertices of G have labels of size $O(\log^3 n)$. The distance $d_G(u, v)$ can be retrieved by finding the first time in the recursion when vertices u and v belong to different fibers of the partition. Consequently, the main result of the paper is the following theorem:

▶ **Theorem 1.** *There exists a distance labeling scheme that constructs in $O(n^2 \log n)$ time labels of size $O(\log^3 n)$ of the vertices of a cube-free median graph $G = (V, E)$. Given the labels of u and v of G, it computes in constant time the distance $d_G(u, v)$ between u and v.*

The remaining part of this note is organized in the following way. Section 2 introduces the notions used in this paper. In Section 3 we review the main results on distance labeling schemes and on median graphs. In Section 4 we recall or establish some properties of general median graphs used in our scheme. Section 5 presents the most important geometric and structural properties of cube-free median graphs, which are the essence of our distance scheme and which do not hold for general median graphs. Section 6 describes our distance labeling scheme for cube-free median graphs and proves Theorem 1. Due to page limits, the missing proofs and the pseudocodes are provided in the full version [22]. In the full version, we also describe a routing labeling scheme with similar performances.

2 Preliminaries

2.1 Basic notions

All graphs $G = (V, E)$ in this note are finite, undirected, simple, and connected. We will write $u \sim v$ if two vertices u and v are adjacent. The *distance* $d_G(u, v)$ between two vertices u and v is the length of a shortest (u, v)-path, and the *interval* $I(u, v) := \{x \in V : d_G(u, x) + d_G(x, v) = d_G(u, v)\}$ consists of all the vertices on shortest (u, v)–paths. A connected subgraph H of G is called *isometric* if $d_H(u, v) = d_G(u, v)$ for any two vertices u, v of H. A subgraph H of G is *gated* if for every vertex $v \notin V(H)$, there exists a vertex $v' \in V(H)$ such that for all $u \in V(H)$, $d_G(v, u) = d_G(v, v') + d_G(v', u)$ (v' is called the *gate* of v in H). For a vertex x of a gated subgraph H of G, the set $F(x) = \{v \in V : x \text{ is the gate of } v \text{ in } H\}$ is called the *fiber* of x with respect to H. The fibers $\{F(x) : x \in H\}$ define a partition of G. The *m-dimensional hypercube* Q_m has all subsets of $\{1, \ldots, m\}$ as the vertex-set and $A \sim B$ iff $|A \triangle B| = 1$.

A graph G is called *median* if the intersection $I(x, y) \cap I(y, z) \cap I(z, x)$ is a singleton for each triplet x, y, z of vertices; this unique intersection vertex is called the *median* of x, y, z. Median graphs are bipartite. Basic examples of median graphs are trees, hypercubes, rectangular grids, and Hasse diagrams of distributive lattices and of median semilattices [5]. The *star* $\mathrm{St}(z)$ of a vertex z of a median graph G is the union of all hypercubes of G containing z. The *dimension* $\dim(G)$ of a median graph G is the largest dimension of an hypercube subgraph of G. A *cube-free median graph* is a median graph G of dimension 2, see Figure 1 for illustrations. Even if cube-free median graphs are the skeletons of 2-dimensional CAT(0) cube complexes, their combinatorial structure is rather intricate. As an example, for $n, m \geq 5$, the Cartesian product $K_{1,n} \times K_{1,m}$ is a non-planar cube-free median graph. Moreover, for any n, one can construct a cube-free median graph containing K_n as a minor by gluing together $\binom{n}{2}$ grids of size $n \times n$ along a common horizontal side. Hence, this class is not a subset of any minor-closed graph family.

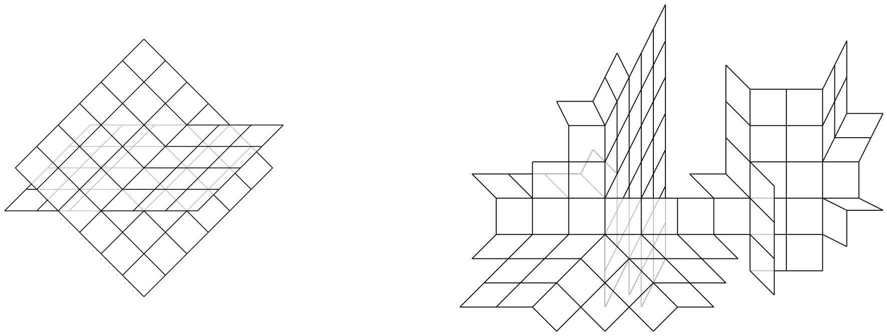


Figure 1 Cube-free median graphs.

2.2 Distance labeling schemes

A *labeling scheme* for a graph family $\mathcal{G}$ consists of an encoding function and a decoding function. These functions depend on the family $\mathcal{G}$ and on the type of queries: adjacency, distance, or routing queries. More formally, a *distance labeling scheme* on a graph family $\mathcal{G}$ consists of an *encoding function* $C_G : V(G) \to \{0,1\}^*$ that gives to every vertex of a graph G of $\mathcal{G}$ a label, and of a *decoding function* $D_G : \{0,1\}^* \times \{0,1\}^* \to \mathbb{N}$ that, given the labels of two vertices u and v of G, can compute efficiently the distance $d_G(u,v)$ between them.

3 Related work

3.1 Distance labeling schemes

Distance Labeling Schemes (DLS) have been introduced in a series of papers by Peleg et al. [38, 39, 27]. Before these works, some closely related notions already existed such as embeddings in a squashed cube [43] (equivalent to distance labeling schemes with labels of size $\log n$ times the dimension of the cube) or labeling schemes for adjacency requests [32]. One of the main results for DLS is that general graphs support distance labeling schemes with labels of size $O(n)$ bits [43, 27, 2]. This scheme is asymptotically optimal since $\Omega(n)$ bits labels are needed for general graphs. Another important result is that there exists a distance labeling scheme for the class of trees with $O(\log^2 n)$ bits labels [38, 3, 24]. Several classes of graphs containing trees also enjoy a distance labeling scheme with $O(\log^2 n)$ bit labels such as bounded tree-width graphs [27], distance-hereditary graphs [25], bounded clique-width graphs [23], and non-positively curved plane graphs [19]. A lower bound of $\Omega(\log^2 n)$ bits on the label length is known for trees [27, 3], implying that all the results mentioned above are optimal as well. Other families of graphs have been considered such as interval graphs, permutation graphs, and their generalizations [9, 26] for which an optimal bound of $\Theta(\log n)$ bits was given, and planar graphs for which there is a lower bound of $\Omega(n^{\frac{1}{3}})$ bits [27] and an upper bound of $O(\sqrt{n})$ bits [28].

3.2 Median graphs

Median graphs and related structures have an extensive literature; several surveys exist listing their numerous characterizations and properties [5, 33, 34]. These structures have been investigated in different contexts by quite a number of authors for more than half a century. In this subsection we briefly review the links between median graphs and CAT(0) cube complexes. We also recall some results, related to the subject of this paper, about the distance and shortest path problems in median graphs and CAT(0) cube complexes. For a survey of results on median graphs and their bijections with median algebras, median semilattices, CAT(0) cube complexes, and solution spaces of 2-SAT formulae, see [5]. For a comprehensive presentation of median graphs and CAT(0) cube complexes as domains of event structures, see the long version of [14].

It is not immediately clear from the definition, but median graphs are intimately related to hypercubes: median graphs can be obtained from hypercubes by amalgams and median graphs are themselves isometric subgraphs of hypercubes [8, 35]. Even more, median graphs are exactly the retracts of hypercubes [4]. Due to the abundance of hypercubes, to each median graph G one can associate a cube complex $X(G)$ obtained by replacing every hypercube of G by a solid unit cube. Then G can be recovered as the 1-skeleton of $X(G)$. The cube complex $X(G)$ can be endowed with several intrinsic metrics, in particular with the ℓ_2-metric. An important class of cube complexes studied in geometric group theory and combinatorics is the class of CAT(0) cube complexes. CAT(0) geodesic metric spaces are usually defined via the nonpositive curvature comparison axiom of Cartan–Alexandrov–Toponogov [13]. For cube complexes (and more generally for cell complexes) the CAT(0) property can be defined in a very simple and intuitive way by the property that *ℓ_2-geodesics between any two points are unique*. Gromov [30] gave a nice combinatorial characterization of CAT(0) cube complexes as *simply connected cube complexes with flag links*. It was also shown in [18, 40] that *median graphs are exactly the 1-skeletons of CAT(0) cube complexes.*

Previous characterizations can be used to show that several cube complexes arising in applications are CAT(0). Billera et al. [10] proved that the space of trees (encoding all tree topologies with a given set of leaves) is a CAT(0) cube complex. Abrams et al. [1, 29] considered the space of all possible positions of a reconfigurable system and showed that in many cases this state complex is CAT(0). Billera et al. [10] formulated the problem of computing the geodesic between two points in the space of trees. In the robotics literature, geodesics in state complexes correspond to the motion planning to get the robot from one position to another one with minimal power consumption. A polynomial-time algorithm for geodesic problem in the space of trees was provided in [37] and, very recently, [31] designed such an algorithm for all CAT(0) cube complexes.

Returning to median graphs, the following is known about the labeling schemes for them. First, the arboricity of any median graph G on n vertices is at most $\log n$, leading to adjacency schemes of $O(\log^2 n)$ bits per vertex. As noted in [21], one $\log n$ factor can be replaced by the dimension of G. Compact distance labeling schemes can be obtained for some subclasses of cube-free median graphs. One particular class is that of *squaregraphs*, i.e., plane graphs in which all inner vertices have degree ≥ 4. For squaregraphs, distance schemes with labels of size $O(\log^2 n)$ follow from a more general result of [19] for plane graphs of nonpositive curvature. Another such class of graphs is that of partial double trees [6]. Those are the median graphs which isometrically embed into a Cartesian product of two trees. The isometric embedding of partial double trees into a product of two trees immediately leads to distance schemes with labels of $O(\log^2 n)$ bits. Finally, with a technically involved proof, it was shown in [20] that there exists a constant M such that any cube-free median graph G with maximum degree Δ can be isometrically embedded into a Cartesian product of at most $\epsilon(\Delta) := M\Delta^{26}$ trees. This immediately shows that cube-free median graph admit distance labeling schemes with labels of length $O(\epsilon(\Delta)\log^2 n)$. Compared with the $O(\log^3 n)$-labeling scheme obtained in the current paper, the disadvantage of the $O(\epsilon(\Delta)\log^2 n)$-labeling scheme is the dependence from the maximum degree Δ of G. The situation is even worse for high dimensional median graphs: [20] presents an example of a 5-dimensional median graph/CAT(0) cube complex with constant degree which cannot be embedded into a Cartesian product of a finite number of trees. Therefore, for general finite median graphs the function $\epsilon(\Delta)$ does not exist. This in some sense explains the difficulty of designing polylogarithmic distance labeling schemes for general median graphs. Nevertheless, we do not have any indication to believe that such schemes do not exist.

4 Fibers in median graphs

In this section, we recall several useful properties of fibers of gated subgraphs of median graphs. From the definition, one can deduce that median graphs satisfy the following quadrangle condition: *For any vertices u, v, w, z such that $d_G(u, z) = k + 1$, $v, w \sim z$, and $d_G(u, v) = d_G(u, w) = k$, there is a unique vertex $x \sim v, w$ such that $d_G(u, x) = k - 1$.*

▶ **Lemma 1.** *[17]: A subgraph H of a median graph G is gated if and only if any vertex $v \notin V(H)$ is adjacent to at most one vertex of H.*

Combinatorially, the stars of median graphs may have quite an arbitrary structure: by a result of [8], there is a bijection (via the simplex graph operation) between the stars of median graphs and arbitrary graphs. However, from the metric point of view, stars $\mathrm{St}(z)$ have interesting properties:

▶ **Proposition 2.** *The stars $St(z)$ and their fibers $F(x), x \in St(z)$, are gated.*

Both properties of Proposition 2 are known for more general graphs: for gatedness of stars, see [15, Theorem 6.17]) and for gatedness of fibers of gated sets, see [16].

Let H be a gated subgraph of G and let $\mathcal{F}(H) = \{F(x) : x \in V(H)\}$ be the partition of V into fibers. We call two fibers $F(x)$ and $F(y)$ *neighboring* (notation $F(x) \sim F(y)$) if there exists an edge $x'y'$ of G with x' in $F(x)$ and y' in $F(y)$. If $F(x)$ and $F(y)$ are neighboring fibers of H, then denote by $\partial_y F(x)$ the set of all vertices $x' \in F(x)$ having a neighbor y' in $F(y)$ and call $\partial_y F(x)$ the *boundary of* $F(x)$ *relative to* $F(y)$. The following three results can be easily proved.

▶ **Lemma 2.** *Two fibers $F(x)$ and $F(y)$ of H are neighboring if and only if $x \sim y$. Moreover, if $F(x) \sim F(y)$, then $\partial_y F(x)$ induces a gated subgraph of G of dimension $\leq \dim(G) - 1$.*

For a vertex x of H and its fiber $F(x)$, the union of all boundaries $\partial_y F(x)$ over all $F(y) \sim F(x)$, $y \in V(H)$, is called the *total boundary* of the fiber $F(x)$ and is denoted by $\partial^* F(x)$. The boundaries $\partial_y F(x)$ constituting $\partial^* F(x)$ are called *branches* of $\partial^* F(x)$.

▶ **Lemma 3.** *The total boundary of any fiber of H is an isometric subgraph of G not containing* $\dim(G)$*-cubes.*

We conclude this section with an additional property of fibers of stars of *median vertices* of G, i.e., vertices minimizing the function $M(x) = \sum_{v \in V} d_G(x, v)$.

▶ **Lemma 4.** *Let m be a median vertex of a median graph G with n vertices. Then any fiber $F(x)$ of the star $St(m)$ of m has at most $n/2$ vertices.*

Unfortunately, the total boundary of a fiber does not always induce a median subgraph. Therefore, one cannot recursively apply the algorithm to the subgraphs induced by the total boundaries $\partial^* F(x)$. However, if G is cube-free, then the total boundaries of fibers are isometric subtrees of G and one can use for them distance schemes for trees. Even in this case, we still need an additional property of $\partial^* F(x)$. We establish it in the next section.

5 Fibers in cube-free median graphs

In this section, we establish additional properties of fibers and of their total boundaries in cube-free median graphs (for other properties of such graphs, see [11]). Using them we can show that for any pair u, v of vertices of G, the following trichotomy holds: the distance $d_G(u, v)$ either can be computed as $d_G(u, m) + d_G(m, v)$, or as the sum of distances from u, v to appropriate vertices u', v' of $\partial^* F(x)$ plus the distance between u', v' in $\partial^* F(x)$, or via a recursive call to the fiber containing u and v.

5.1 Classification of fibers

Let z be an arbitrary vertex of G and let $\mathcal{F}_z = \{F(x) : x \in \mathrm{St}(z)\}$ denote the partition of V into the fibers of $\mathrm{St}(z)$. We distinguish two types of fibers: the fiber $F(x)$ is called a *panel* if x is adjacent to z and $F(x)$ is called a *cone* if x has distance two to z. The interval $I(x,z)$ is the edge xz if $F(x)$ is a panel and is a square $Q_x := (x, y', z, y'')$ if $F(x)$ is a cone. In the second case, since y' and y'' are the only neighbors of x in $\mathrm{St}(z)$, by Lemma 2 we deduce that the cone $F(x)$ is adjacent to the panels $F(y')$ and $F(y'')$ and that $F(x)$ is not adjacent to any other panel or cone. By the same lemma, a panel $F(y)$ is not adjacent to any other panel, but $F(y)$ is adjacent to all cones $F(x)$ such that the square Q_x contains the edge yz.

5.2 Total boundaries of fibers are quasigated

For a set A, an *imprint* of a vertex $u \notin A$ on A is a vertex $a \in A$ such that $I(u,a) \cap A = \{a\}$. Denote by $\Upsilon(u,A)$ the set of all imprints of u on A. The most important property of imprints is that for any vertex $z \in A$, there exists a shortest (u,z)-path passing via an imprint. Therefore, if the set $\Upsilon(u,A)$ has constant size, one can store in the label of u the distances to the vertices of $\Upsilon(u,A)$. Using this, for any $z \in A$, one can compute $d_G(u,z)$ as $\min\{d_G(u,a) + d_G(a,z) : a \in \Upsilon(u,A)\}$. Note that A is gated iff any $u \notin A$ has a unique imprint on A. We will say that a set A is *quasigated* if $|\Upsilon(u,A)| \leq 2$ for any vertex $u \notin A$. The main goal of this subsection is to show that the total boundaries of fibers are quasigated.

Let T be a tree with a distinguished vertex r in G, called the *root* of T. We will say that a rooted tree T has *gated branches* if for any vertex x of T the unique path $P(x,r)$ of T connecting x to the root r is a gated subgraph of G. Lemma 3 implies:

▶ **Lemma 5.** *The total boundary of any fiber is an isometric tree with gated branches.*

By Lemma 5, $\partial^* F(x)$ has gated branches, however $\partial^* F(x)$ is not necessarily gated itself. Since a panel $F(x)$ may be adjacent to an arbitrary number of cones, one can think that the imprint-set $\Upsilon(u, \partial^* F(x))$ of a vertex u of $F(x)$ may have an arbitrarily large size. The following lemma shows that this is not the case, namely that $|\Upsilon(u, \partial^* F(x))| \leq 2$. This is one of the key ingredients in the design of the distance labeling scheme presented in Section 6. This property is no longer true for median graphs of dimension > 2.

▶ **Lemma 6.** *Any rooted (at r) tree T with gated branches of G is quasigated.*

Proof. Pick any $u \in V \setminus V(T)$ and suppose by way of contradiction that $\Upsilon(u,T)$ contains three distinct imprints x_1, x_2, and x_3. Since T has gated branches, none of the vertices x_1, x_2, x_3 belong to the path of T between r and another vertex from this triplet. In particular, r is different from x_1, x_2, x_3. Suppose additionally that among all rooted trees T' with gated branches of G and such that $|\Upsilon(u,T')| \geq 3$, the tree T has the minimal number of vertices. This minimality choice (and the fact that any subtree of T containing r is also a rooted tree with gated branches) implies that T is exactly the union of the three gated paths $P(r,x_1), P(r,x_2)$, and $P(r,x_3)$. Therefore, x_1, x_2 and x_3 are the leaves of T.

Let y_i be the neighbor of x_i in the path $P(r,x_i)$, $i = 1,2,3$. Since G is bipartite, either $x_i \in I(y_i,u)$ or $y_i \in I(x_i,u)$. Since $x_i \in \Upsilon(u,T)$, necessarily $x_i \in I(y_i,u)$. Let T'_i be the subtree of T obtained by removing the leaf x_i. From the minimality choice of T, we cannot replace T by the subtree T'_i. This means that $|\Upsilon(u,T'_i)| \leq 2$. Since $x_j, x_k \in \Upsilon(u,T'_i)$ for $\{i,j,k\} = \{1,2,3\}$, necessarily $I(y_i,u) \cap \{x_j,x_k\} \neq \varnothing$ holds.

First, notice that $x_1, x_2, x_3 \in I(u,r)$. Indeed, let z_i denote the median of the triplet x_i, u, r. If $z_i \neq x_i$, since $z_i \in I(x_i,r) = P(x_i,r) \subset T$ and $z_i \in I(u,x_i)$, we obtain a contradiction with the inclusion of x_i in $\Upsilon(u,T)$. Thus $z_i = x_i$, yielding $x_i \in I(u,z_i)$.

Now, suppose without loss of generality that $d_G(r, x_3) = \max\{d_G(r, x_i) : i = 1, 2, 3\} := k$. Since $I(y_3, u) \cap \{x_1, x_2\} \neq \varnothing$ as shown above, we can suppose that $x_2 \in I(y_3, u)$. Since $x_3 \in I(y_3, u)$, from these inclusions we obtain that $d_G(x_3, u) + 1 = d_G(y_3, x_2) + d_G(x_2, u)$. Then $d_G(x_3, u) \geq d_G(x_2, u)$, and we conclude that $d_G(x_3, u) = d_G(x_2, u)$ and $d_G(y_3, x_2) = 1$. Since $x_2, x_3 \in I(r, u)$, $d_G(x_3, r) = d_G(x_2, r)$. We distinguish two cases:

Case 1. $d_G(x_1, r) = k$.

Since x_1, x_2, x_3 have the same distance k to r, we can apply to x_1 the same analysis as to x_3 and deduce that the neighbor y_1 of x_1 in T coincides with one of the vertices y_2 or y_3. Since $y_2 = y_3 = y$, we conclude that the vertices x_1, x_2, x_3 have the same neighbor y in T. Since y is closer to r than each of the vertices x_1, x_2, x_3 and since $x_1, x_2, x_3 \in I(r, u)$, we conclude that $x_1, x_2, x_3 \in I(y, u)$. Applying the quadrangle condition three times, we can find three vertices $x_{i,j}, i, j \in \{1, 2, 3\}, i \neq j$, such that $x_{i,j} \sim x_i, x_j$ and $d_G(x_{i,j}, u) = k - 1$. If two of the vertices $x_{1,2}, x_{2,3}$, and $x_{3,1}$ coincide, then we will get a forbidden $K_{2,3}$. Thus $x_{1,2}, x_{2,3}$, and $x_{3,1}$ are pairwise distinct. Since G is bipartite, this implies that $d_G(x_i, x_{j,k}) = 3$ for $\{i, j, k\} = \{1, 2, 3\}$. Since $x_{1,2}, x_{2,3} \in I(x_2, u)$, by quadrangle condition there exists a vertex w such that $w \sim x_{1,2}, x_{2,3}$ and $d_G(w, u) = k - 2$. Since G is bipartite, $d_G(w, x_{3,1})$ equals to 3 or to 1. If $d_G(w, x_{3,1}) = 3 = d(y, w)$, then the triplet $y, w, x_{3,1}$ has two medians x_1 and x_3, which is impossible, because G is median. Thus $d_G(w, x_{3,1}) = 1$, i.e., $w \sim x_{3,1}$. Then one can easily see that the vertices $y, x_1, x_2, x_3, x_{1,2}, x_{2,3}, x_{3,1}, w$ define an isometric 3-cube of G, contrary to the assumption that G is cube-free. This finishes the analysis of Case 1.

Case 2. $d_G(x_1, r) < k$.

This implies that $d_G(r, x_1) \leq k-1 = d_G(r, y)$. Let r' be the neighbor of r in the (r, y)-path of T. Note that $r' \notin I(r, x_1) = P(r, x_1)$. Otherwise, $r' \in P(r, x_1) \cap P(r, x_2) \cap P(r, x_3)$ and we can replace T by the subtree T' rooted at r' and consisting of the subpaths of $P(r, x_i)$ between r' and x_i, $i = 1, 2, 3$. Clearly T' is a rooted tree with gated branches and $x_1, x_2, x_3 \in \Upsilon(u, T')$, contrary to the minimality choice of T. Thus $r' \notin P(r, x_1)$.

Let also $P(r, x_1) = (r, v_1, \ldots, v_{m-1}, v_m =: x_1)$. Note that r may coincide with y_1 and x_1 may coincide with v_1. Since $v_1, r' \in I(r, u)$, by quadrangle condition we will find $v'_2 \sim v_1, r'$ at distance $d_G(r, u) - 2$ from u. Since $r' \notin I(r, x_1)$, $v'_2 \neq v_2$. Since $v_2, v'_2 \in I(v_1, u)$, by quadrangle condition we will find $v'_3 \sim v_2, v'_2$ at distance $d_G(r, u) - 3$ from u. Again, since $r' \notin I(r, x_1)$, $v'_3 \neq v_3$. Continuing this way, we will find the vertices $v'_2, v'_3, \ldots, v'_m, v'_{m+1} =: x'_1$ forming an (r', x'_1)-path $P(r', x'_1)$ and such that $v'_{i+1} \sim v_i, v'_i$, $v'_{i+1} \neq v_{i+1}$, and v'_{i+1} is one step closer to u than v_i and v'_i. From its construction, $P(r', x'_1)$ is a shortest path. We assert that $P(r', x'_1)$ is gated. Otherwise, by Lemma 1, we can find two vertices v'_{i-1} and v'_{i+1} having a common neighbor z' different from v'_i. Let z be the median of the triplet z', v_{i-1}, v_{i+1}. Then z is a common neighbor of z', v_{i-1}, v_{i+1} and z is different from v_i (otherwise, we obtain a forbidden $K_{2,3}$). But then the vertices $v_{i-1}, v_i, v_{i+1}, v'_{i-1}, v'_i, v'_{i+1}, z, z'$ induce in G an isometric 3-cube, contrary to the assumption that G is cube-free. Consequently, $P(r', x'_1)$ is a gated path of G.

Let T'' be the tree rooted at r' and consisting of the gated path $P(r', x'_1)$ and the gated subpaths of $P(r, x_2)$ and $P(r, x_3)$ between r' and x_2, x_3, respectively. Clearly, T'' is a rooted tree with gated branches. Notice that $x'_1, x_2, x_3 \in \Upsilon(u, T'')$. Indeed, if x_2 or x_3 belonged to $I(x'_1, u)$, then $x'_1 \in I(x_1, u)$ and we would conclude that x_2 or x_3 belongs to $I(x_1, u)$, which is impossible because $x_1 \in \Upsilon(u, T)$. On the other hand, x'_1 cannot belong to $I(x_2, u)$ or to $I(x_3, u)$ because $d_G(x'_1, u) = d_G(x_1, u) - 1 \leq d_G(x_2, u) = d_G(x_3, u)$. Consequently, $|\Upsilon(u, T'')| \geq 3$. Since T'' contains less vertices than T, we obtain a contradiction with the minimality choice of T. This concludes the analysis of Case 2, thus T is quasigated. ◀

Applying Lemmas 5 and 6 to the subgraph of G induced by the fiber $F(x)$, we obtain:

▶ **Corollary 3.** *The total boundary $\partial^* F(x)$ of any fiber $F(x)$ is quasigated.*

5.3 Classification of pairs of vertices

In Subsection 5.1, we classified the fibers of $\mathrm{St}(z)$ into panels and cones. In this subsection, we use it to provide a classification of pairs of vertices of G with respect to the partition into fibers, which extends the one done in [19] for planar median graphs.

Let z be an arbitrary fixed vertex of G. Let $\mathcal{F}_z = \{F(x) : x \in \mathrm{St}(z)\}$ be the partition of V into the fibers of $\mathrm{St}(z)$. Let u, v be two arbitrary vertices of G and suppose that u belongs to the fiber $F(x)$ and v belongs to the fiber $F(y)$ of $\mathcal{F}_z$. We say that u and v are *roommates* if they belong to the same fiber, i.e., $x = y$. We say that u and v are *1-neighboring* if $F(x)$ and $F(y)$ are two neighboring fibers (then one of them is a panel and another is a cone). We say that u and v are *2-neighboring* if $F(x)$ and $F(y)$ are distinct cones neighboring with a common panel, i.e., there exists a panel $F(w) \sim F(x), F(y)$. Finally, we say that u and v are *separated* if the fibers $F(x)$ and $F(y)$ are distinct, are not neighboring, and if both $F(x)$ and $F(y)$ are cones, then they are not 2-neighboring. From the definition it follows that any two vertices u, v of G are either roommates, or separated, or 1-neighboring, or 2-neighboring.

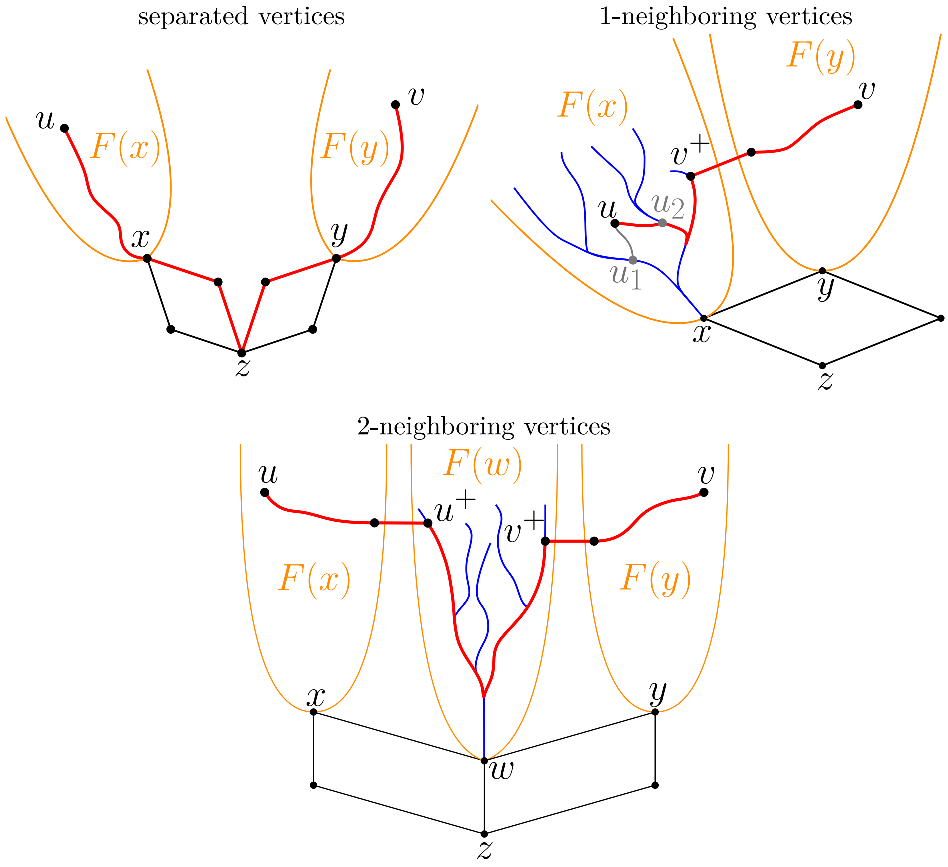


Figure 2 To Lemmas 7, 8 and 9: in red, shortest paths between separated, 1-neighboring, and 2-neighboring vertices u and v. The total boundaries of the panels appear in blue.

We continue with distance formulae for separated, 2-neighboring, and 1-neighboring vertices. The illustration of each of the formulae is provided in Figure 2.

▶ **Lemma 7.** *Two vertices u and v are separated if and only if $d_G(u,v) = d_G(u,z) + d_G(z,v)$.*

▶ **Lemma 8.** *Let u and v be two 1-neighboring vertices such that u belongs to the panel $F(x)$ and v belongs to the cone $F(y)$. Let u_1 and u_2 be the two imprints of u on the total boundary $\partial^* F(x)$ and let v^+ be the gate of v in $F(x)$. Then, $d_G(u,v) = \min\{d_G(u,u_1) + d_{\partial^* F(x)}(u_1,v^+), d_G(u,u_2) + d_{\partial^* F(x)}(u_2,v^+)\} + d_G(v^+,v)$.*

▶ **Lemma 9.** *Let u and v be two 2-neighboring vertices belonging to the cones $F(x)$ and $F(y)$, respectively, and let $F(w)$ be the panel neighboring $F(x)$ and $F(y)$. Let u^+ and v^+ be the gates of u and v in $F(w)$. Then $d_G(u,v) = d_G(u,u^+) + d_{\partial^* F(w)}(u^+,v^+) + d_G(v^+,v)$.*

6 Distance labeling scheme for cube-free median graphs

Let $G = (V,E)$ be a cube-free median graph with n vertices and let m be a median vertex of G. Let u,v be any pair of vertices of G for which we have to compute the distance $d_G(u,v)$. Applying Lemmas 7, 8, and 9 with m instead of z, the distance $d_G(u,v)$ can be computed once u and v are separated, 1-neighboring, or 2-neighboring and once u and v keep in their labels the distances to m, to the respective gates u^+ and v^+, and to the imprints u_1 and u_2 if u belongs to a panel. It also requires keeping in the labels of u and v the information necessary to compute each of the distances $d_{\partial^* F(x)}(u_1,v^+), d_{\partial^* F(x)}(u_2,v^+), d_{\partial^* F(w)}(u^+,v^+)$. Since the total boundaries are isometric trees, this can be done by keeping in the label of u the labels of u_1, u_2, and u^+ in a distance labeling scheme for trees, as well as keeping in the label of v such a label of v^+. This shows that $d_G(u,v)$ can be computed in all cases except when u and v are roommates. Since $F(x)$ is median, we can apply the same recursive procedure to each fiber $F(x)$ instead of G. Therefore, $d_G(u,v)$ is computed in the first recursive call when u and v will no longer belong to the same fiber of the current median vertex (we will sometimes refer at this median vertex as the *separator* of u and v). Since at each step the division into fibers is performed with respect to a median, $|F(x)| \leq n/2$ by Lemma 4, thus the tree of recursive calls has logarithmic depth.

In this section, we present the distance labeling scheme. The encoding scheme is described by the algorithm Dist_Enc presented in Subsection 6.1. Subsection 6.2 presents the algorithm Dist for answering distance queries. In Subsection 6.3, we briefly explain how a constant query time can be achieved by adding $O(\log^2 n)$ bits in head of each label.

6.1 Encoding

We describe now how Dist_Enc constructs for every vertex u of G a distance label $\mathrm{LD}(u)$. This is done recursively and every depth of the recursion is called a *step*. Initially, we suppose that every vertex u of G is given a unique identifier $\mathrm{id}(u)$. We define this naming step as Step 0 and denote the corresponding part of $\mathrm{LD}(u)$ by $\mathrm{LD}_0(u)$, i.e., $\mathrm{LD}_0(u) := \mathrm{id}(u)$. At Step 1, Dist_Enc computes a median vertex m of G, the star $\mathrm{St}(m)$ of m, and the partition $\mathcal{F}_m := \{F(x) : x \in \mathrm{St}(m)\}$ of V into fibers. Every vertex u of G receives the identifier $\mathrm{id}(m)$ of m and its distance $d_G(u,m)$ to m. After that, every vertex x of $\mathrm{St}(m)$ receives a special identifier $\mathrm{L}_{\mathrm{St}(m)}(x)$ of size $O(\log|V|)$ given by a distance labeling for the star $\mathrm{St}(m)$. Then, Dist_Enc computes the gate $u^{\downarrow}$ in $\mathrm{St}(m)$ of every vertex u of G and adds its identifier $\mathrm{L}_{\mathrm{St}(m)}(u^{\downarrow})$ to $\mathrm{LD}(u)$. The identifiers $\mathrm{L}_{\mathrm{St}(m)}(x)$ of the vertices of $\mathrm{St}(m)$ can also be used to distinguish the fibers of $\mathrm{St}(m)$. This triplet $(\mathrm{id}(m), d_G(u,m), \mathrm{L}_{\mathrm{St}(m)}(u^{\downarrow}))$ contains the necessary information relative to $\mathrm{St}(m)$ and is thus referred as the *part "star"* of the information $\mathrm{LD}_1(u)$ given to u at Step 1. We denote this part by $\mathrm{LD}_1^{\mathrm{St}}(u)$. We also set $\mathrm{LD}_1^{\mathrm{St[Med]}}(u) := \mathrm{id}(m)$, $\mathrm{LD}_1^{\mathrm{St[Dist]}}(u) := d_G(u,m)$ and $\mathrm{LD}_1^{\mathrm{St[gate]}}(u) := \mathrm{L}_{\mathrm{St}(m)}(u^{\downarrow})$ for the three components of the label $\mathrm{LD}_1^{\mathrm{St}}(u)$.

Afterwards, at Step 1, the algorithm considers each fiber $F(x)$ of $\mathcal{F}_m$. If $F(x)$ is a panel, then the algorithm computes the total boundary $\partial^* F(x)$ of $F(x)$. The vertices v of the quasigated tree $\partial^* F(x)$ are given special identifiers $\mathrm{LD}_{\partial^* F(x)}(v)$ of size $O(\log^2 |V|)$ consisting of a distance labeling scheme for trees (see [24]). For each vertex u of the panel $F(x)$, the algorithm computes the two imprints u_1 and u_2 of u in $\partial^* F(x)$ (it may happen that $u_1 = u_2$) and stores $(\mathrm{LD}_{\partial^* F(x)}(u_1), d_G(u, u_1))$ and $(\mathrm{LD}_{\partial^* F(x)}(u_2), d_G(u, u_2))$ in $\mathrm{LD}_1^{\mathrm{1st}}(u)$ and $\mathrm{LD}_1^{\mathrm{2nd}}(u)$. If $F(x)$ is a cone and $F(w_1)$ and $F(w_2)$ are the two panels neighboring $F(x)$, then for each vertex u of $F(x)$, the algorithm computes the gates u_1^+ and u_2^+ of u in $F(w_1)$ and $F(w_2)$, respectively. Since $u_i^+ \in \partial_x F(w_i) \subset \partial^* F(x), i = 1, 2$, the labels $\mathrm{LD}_{\partial^* F(w_1)}(u_1^+)$ and $\mathrm{LD}_{\partial^* F(w_2)}(u_2^+)$ in the distance labelings of trees $\partial^* F(w_1)$ and $\partial^* F(w_2)$ are well-defined. Therefore, the algorithm stores $(\mathrm{LD}_{\partial^* F(w_1)}(u_1^+), d_G(u, u_1^+))$ and $(\mathrm{LD}_{\partial^* F(w_2)}(u_2^+), d_G(u, u_2^+))$ in $\mathrm{LD}_1^{\mathrm{1st}}(u)$ and $\mathrm{LD}_1^{\mathrm{2nd}}(u)$. This ends Step 1.

Since $\mathcal{F}_m$ partitions V into gated median subgraphs, the label $\mathrm{LD}_2(u)$ added to $\mathrm{LD}(u)$ at Step 2 is constructed as $\mathrm{LD}_1(u)$ replacing G by the fiber $F(u^\downarrow)$ containing u, and so on. Since each fiber contains no more than half of the vertices of the current graph, at Step $\lceil \log |V| \rceil$, the fiber containing any vertex consists solely of this vertex, and the algorithm stops. Therefore, for each pair of vertices u and v of G, there exists a step of the recursion after which u and v are no longer roommates.

6.2 Distance queries

Let u and v be two vertices of G and let $\mathrm{LD}(u)$ and $\mathrm{LD}(v)$ be their labels returned by Dist_Enc. Here we describe how the algorithm Dist computes the information about the relative positions of u and v with respect to each other and how, using it, computes $d_G(u, v)$. First, the algorithm has to detect if u and v coincide or not. If $u \neq v$, then Dist finds the largest integer i such that $\mathrm{LD}_i^{\mathrm{St[Med]}}(u) = \mathrm{LD}_i^{\mathrm{St[Med]}}(v)$. This corresponds to the first time the vertices u and v belong to different fibers in a partition. Let m be the median vertex of the current median graph that is the separator of u and v. Then, the algorithm Dist retrieves the distances $d := d_G(u^\downarrow, v^\downarrow)$, $d_u := d_G(u^\downarrow, m)$ and $d_v := d_G(v^\downarrow, m)$. This is done by using the identifiers $\mathrm{LD}_i^{\mathrm{St[gate]}}(u)$ and $\mathrm{LD}_i^{\mathrm{St[gate]}}(v)$ and the distance decoder for distance labeling in stars. With this information at hand, one can easily decide for each of the vertices u and v if it belongs to a cone or to a panel, and moreover decide if the vertices u and v are 1-neighboring, 2-neighboring, or separated. In each of these cases, a call to an appropriate function is done.

First suppose that the vertices u and v are 1-neighboring ($d = 1$ and one of d_u, d_v is 1 and the other is 2), i.e., one of the vertices u, v belongs to a cone, the other one belongs to a panel, and the cone and the panel are neighboring. The function `Dist_1-Neighboring` returns the distance $d_G(u, v)$ in the assumption that u belongs to a panel and v belongs to a cone (if v belongs to a panel and u to a cone, it suffices to swap the names of the vertices u and v before using `Dist_1-Neighboring`). The function finds the gate v^+ of v in the panel of u by looking at $\mathrm{LD}_i^{\mathrm{St[gate]}}(v)$ (it also retrieves the distance $d_G(v, v^+)$). It then retrieves the imprint u^* of u (and the distance $d_G(u, u^*)$) on the total boundary of the panel that minimizes the distance of u to one of the two imprints plus the distance from this imprint to the gate v^+ using their tree distance labeling scheme. Finally, `Dist_1-Neighboring` returns $d_G(u, u^*) + d_G(u^*, v^+) + d_G(v^+, v)$ as $d_G(v, u)$.

Now suppose that the vertices u and v are 2-neighboring (i.e., $d = d_u = d_v = 2$). Then both u and v belong to cones. By inspecting $\mathrm{LD}_i^{\mathrm{St[gate]}}(u)$ and $\mathrm{LD}_i^{\mathrm{St[gate]}}(v)$, the function `Dist_2-Neighboring` determines the panel $F(w)$ sharing a border with the cones $F(u^\downarrow)$ and $F(v^\downarrow)$. Then, the function retrieves the respective gates u^+ and v^+ of u and v in this panel $F(w)$ and the distances $d_G(u, u^+)$ and $d_G(v, v^+)$. The distance between the gates u^+ and v^+ is retrieved using the distance decoder for trees. The algorithm returns $d_G(u, u^+) + d_G(u^+, v^+) + d_G(v^+, v)$ as $d_G(u, v)$.

In the remaining cases, the vertices u and v are separated. By Lemma 7, $d_G(u, v) = d_G(u, m) + d_G(m, v)$. Both u and v have stored the median vertex m and their distances to m. Therefore, `Dist_Separated` simply returns the sum of those two distances.

6.3 Complexity analysis and improved query time

The correctness of DIST results from the following properties of G: stars and fibers are gated (Proposition 2); total boundaries of fibers are quasigated (Corollary 3) isometric trees with gated branches (Lemma 5); the formulae for computing the distance between separated, 1-neighboring, and 2-neighboring vertices (Lemmas 7, 8, and 9). At each step of the encoding, $O(\log^2 n)$ bits are added to the label of every vertex (due to the tree-distance labeling scheme they contain). Since there are $\lceil \log n \rceil$ steps, the total length of each label is $O(\log^3 n)$. For decoding the labels, it suffices to read them once to find when the vertices are no longer roommates. This is done in time $O(\log^2 n)$ assuming the *word-RAM model*. Then it might be necessary to decode the distance labels for trees. This can be done in constant time [24].

To sum up, the most costly part of decoding the labels $\mathrm{LD}(u)$ and $\mathrm{LD}(v)$ is to read them up to find the (median) separator of u and v. But with an appropriate $O(\log^2 n)$ bits information concatenated to $\mathrm{LD}(u)$ and $\mathrm{LD}(v)$, one can find this median vertex in $O(1)$ time and then directly jump to the corresponding part of $\mathrm{LD}(u)$ and $\mathrm{LD}(v)$. For that, consider the tree T (of recursive calls) in which vertices at depth i are the median vertices chosen at step i and in which the children of a vertex x are the medians chosen at step $i + 1$ in the fibers generated by x at step i. We can observe that every vertex of G appears in this tree, that the separator m of any two vertices u and v of G is their nearest common ancestor in the tree T, and that its depth j in this tree corresponds to its position in $\mathrm{LD}(u)$ and $\mathrm{LD}(v)$, i.e., $\mathrm{LD}_j^{\mathrm{St[Med]}}(u) = \mathrm{LD}_j^{\mathrm{St[Med]}}(v) = \mathrm{id}(m)$. As noticed in [39], any distance labeling for trees T can be modified to support *nearest common ancestor's depth (NCAD)* queries by adding the depth $\mathrm{depth}(u)$ of u in T to the label $L(u)$ given to each vertex $u \in V(T)$ by the distance labeling. Given two vertices u and v of T, the NCAD decoder then returns $\frac{1}{2}(\mathrm{depth}(u) + \mathrm{depth}(v) - d_T(u, v))$. So, during the execution of DIST_ENC, we can also construct the tree T of recursive calls and then give an NCAD label $L'(u)$ in T to every vertex of G. Now, the first step of DIST will consist in decoding $L'(u)$ and $L'(v)$. Then the algorithm directly reads the parts of $\mathrm{LD}(u)$ and $\mathrm{LD}(v)$ corresponding to the last common median they stored. This establishes Theorem 1.

7 Conclusion

In this paper we presented a distance labeling scheme for cube-free median graphs G with labels of size $O(\log^3 n)$. For that, we considered the partitioning of G into fibers (of size $\leq n/2$) of the star $\mathrm{St}(m)$ of a median vertex m. Each fiber is further recursively partitioned using the same algorithm. We classified the fibers into panels and cones and the pairs of vertices u, v of G into roommates, separated, 1-neighboring, and 2-neighboring pairs. If u and v are roommates, then $d_G(u, v)$ is taken at a later step of the recursion. Otherwise, we showed how to retrieve $d_G(u, v)$ by keeping in the labels of u and v some distances from those vertices to some gates/imprints. Our main ingredient is the fact that the total boundaries of fibers of cube-free median graphs are isometric quasigated trees.

This last property of fibers is an obstacle in generalizing our approach to all median graphs, or even to median graphs of dimension 3. The main problem is that the total boundary is no longer a median graph. Therefore, one cannot apply to this total boundary the distance scheme for cube-free median graphs. Nevertheless, a more brute-force approach works for arbitrary median graphs G of constant maximum degree Δ. In this case, all hypercubes of G have constant size. Thus, the star $\mathrm{St}(m)$ cannot have more than $O(2^{\Delta})$ vertices, i.e., $\mathrm{St}(m)$ has a constant number of fibers. Since every fiber is gated, at every step of the encoding algorithm, every vertex v can store in its label the distance from v to its gates in all fibers of $\mathrm{St}(m)$. Consequently, this leads to distance labeling scheme with labels of (polylogarithmic) length $O(2^{\Delta}\log^3 n)$ for all median graphs with constant maximum degree Δ. We would like to finish this paper with the following question: *Does there exist a polylogarithmic distance labeling scheme for general median graphs or for median graphs of constant dimension?*

References

1 A. Abrams and R. Ghrist. State complexes for metamorphic robots. *Intl. J. Robotics Res.*, 23(7-8):811–826, 2004. `doi:10.1177/0278364904045468`.

2 S. Alstrup, C. Gavoille, E.B. Halvorsen, and H. Petersen. Simpler, faster and shorter labels for distances in graphs. In *SODA*, pages 338–350, 2016.

3 S. Alstrup, I.L. Gørtz, E.B. Halvorsen, and E. Porat. Distance Labeling Schemes for Trees. In *ICALP*, pages 132:1–132:16, 2016.

4 H.-J. Bandelt. Retracts of hypercubes. *J. Graph Theory*, 8(4):501–510, 1984. `doi:10.1002/jgt.3190080407`.

5 H.-J. Bandelt and V. Chepoi. Metric graph theory and geometry: a survey. *Contemporary Mathematics*, 453:49–86, 2008.

6 H.-J. Bandelt, V. Chepoi, and D. Eppstein. Ramified rectilinear polygons: coordinatization by dendrons. *Discr. Comput. Geom.*, 54(4):771–797, 2015. `doi:10.1007/s00454-015-9743-5`.

7 H.-J. Bandelt and J. Hedlíková. Median algebras. *Discr. Math.*, 45(1):1–30, 1983.

8 H.-J. Bandelt and M. van de Vel. Embedding topological median algebras in products of dendrons. *Proc. London Math. Soc.*, s3-58(3):439–453, 1989.

9 F. Bazzaro and C. Gavoille. Localized and compact data-structure for comparability graphs. *Discr. Math.*, 309(11):3465–3484, 2009.

10 L. J. Billera, S.P. Holmes, and K. Vogtmann. Geometry of the space of phylogenetic trees. *Adv. Appl. Math.*, 27:733–767, 2001.

11 B. Brešar, S. Klavžar, and R. Škrekovski. On cube-free median graphs. *Discr. Math.*, 307(3-5):345–351, 2007.

12 M. A. Breuer and J. Folkman. An unexpected result in coding the vertices of a graph. *J. Math. Anal. Appl.*, 20(3):583–600, 1967.

13 M.R. Bridson and A. Haefliger. *Metric Spaces of Non-Positive Curvature*, volume 319 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, Berlin, 1999.

14 J. Chalopin and V. Chepoi. A Counterexample to Thiagarajan's Conjecture on Regular Event Structures. In *ICALP*, pages 101:1–101:14. arXiv:1605.08288, 2017.

15 J. Chalopin, V. Chepoi, H. Hirai, and D. Osajda. Weakly modular graphs and nonpositive curvature. *Memoirs of AMS*, (to appear).

16 M. Chastand. Fiber-complemented graphs. I. Structure and invariant subgraphs. *Discr. Math.*, 226(1-3):107–141, 2001.

17 V. Chepoi. Classification of graphs by means of metric triangles. *Metody Diskret. Analiz.*, 49:75–93, 96, 1989.

18 V. Chepoi. Graphs of some CAT(0) complexes. *Adv. Appl. Math.*, 24(2):125–179, 2000.

19 V. Chepoi, F. F. Dragan, and Y. Vaxès. Distance and routing labeling schemes for non-positively curved plane graphs. *J. Algorithms*, 61(2):60–88, 2006.

20 V. Chepoi and M. F. Hagen. On embeddings of CAT(0) cube complexes into products of trees via colouring their hyperplanes. *J. Comb. Theory, Ser. B*, 103(4):428–467, 2013.

21 V. Chepoi, A. Labourel, and S. Ratel. On density of subgraphs of Cartesian products. *CoRR*, abs/1711.11485, 2017. `arXiv:1711.11485`.

22 Victor Chepoi, Arnaud Labourel, and Sébastien Ratel. Distance and routing labeling schemes for cube-free median graphs. *CoRR*, abs/1809.10508, 2018. `arXiv:1809.10508`.

23 B. Courcelle and R. Vanicat. Query efficient implementation of graphs of bounded clique-width. *Discr. Appl. Math.*, 131(1):129–150, 2003.

24 O. Freedman, P. Gawrychowski, P. K. Nicholson, and O. Weimann. Optimal distance labeling schemes for trees. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 185–194. ACM, 2017.

25 C. Gavoille and C. Paul. Distance labeling scheme and split decomposition. *Discr. Math.*, 273(1-3):115–130, 2003.

26 C. Gavoille and C. Paul. Optimal distance labeling for interval graphs and related graph families. *SIAM J. Discr. Math.*, 22(3):1239–1258, 2008.

27 C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. *J. Algorithms*, 53(1):85–112, 2004.

28 P. Gawrychowski and P. Uznanski. A note on distance labeling in planar graphs. *CoRR*, abs/1611.06529, 2016. `arXiv:1611.06529`.

29 R. Ghirst and Peterson V. The geometry and topology of reconfiguration. *Adv. Appl. Math.*, 38:302–323, 2007.

30 M. Gromov. Hyperbolic groups. In S. M. Gersten, editor, *Essays in group theory*, volume 8 of *Math. Sci. Res. Inst. Publ.*, pages 75–263. Springer, New York, 1987.

31 K. Hayashi. A Polynomial Time Algorithm to Compute Geodesics in CAT(0) Cubical Complexes. In *ICALP*, pages 78:1–78:14, 2018.

32 S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM J. Discr. Math.*, 5(4):596–603, 1992.

33 S. Klavžar and H.M. Mulder. Median Graphs: characterizations, Location Theory and Related Structures. *J. Combin. Math. Combin. Comput.*, 30:103–127, 1999.

34 D.E. Knuth. *The Art of Computer Programming : Vol. 4. Fascicle 0, Introduction to combinatorial algorithms and Boolean functions.* Boston, Mass. ; London : Addison-Wesley, 2008. "Newly available sections of the classic work" –cover.

35 H.M. Mulder. *The Interval Function of a Graph*, volume 132 of *Mathematical Centre Tracts*. Mathematisch Centrum, Amsterdam, 1980.

36 H.M. Mulder and A. Schrijver. Median graphs and Helly hypergraphs. *Discr. Math.*, 25(1):41–50, 1979. `doi:10.1016/0012-365X(79)90151-1`.

37 M. Owen and J.S. Provan. A Fast Algorithm for Computing Geodesic Distances in Tree Space. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 8(1):2–13, 2011. `doi:10.1109/TCBB.2010.3`.

38 D. Peleg. Proximity-preserving labeling schemes. *J. Graph Theory*, 33(3):167–176, 2000. `doi:10.1002/(SICI)1097-0118(200003)33:3\%3C167::AID-JGT7\%3E3.0.CO;2-5`.

39 D. Peleg. Informative labeling schemes for graphs. *Theor. Comput. Sci.*, 340(3):577–593, 2005.

40 M. Roller. Poc sets, median algebras and group actions. Technical report, Univ. of Southampton, 1998.

41 M. Sageev. CAT(0) cube complexes and groups. In M. Bestvina, M. Sageev, and K. Vogtmann, editors, *Geometric Group Theory*, volume 21 of *IAS/Park City Mathematics Series*, pages 6–53. AMS, Institute for Advanced Study, 2012.

42 T.J. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978. `doi:10.1145/800133.804350`.

43 P. M. Winkler. Proof of the squashed cube conjecture. *Combinatorica*, 3(1):135–139, 1983.

44 G. Winskel and M. Nielsen. Models for Concurrency. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science (Vol. 4)*, pages 1–148. Oxford University Press, 1995.

One-Dimensional Guarded Fragments

Emanuel Kieroński
University of Wrocław, Poland
kiero@cs.uni.wroc.pl

Abstract

We call a first-order formula one-dimensional if every maximal block of existential (or universal) quantifiers in it leaves at most one variable free. We consider the one-dimensional restrictions of the guarded fragment, GF, and the tri-guarded fragment, TGF, the latter being a recent extension of GF in which quantification for subformulas with at most two free variables need not be guarded, and which thus may be seen as a unification of GF and the two-variable fragment, FO^2. We denote the resulting formalisms, resp., GF_1, and TGF_1. We show that GF_1 has an exponential model property and NExpTime-complete satisfiability problem (that is, it is easier than full GF). For TGF_1 we show that it is decidable, has the finite model property, and its satisfiability problem is 2-ExpTime-complete (NExpTime-complete in the absence of equality). All the above-mentioned results are obtained for signatures with no constants. We finally discuss the impact of their addition, observing that constants do not spoil the decidability but increase the complexity of the satisfiability problem.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases guarded fragment, two-variable logic, satisfiability, finite model property

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.16

Related Version A full version of this paper is available at `https://arxiv.org/abs/1904.04572`.

Funding Supported by Polish National Science Centre grant No 2016/21/B/ST6/01444.

Acknowledgements The author would like to thank Sebastian Rudolph and the anonymous reviewers for their helpful comments.

1 Introduction

The *guarded fragment* of first-order logic, GF, is obtained by requiring all quantifiers to be appropriately relativised by atoms. It was introduced by Andréka, van Benthem and Németi [1] as a generalization of propositional modal logic and may be also seen as an extension of some standard description logics. GF has good algorithmic and model-theoretic properties. In particular, Grädel proved that its satisfiability problem is decidable, it has a tree-like model property and the finite model property [7]. The idea of GF turned out to be very fruitful and found numerous applications. In this paper we consider some modifications of the syntax of GF. Our aim is to check if in this way we can obtain interesting fragments with better complexity and/or attractive expressiveness.

The satisfiability problem for GF is 2-ExpTime-complete. This relatively high complexity can be lowered to ExpTime either by bounding the number of variables, or the arity of relation symbols [7]. We propose another way of decreasing the complexity without sacrificing either the number of variables or the arity of relations. The idea is to restrict formulas to be *one-dimensional*. We say that a formula is one-dimensional if every maximal block of existential (or universal) quantifiers in it leaves at most one variable free. We remark that the one-dimensional restriction of full first-order logic, F_1, is undecidable, as observed by Hella and Kuusisto [9]. We denote the intersection of F_1 and GF by GF_1 and call it the *one-dimensional guarded fragment*. While this variation decreases the expressive power of the logic, we believe that it is still quite interesting, as, in particular, it still embeds propositional modal logic, and most standard description logics embeddable in full GF. Thus, as GF, it may serve as an extension of modal/description logics to contexts with relations of arbitrary arity. We show that the satisfiability problem for GF_1 is NExpTime-complete and that it has an exponential model property, that is, its every satisfiable formula has a model of size bounded exponentially in its length. This is in contrast to full GF in which one can enforce doubly exponentially large models. Moreover, proving the finite model property for GF_1 is much easier than for full GF, in particular it does not need complicated combinatorial constructions used in the case of GF (in [7], and in Bárány, Gottlob and Otto [2]). We obtain a corresponding NExpTime-lower bound even for a weaker logic, *uniform* GF_1, that is the intersection of GF_1 and *uniform* F_1, UF_1, the latter being a decidable restriction of F_1 introduced in [9] as a canonical generalization of the two-variable fragment FO^2 (with equality) to scenarios involving relations of arity greater than two (see Kieroński, Kuusisto [13] where NExpTime-completeness of UF_1 is shown). This is slightly surprising, since in many aspects UF_1 behaves similarly to the two-variable fragment, FO^2, and the guarded version of the latter is ExpTime-complete [7].

We also consider an extension of GF called the *tri-guarded fragment*, TGF. In TGF quantification for subformulas with at most two free variables may be used freely, without guards. Hence, TGF unifies GF and the already-mentioned FO^2. We borrowed the term *tri-guarded fragment* from a recent work by Rudolph and Šimkus [15], but, actually, the idea behind TGF is not new and can be traced back already in Kazakov's PhD thesis [11] where the fragment $\mathrm{GF}|\mathrm{FO}^2$, essentially identical with TGF, was defined. A similar logic, GF *with binary cross product*, $\mathrm{GF}^{\times_2}$, is also considered by Bourhis, Morak and Pieris [4]. Both $\mathrm{GF}|\mathrm{FO}^2$ and $\mathrm{GF}^{\times_2}$ do not allow constant symbols. We remark that in our initial scenario we also assume that constants are not present in signature; however, we will discuss their addition later.

Similarly to GF, FO^2 is a seminal fragment of first-order logic, and its importance is justified, *inter alia*, by its close relationships to modal and description logics. Mortimer [14] demonstrated that it has the finite model property and Grädel, Kolaitis and Vardi [8] proved that its satisfiability problem is NExpTime-complete. Each of the logics GF, FO^2 has some advantages and drawbacks with respect to the other. We mention here the fact that GF allows only to express properties of a local character, *e.g.*, it cannot express $\forall xy(Px \wedge Qy \rightarrow Rxy)$, while FO^2 does not allow for a non-trivial use of relations of arity greater than two. TGF offers a substantial improvement in these aspects. Moreover, in TGF we can embed the Gödel class, that is the class of all prenex formulas of the form $\forall xy \exists \bar{z} \psi(x, y, \bar{z})$. Indeed, any such formula has an equisatisfiable TGF formula obtained just by an addition of a dummy guard, as follows, $\forall xy \exists \bar{z}(G(x, y, \bar{z}) \wedge \psi(x, y, \bar{z}))$, where G is a fresh relation symbol of the appropriate arity. Such embedding implies, however, that the satisfiability problem for TGF with equality is undecidable, since the Gödel class with equality is undecidable, as proved by Goldfarb [6]. The undecidability of TGF with equality is also shown in [15] by a direct grid encoding. On the positive side, it turns out that the satisfiability problem for TGF *without equality* is decidable and 2-ExpTime-complete. It was proved in [11] by a resolution method, and follows also from the decidability of $\mathrm{GF}^{\times}_{2}$, shown in [4] by a use of the database-theoretic concept of *chase*.[1]

[1] A footnote in [4] suggests that the decidability of GF with binary cross-product is retained in the presence of equality. This has however been later later refuted by the authors (private communication). GF with binary cross product with equality is undecidable by the same arguments we gave for TGF.

In this paper we consider a natural combination of GF_1 and TGF, the *one-dimensional triguarded fragment*, TGF_1, which, on the one hand, allows us to use unguarded quantification for subformulas with at most two free variables, but, on the other hand, requires to obey the one-dimensionality restriction. We show that this variant is decidable even in the presence of equality. The complexity, however, depends on the presence/absence of equality: The satisfiability problem is 2-ExpTime-complete with equality and NExpTime-complete without it. The logic has the finite model property (we remark that whether full TGF has the finite model property is an open question), and, again, a bound on the size of minimal models is doubly- or singly exponential, depending on whether equality is allowed or not. TGF_1 may be seen as a decidable generalization of FO^2 (with equality) to scenarios with relations of arity greater than two, alternative and orthogonal in the expressive power to the above-mentioned UF_1. We also remark that TGF_1 can express the concept of *nominals* from description logics, since the combination of equality and unguarded quantification for subformulas with two free variables allows us to say that some unary predicates hold for unique elements of a model. Thus we can embed in TGF_1, *e.g.*, the description logic $\mathcal{ALC}$ plus inverse roles ($\mathcal{I}$), nominals ($\mathcal{O}$), role hierarchies ($\mathcal{H}$), and any Boolean combination of roles (including their negations).

We then briefly consider applications of the ideas of one-dimensionality and tri-guardedness to two decidable extensions of GF, namely, the loosely guarded fragment, LGF, introduced by van Benthem [17], and the guarded negation fragment, GNFO, proposed by Bárány, ten Cate and Segoufin [3]. Regarding one-dimensionality, it helps in the case of LGF: one-dimensional LGF has an exponential model property and NExpTime-complete satisfiability problem (exactly as GF_1), but does not help in the case of GNFO, where the one-dimensional variant remains 2-ExpTime-hard. Regarding the tri-guardedness, the results are negative: both LGF and GNFO, even in their one-dimensional variants, become undecidable when unguarded quantification for subformulas with two free variables is allowed.

As remarked, all the results discussed above are obtained under the assumption that constants are not present in signatures. It turns out that all the decidability results are preserved in the presence of constants. However, interestingly, the computational complexity may change (we recall that for GF constants make no difference [7]). This is also the case for TGF with constants, without equality, which is shown in [15] to be 2-NExpTime-complete. Here we show that a 2-NExpTime-lower bound can be obtained even for TGF_1 with constants, without equality. We also observe that the presence of constants lifts the complexity of GF_1 to 2-ExpTime.

In Table 1 we summarize the above-discussed complexity results for the variations of GF. We point out an interesting status of TGF_1: it is NExpTime-complete without equality and constants, 2-ExpTime-complete with equality and without constants, and 2-NExpTime-complete with constants (with or without equality).

We finally remark that further pushing the concepts of one-dimensionality and tri-guardedness to, resp., *two*-dimensionality and *tetra*-guardedness does not lead to attractive results. Indeed, a 2-ExpTime lower bound for two-dimensional GF can be shown by a slight adaptation of the bound for full GF from [7]; allowing for unguarded quantification for subformulas with three free variables gives undecidability, as the resulting logic contains the undecidable three-variable fragment of FO (see, *e.g.*, Kahr, Moore and Wang [10]). Undecidability of the three-variable fragment can be easily shown even using only one-dimensional formulas.

Table 1 Complexities of the guarded fragments. If the presence of constants makes a difference, the complexity of the variant with constants is given in the brackets. All logics have the finite model property. Results of this paper are distinguished in bold.

logic	with =	without =
GF	2-ExpTime.	2-ExpTime
TGF	undecidable	2-ExpTime (2-NExpTime)
GF_1	**NExpTime (2-ExpTime)**	**NExpTime (2-ExpTime)**
TGF_1	**2-ExpTime (2-NExpTime)**	**NExpTime (2-NExpTime)**

2 Preliminaries

We mostly work with purely relational signatures with no constants and function symbols (only in Section 6 we consider signatures with constants). For convenience we also assume that there are no relation symbols of arity 0. We refer to structures using Fraktur capital letters, and to their domains using the corresponding Roman capitals. Given a structure $\mathfrak{A}$ and some $B \subseteq A$ we denote by $\mathfrak{A}{\restriction}B$ or just by $\mathfrak{B}$ the restriction of $\mathfrak{A}$ to its subdomain B.

We usually use $a, b, \ldots$ to denote elements from domains of structures, $\bar{a}$, $\bar{b}$, ... for tuples of elements, x, y, ... for variables and $\bar{x}$, $\bar{y}$, ... for tuples of variables; all of these possibly with some decorations. For a tuple of variables $\bar{x}$ we use $\psi(\bar{x})$ to denote a formula (or subformula) ψ, whose all free variables are in $\bar{x}$.

An *atomic l-type* β over a signature σ is a maximal consistent set of atomic or negated atomic formulas (including equalities/inequalities) over σ in l variables $x_1, \ldots, x_l$. We often identify a type with the conjunction of its elements, $\beta(x_1, \ldots, x_l)$. For an l-type β we denote by $\beta{\restriction}x_i$ $(i = 1, \ldots, l)$ the 1-type obtained by removing from β all the literals that use some x_j, with $j \neq i$, and then replacing all occurrences of x_i by x_1. We will be particularly interested in 1-types and 2-types over signatures σ consisting of the relation symbols used in some given formula. Observe that the number of 1-types is bounded by a function which is exponential in $|\sigma|$, and hence also in the length of the formula. This is because a 1-type just corresponds to a subset of σ. On the other hand, the number of 2-types may be doubly exponentially large. Indeed, using an n-ary predicate and two fixed variables one can build 2^n atoms which then can be used to form 2^{2^n} different 2-types.

Let $\mathfrak{A}$ be a structure, and let $a, b \in A$ be such that $a \neq b$. We denote by $\mathrm{tp}^{\mathfrak{A}}(a)$ the unique atomic 1-type *realized* in $\mathfrak{A}$ by the element a, *i.e.*, the 1-type $\alpha(x)$ such that $\mathfrak{A} \models \alpha(a)$; similarly by $\mathrm{tp}^{\mathfrak{A}}(a, b)$ we denote the unique atomic 2-type realized in $\mathfrak{A}$ by pair (a, b), *i.e.*, the 2-type $\beta(x, y)$ such that $\mathfrak{A} \models \beta(a, b)$. For $B \subseteq A$ we denote by $\boldsymbol{\alpha}[B]$ the set of all 1-types realized in $\mathfrak{A}$ by elements of B.

Below we define several fragments of first-order logic, FO, including two new fragments, GF_1 and TGF_1. Each of the fragments is defined as the least set of formulas (i) containing all atomic formulas (including equalities), (ii) closed under Boolean connectives, and (iii) satisfying appropriate (depending on the fragment) rules of using quantifiers, specified below ($\bar{x}$, $\bar{y}$ represent here any tuples of variables and x, y represent any variables):

- *Guarded fragment* of first-order logic, GF:
  - if $\psi(\bar{x}, \bar{y}) \in \mathrm{GF}$ then $\forall \bar{x}(\gamma(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{y}))$ and $\exists \bar{x}(\gamma(\bar{x}, \bar{y}) \wedge \psi(\bar{x}, \bar{y}))$ belong to GF, where $\gamma(\bar{x}, \bar{y})$ is an atomic formula containing all the free variables of ψ, called a *guard* for ψ.
- *One-dimensional fragment* of first-order logic, F_1:
  - if $\psi(\bar{x}, y) \in \mathrm{F}_1$ then $\exists \bar{x} \psi(\bar{x}, y)$ and $\forall \bar{x} \psi(\bar{x}, y)$ belong to F_1.

- *One-dimensional guarded fragment*, GF_1:
  - if $\psi(\bar{x}, y) \in \mathrm{GF}_1$ then $\forall \bar{x}(\gamma(\bar{x}, y) \rightarrow \psi(\bar{x}, y))$ and $\exists \bar{x}(\gamma(\bar{x}, y) \wedge \psi(\bar{x}, y))$ belong to GF_1, where $\gamma(\bar{x}, y)$ is a guard for ψ.
- *Tri-guarded fragment*, TGF:
  - if $\psi(\bar{x}, \bar{y}) \in \mathrm{TGF}$ then $\forall \bar{x}(\gamma(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{y}))$ and $\exists \bar{x}(\gamma(\bar{x}, \bar{y}) \wedge \psi(\bar{x}, \bar{y}))$ belong to TGF, where $\gamma(\bar{x}, \bar{y})$ is a guard for ψ,
  - if $\psi(x, y)$ is in TGF, then $\exists x \psi(x, y)$ and $\forall x \psi(x, y)$ belong to TGF.
- *One-dimensional tri-guarded fragment*, TGF_1:
  - if $\psi(\bar{x}, y) \in \mathrm{TGF}_1$ then $\forall \bar{x}(\gamma(\bar{x}, y) \rightarrow \psi(\bar{x}, y))$ and $\exists \bar{x}(\gamma(\bar{x}, y) \wedge \psi(\bar{x}, y))$ belong to TGF_1, where $\gamma(\bar{x}, y)$ is a guard for ψ,
  - if $\psi(x, y)$ is in TGF_1, then $\exists x \psi(x, y)$ and $\forall x \psi(x, y)$ belong to TGF_1.

Note that GF_1 is just the intersection of GF and F_1, TGF contains both GF and FO^2, and TGF_1 is the intersection of TGF and F_1, containing full FO^2.

We recall that the satisfiability problem for F_1 is undecidable [9]. To regain decidability its *uniform* restriction, UF_1, was introduced in [9]. Roughly speaking, a boolean combination of atoms is allowed in UF_1 if all of them use precisely the same set of variables; the exceptions are atoms with one free variable and equalities, which may be used freely. See [9] or [13] for a formal definition and more details on UF_1.

We will also be interested in the loosely guarded fragment, LGF, the guarded negation fragment, GNFO, and their one-dimensional and tri-guarded variations. They will be introduced in Section 5.

3 Finite model property

In this section we prove the finite model property for TGF_1 and obtain (essentially optimal) upper bounds on the size of minimal models of its satisfiable formulas, as well as of formulas of its interesting subfragments.

We introduce a Scott-type normal form for TGF_1. Given a TGF_1 formula φ we say that it is in *normal form* if it has the following shape

$$\bigwedge_{i \in I} \forall \bar{x}(\gamma_i(\bar{x}) \rightarrow \psi_i(\bar{x})) \wedge \bigwedge_{i \in I'} \forall x \exists \bar{y} \psi'_i(x, \bar{y}) \wedge \forall x y \psi''(x, y) \qquad (1)$$

where I, I' are some sets of indices, the ψ_i, ψ'_i, and ψ'' represent arbitrary quantifier-free formulas, and for every i, γ_i is a proper guard for ψ_i. We remark that we do not require guards in formulas of the form $\forall\exists$, even if they contain more than two variables, as their presence there is inessential (cf. Remark in [7], p. 1725). In a rather standard fashion one can show the following lemma.

▶ **Lemma 1.** *There is a polynomial nondeterministic procedure, taking as its input a* TGF_1 *formula* φ *and producing a normal form formula* φ' *(over an extended signature), such that*
(i) *if* $\mathfrak{A} \models \varphi$ *for some structure* $\mathfrak{A}$ *then there is a run of the procedure producing a normal form* φ' *such that* $\mathfrak{A}' \models \varphi'$ *for some expansion* $\mathfrak{A}'$ *of* $\mathfrak{A}$*,*
(ii) *if the procedure has a run producing* φ' *and* $\mathfrak{A}' \models \varphi'$*, for some* $\mathfrak{A}'$*, then* $\mathfrak{A}' \models \varphi$*.*
Moreover, if φ *is without equality then the procedure produces* φ' *without equality; if* φ *is in* GF_1 *then the last conjunct* $\forall x y \psi''(x, y)$ *is not present in* φ'*.*

Lemma 1 allows us, when dealing with decidability or complexity issues and when considering the size of minimal models of formulas in TGF_1, to restrict attention to normal form sentences. The part of this lemma starting with "moreover" will allow us to use it effectively for TGF_1 without equality and for GF_1.

Our normal form is similar to normal form for GF [7]. It adapts the latter to the one-dimensional setting and extends it by the last type of conjuncts. The conversion to normal form in [7] is deterministic, it however cannot be used directly in our case as it adds one free variable to every subformula, which spoils one-dimensionality and may lead to unguarded subformulas with three variables.

Let φ be a normal form formula and $\mathfrak{A}$ its model. Take $a \in A$ and a conjunct $\psi = \forall x \exists \bar{y} \psi_i'(x, \bar{y})$ of φ. Let $\bar{b}$ be a tuple of elements of $\mathfrak{A}$ such that $\mathfrak{A} \models \psi_i'(a, \bar{b})$. Then $\mathfrak{A}\restriction(\{a\} \cup \bar{b})$ is called a *witness structure* for a and ψ.

▶ **Theorem 2.** *Every satisfiable formula φ in*

(i) TGF_1 *(with equality) has a finite model of size bounded doubly exponentially in $|\varphi|$.*
(ii) TGF_1 *without equality has a finite model of size bounded exponentially in $|\varphi|$.*
(iii) GF_1 *(with or without equality) has a finite model of size bounded exponentially in $|\varphi|$.*

We concentrate on showing (i) and then obtain (ii) and (iii) as a corollary from the finite model construction presented. Let φ be a normal form TGF_1 formula as in (1), and denote $n = |\varphi|$. Let us fix an arbitrary model $\mathfrak{A}$ of φ. We construct a bounded model $\mathfrak{B} \models \varphi$. We mimic the scheme of the classical construction from [8] showing an exponential model property for FO^2, in particular we adapt the notions of *kings* and *court*. The details, however, are more complicated.

Court. We say that an element $a \in A$ is a *king* if $\mathrm{tp}^{\mathfrak{A}}(a)$ is realized in $\mathfrak{A}$ only by a; $\mathrm{tp}^{\mathfrak{A}}(a)$ is then called *royal*. As in the case of FO^2 kings are important as their duplication may be forbidden by formulas like $\forall xy(Px \wedge Py \rightarrow x = y)$. Let $K \subseteq A$ be the set of kings of $\mathfrak{A}$. For each $a \in K$ and each $i \in I'$ choose a witness structure $\mathfrak{W}_{a,i}$ for a and ψ_i' in $\mathfrak{A}$. Let $C = K \cup \bigcup_{a,i} W_{a,i}$. We call $\mathfrak{C}$ the *court* of $\mathfrak{A}$. The court will be retained in $\mathfrak{B}$. Note that the number of elements in C is bounded exponentially in n, and it that the structure $\mathfrak{C}$ can be described using exponentially many bits (the latter is true since the arity of all relation symbols is bounded by n). Note that K, and thus also C may be empty.

Pattern witness structures. For each non-royal element $a \in A \setminus K$ we say that the isomorphism type of the structure $\mathfrak{A}\restriction(K \cup \{a\})$ is the $\mathfrak{K}$-type of a. Note that from a $\mathfrak{K}$-type of an element one can infer its 1-type, and that the number of the $\mathfrak{K}$-types realized in $\mathfrak{A}$ is bounded doubly exponentially in n. Denote by $\boldsymbol{\alpha}^{\mathfrak{K}}$ the set of $\mathfrak{K}$-types realized in $\mathfrak{A}$ by the elements of $A \setminus K$. Later, we will allow ourselves to use the notion of a $\mathfrak{K}$-type in a natural way also for other structures with a distinguished substructure $\mathfrak{K}$. For each $\pi \in \boldsymbol{\alpha}^{\mathfrak{K}}$ choose an element a having $\mathfrak{K}$-type π in $\mathfrak{A}$ and for each $i \in I'$ choose a witness structure $\mathfrak{W}_{\pi,i}$ for a and ψ_i'. Let $\mathfrak{W}_{\pi,i}^* = \mathfrak{W}_{\pi,i}\restriction(W_{\pi,i} \setminus (K \cup \{a\}))$. For each $\pi \in \boldsymbol{\alpha}^{\mathfrak{K}}$, $i \in I'$ and $j = 0, 1, 2$ let $\mathfrak{W}_{\pi,i,j}^*$ be a fresh isomorphic copy of $\mathfrak{W}_{\pi,i}^*$.

Universe. We define the universe of $\mathfrak{B}$ as follows $B := C \cup \bigcup_{\pi,i,j} W_{\pi,i,j}^*$, where π ranges over $\boldsymbol{\alpha}^{\mathfrak{K}}$, i over I' and j over $\{0, 1, 2\}$. We emphasise that the sets $W_{\pi,i,j}^*$ are disjoint from C and from each other. We retain in $\mathfrak{B}$ the structure on C from $\mathfrak{A}$ and for each π, i, j we make $\mathfrak{B}\restriction(K \cup W_{\pi,i,j}^*)$ isomorphic to $\mathfrak{A}\restriction(K \cup W_{\pi,i}^*)$. This, in particular, makes the $\mathfrak{K}$-type in $\mathfrak{B}$ of each element b belonging to some $W_{\pi,i,j}^*$ identical with the $\mathfrak{K}$-type in $\mathfrak{A}$ of the counterpart of b from the original substructure $\mathfrak{W}_{\pi,i}$.

Witness structures for the court. Let us consider an element $c \in C \setminus K$, and denote by π its $\mathfrak{K}$-type in $\mathfrak{A}$. For every $i \in I'$ make $\mathfrak{B}\restriction(\{c\} \cup (W_{\pi,i} \cap K) \cup W_{\pi,i,0}^*)$ isomorphic to $\mathfrak{W}_{\pi,i}$. This provides a witness structure for c and ψ_i' in $\mathfrak{B}$. Note that a single such step (for fixed c

and i) consists in defining relations on tuples containing c, at least one element of $W^*_{\pi,i,0}$ and possibly some elements of K, since relations on other relevant tuples were defined in the desired way in step *Universe*. Note that no conflicts (attempts to set the same atom to both true and false) can arise, when we perform this step for some c and i and then for the same c and some $i' \neq i$, because in the first case we define truth-values of relations only on tuples containing some element from $W^*_{\pi,i,0}$, and in the second – only on tuples containing some element from $W^*_{\pi,i',0}$, and $W^*_{\pi,i,0}$ is disjoint from $W^*_{\pi,i',0}$. Finally, when we perform this step for some c, and then for some $c' \neq c$ no conflicts arise since in the first case we define relations only on tuples containing c but not c' and in the second – only on tuples containing c' but not c.

Witness structures for the other elements. Consider now any element $b \in B \setminus C$. Assume it belongs to $W^*_{\pi',i',j'}$ and that π is the $\mathfrak{K}$-type of b in $\mathfrak{B}\restriction(K \cup \{b\})$. For each $i \in I'$ make the structure on $\{b\} \cup (W_{\pi,i} \cap K) \cup W^*_{\pi,i,(j'+1 \mod 3)}$ isomorphic to $\mathfrak{W}_{\pi,i}$. This provides a witness structure for b and ψ'_i in $\mathfrak{B}$. Again, to do it we need to define relations on some tuples containing b and some element of $W^*_{\pi,i,(j'+1 \mod 3)}$, and, due to our strategy, this can be done without conflicts.

Completing the structure. For any pair of distinct elements $b, b' \in B$ whose 2-type has not yet been defined in $\mathfrak{B}$ choose a pair of distinct elements a, a' with $\text{tp}^{\mathfrak{A}}(a) = \text{tp}^{\mathfrak{B}}(b)$ and $\text{tp}^{\mathfrak{A}}(a') = \text{tp}^{\mathfrak{B}}(b')$, and set $\text{tp}^{\mathfrak{B}}(b, b') := \text{tp}^{\mathfrak{A}}(a, a')$. An appropriate pair a, a' exists even if $\text{tp}^{\mathfrak{B}}(b) = \text{tp}^{\mathfrak{B}}(b')$ since at least one of b, b' has a non-royal type. For any tuple $\bar{b}$ of elements of B containing at least three distinct elements, and any relation symbol R of arity $|\bar{b}|$, if the truth-value of $R(\bar{b})$ in $\mathfrak{B}$ has not yet been defined then set it to *false*.

This finishes the definition of $\mathfrak{B}$. Let us now estimate its size. We can bound the number and the arity of relation symbols by $n = |\varphi|$. Then the size of K is bounded by the number of possible 1-types, 2^n. The size of C is bounded by $2^n \cdot n(n-1)$, as each element a of K may need at most n witness structures each of them containing (besides a) at most $n-1$ elements. The number of possible relations of arity at most n on a a set of $2^n + 1$ elements is bounded by $2^{(2^n+1)^n} \leq 2^{2^{n^2+n}}$, thus the number of $\mathfrak{K}$-types is bounded by $(2^{2^{n^2+n}})^n = 2^{n \cdot 2^{n^2+n}} \leq 2^{2^{n^2+2n}} \leq 2^{2^{2n^2}}$ (for $n > 1$). Finally, we can bound the size of B by $2^n + 2^n \cdot n(n-1) + 3n(n-1) \cdot 2^{2^{2n^2}}$, doubly exponentially in n.

Presently, we explain that $\mathfrak{B} \models \varphi$. First note that for each $b \in B$ and each $i \in I'$ there is an appropriate witness structure: if $b \in K$ then this witness structure is provided in $\mathfrak{C}$ which is a substructure of $\mathfrak{B}$. If $b \in C \setminus K$ or $b \in B \setminus C$ then a proper witness structure is provided explicitly either in step *Witness structure for the court* or, resp., *Witness structures for the other elements*. Thus $\mathfrak{B}$ satisfies all conjuncts of φ of the form $\forall x \exists \bar{y} \psi'_i(x, \bar{y})$.

Consider any conjunct $\forall \bar{x}(\gamma_i(\bar{x}) \rightarrow \psi_i(\bar{x}))$ of φ and a tuple of elements $\bar{b}$ such that $\mathfrak{B} \models \gamma_i(\bar{b})$. If $\bar{b} \subseteq C$ or $\bar{b} \subseteq K \cup W^*_{\pi,i,j}$ for some π, i, j then the structure on $\bar{b}$ was made an isomorphic copy of some substructure of $\mathfrak{A}$ in step *Universe*. Otherwise $\bar{b}$ contains at least two distinct elements. In this case the structure on $\bar{b}$ was made an isomorphic copy of some substructure of $\mathfrak{A}$ either in one of the steps *Witness structures for the court*, *Witness structures for the other elements* or in step *Completing the structure* (in this last subcase $\bar{b}$ contains precisely two distinct elements). Thus $\mathfrak{B} \models \psi_i(\bar{b})$. Finally, consider the conjunct $\forall xy \psi''(x, y)$ and take any pair $b, b' \in B$. Again, the structure on $\{b, b'\}$ is an isomorphic copy of a substructure of $\mathfrak{A}$ defined (at the latests) in step *Completing the structure*.

This finishes the proof of (i). To see (ii) and (iii) we first observe that in both cases every satisfiable formula φ has a model without kings. Given a structure $\mathfrak{A}$ we define two new structures $2\mathfrak{A}$ and $2\mathfrak{A}^+$, each of them with universe $A \times \{0,1\}$ and the substructures on $A \times \{0\}$ and $A \times \{1\}$ isomorphic to $\mathfrak{A}$. In $2\mathfrak{A}$ we make these two copies of $\mathfrak{A}$ completely disjoint by setting the truth-value of $R(\bar{a})$ to *false* for any R and any tuple $\bar{a}$ (of the appropriate length) contained neither in $A \times \{0\}$ nor $A \times \{1\}$. In $2\mathfrak{A}^+$, for any tuple $\bar{a}$ contained neither in $A \times \{0\}$ nor $A \times \{1\}$ and for any relation symbol R of arity $|\bar{a}|$, if this tuple contains at least three distinct elements then we also define $R(\bar{a})$ to be *false*. If $\bar{a}$ contains just two distinct elements, say $(a,0)$ and $(a',1)$, then for any relation symbol R or arity $|\bar{a}|$ set $R(\bar{a})$ *true* iff $\mathfrak{A} \models R(\bar{a}{\restriction}1)$ where $\bar{a}{\restriction}1$ is the projection of the elements of $\bar{a}$ on their first position.

Observations that if φ is without equality and $\mathfrak{A} \models \varphi$ then $2\mathfrak{A}^+ \models \varphi$, and that if φ is in GF_1 (even with equality) and $\mathfrak{A} \models \varphi$ then $2\mathfrak{A} \models \varphi$ are routine. Of course our new models are without kings. Starting our small model construction from a model without kings we get $K = \emptyset$ and thus $\mathfrak{K}$-types trivialize to 1-types, which means that their number is bounded singly exponentially. Also $C = \emptyset$ and thus we construct $\mathfrak{B}$ out of the $W^*_{\pi,i,j}$ where π ranges over the set of 1-types, the number of possible i is linear in n and there are just three possible values of j. The size of each $\mathfrak{W}^*_{\pi,i,j}$ is linear in n. The size of the constructed models can be thus estimated by $3n(n-1) \cdot 2^n$. Hence part (ii) and (iii) of Thm. 2 hold.

4 Complexity

In this section we establish the complexity of the considered logics.

▶ **Theorem 3.** *The satisfiability problem (= finite satisfiability problem)*

(i) *for* TGF_1 *with equality is* 2-ExpTime*-complete.*

(ii) *for* TGF_1 *without equality is* NExpTime*-complete.*

(iii) *for* GF_1 *is* NExpTime*-complete.*

Upper bound in (i). We design an alternating satisfiability test for TGF_1 using only exponential space. A 2-ExpTime-upper bound follows then from the fact that AExpSpace=2-ExpTime (Chandra, Kozen, Stockmeyer [5]). The procedure takes as its input a TGF_1 formula φ and works as described below. For simplicity our description is slightly informal. In particular, we do not precisely specify how structures constructed during its execution are represented. We also allow ourselves to write “guess an object X such that Y” instead of more accurate “guess an object X; verify if X meets property Y; if it does not then **reject**”.

1. Nondeterministically compute a normal form φ' as in Lemma 1. Let $n := |\varphi'|$.
2. **Guess** a set of 1-types $\boldsymbol{\alpha} = \boldsymbol{\alpha}_r \dot{\cup} \boldsymbol{\alpha}_{nr}$ over the signature of φ' (royal and non-royal types), such that for any α_1, α_2 (possibly $\alpha_1 = \alpha_2$) such that $\alpha_1 \in \boldsymbol{\alpha}$ and $\alpha_2 \in \boldsymbol{\alpha}_{nr}$ there is a 2-type β such that $\beta{\restriction}x_1 = \alpha_1$ and $\beta{\restriction}x_2 = \alpha_2$, and β does not violate the universal conjuncts of φ'.
3. **Guess** structures $\mathfrak{K}$, $\mathfrak{C}$ of size at most 2^n and $2^n \cdot n^2$, resp., with $\mathfrak{K}$ being a substructure of $\mathfrak{C}$, such that (i) $\boldsymbol{\alpha}[K] = \boldsymbol{\alpha}_r$, (ii) $\boldsymbol{\alpha}[C \setminus K] \subseteq \boldsymbol{\alpha}_{nr}$, (iii) each element of K has all the required witness structures for $\forall\exists$ conjuncts of φ' in $\mathfrak{C}$, and (iv) universal conjuncts of φ' are not violated in $\mathfrak{C}$.
4. **Universally choose** an element $c \in C \setminus K$ and a conjunct ψ of φ' of type $\forall\exists$. Set $\mathfrak{F} := \mathfrak{C}{\restriction}(K \cup \{c\})$.
5. Set $\mathit{Counter} := 0$.

6. **Guess** an extension $\mathfrak{D}$ of $\mathfrak{F}$, with universe $D = K \cup \{c\} \cup \{a_1, \ldots, a_t\}$, such that (i) $\mathrm{tp}^{\mathfrak{D}}(a_i) \in \boldsymbol{\alpha}_{nr}$ for all i, (ii) for some $k_1, \ldots, k_s \in K$ the structure $\mathfrak{W} = \mathfrak{D} \restriction \{c, k_1, \ldots, k_s, a_1, \ldots, a_t\}$ is a witness structure for c and ψ, (iii) universal conjuncts of φ' are not violated in $\mathfrak{D}$. If $t = 0$ then **accept**.
7. **Universally choose** a new value for c from $\{a_1, \ldots, a_t\}$ and a conjunct ψ of φ' of the form $\forall\exists$. Set $\mathfrak{F} := \mathfrak{F}\restriction(K \cup \{c\})$.
8. $\mathit{Counter} := \mathit{Counter} + 1$
9. If $\mathit{Counter} < 2^{2^{2n^2}}$ then goto 6 else **accept**.

Let us first note that exponential space is sufficient to perform the above algorithm. By Lemma 1 we have that n is bounded polynomially in $|\varphi|$. The number of 1-types in $\boldsymbol{\alpha}$ is also bounded by 2^n, as a 1-type is determined by a subset of the signature. For some pairs of 1-types we need to guess a 2-type whose description is exponential (there are at most 2^n tuples of length not greater than n consisting of a pair of elements, and at most n relation symbols). The size of the structure $\mathfrak{C}$ guessed in Step 4 is explicitly required to be exponential in n. Also its description requires only exponentially many bits (recall that the arity of all relations is bounded by n). Analogously we can bound the size of structures $\mathfrak{D}$ guessed in Step 6. Finally, the value of $\mathit{Counter}$ is bounded doubly exponentially, so it also can be written using exponentially many bits.

Now we argue that the procedure accepts its input φ iff φ is satisfiable. Assume first that the procedure accepts φ. We show that then φ' (and thus, by Lemma 1, also φ) has a model. Consider an accepting run of the procedure. We may assume w.l.o.g. that this run is uniform, that is, when entering step 6, in configurations differing only in the values of $\mathit{Counter}$ (but with isomorphic $\mathfrak{F}$s) it makes the same (isomorphic) guesses of $\mathfrak{D}$. Then the modification of this procedure in which Step 9 is replaced just by 'Goto 6' can run infinitely (if necessary) without clashes. Indeed if the value $\mathit{Counter} = 2^{2^{2n^2}}$ is reached we have a guarantee that the $\mathfrak{K}$-type of the current c appeared before in the computation (cf. our estimations on the size of the small model constructed in the proof of Thm. 2, in particular on the number of $\mathfrak{K}$-types). We can construct a model for φ' starting from the substructure $\mathfrak{C}$ guessed in Step 4, and then providing witness structures for all conjuncts of the form $\forall\exists$ of φ' and elements c in accordance with guesses of $\mathfrak{D}$ is Step 6 (we add fresh copies of elements $a_1, \ldots, a_t$ and make the structure on the union of K, $\{c\}$ and the set of the newly added elements isomorphic to $\mathfrak{D}$). We complete the (usually infinite) structure as in Step *Completing the structure* of the small model construction from the proof of Thm. 2 using the 2-types guaranteed in Step 1. As in that proof we can also show that the constructed structure is a model of φ'.

Conversely, assume that φ has a model $\mathfrak{A}^*$. Nondeterministically compute its normal form φ' and let $\mathfrak{A} \models \varphi'$ be an expansion of $\mathfrak{A}^*$ guaranteed by Lemma 1. Let $\mathfrak{B}$ be a model of φ' constructed as in the proof of Thm. 2, starting from $\mathfrak{A}$. W can now make all the guesses of our procedure in accordance with $\mathfrak{B}$: denoting $K_{\mathfrak{B}}$ and $C_{\mathfrak{B}}$ the set of kings and a court of $\mathfrak{B}$, resp., we set $\boldsymbol{\alpha}_r := \boldsymbol{\alpha}[K_{\mathfrak{B}}]$, $\boldsymbol{\alpha}_{nr} := \boldsymbol{\alpha}[B \setminus K_{\mathfrak{B}}]$, $\mathfrak{K} := \mathfrak{K}_{\mathfrak{B}}$, $\mathfrak{C} := \mathfrak{C}_{\mathfrak{B}}$. Then in the loop 6-9, when a structure $\mathfrak{D}$ containing a witness structure for c and ψ is going to be guessed we choose an element $c' \in \mathfrak{B}$ such that the $\mathfrak{K}$-types of c' in $\mathfrak{B}$ and c in $\mathfrak{F}$ are identical and find a witness structure for c' and ψ in $\mathfrak{B}$. We set $\mathfrak{D}$ to be isomorphic to the restriction of $\mathfrak{B}$ to the union of $K_{\mathfrak{B}}$ and this witness structure. This strategy naturally leads to acceptance.

Lower bound in (i). We encode computations of an alternating Turing machine M working in exponential space on its input $\bar{a} = a_{i_0} \ldots a_{i_{n-1}}$.

The general idea of the proof is not far from the ideas used in the proofs of the 2-ExpTime-lower bound for GF [7] and 2-NExpTime-lower bound for TGF with constants [15]. We must, however, be careful to avoid quantification leaving more than one variable free, which happens in both the above-mentioned proofs. *E.g.*, in [7] configurations of a Turing machine are encoded by pairs of elements a_1, a_2; concretely, by the truth-values of some relations of arity $O(n)$ on tuples consisting of a_1, a_2. To enforce existence of successor configurations quantification leaving two free variables is needed there.

We assume that M has states $s_0, s_1, \ldots, s_k$, where s_0 is the initial state, s_{k-1} is the only accepting state, and s_k is the only rejecting state. The alphabet of M consists of letters $a_0, \ldots, a_l$ where a_0 represents *blank*. Without loss of generality we assume that M has precisely two possible moves in every configuration, that on its every computation path it enters the accepting or rejecting state no later than in 2^{2^n}-th step, and then, after reaching such final state, does not stop but works infinitely in a trivial way, without changing its configuration.

For $i = 0, \ldots, k$ we use a predicate S_i, for $i = 0, \ldots, l$ we use a predicate A_i and to describe the head position we use a predicate H. Each of the S_i, A_i and H is of arity $1 + n$.

We enforce the existence of two kings, called *zero* and *one*, marked, resp., by unary predicates Z and O. They will also be called *bits*, serve as binary digits and will be used to encode the numbers of tape cells.

$$\exists x(Z(x) \wedge \neg O(x)) \wedge \forall xy(Z(x) \wedge Z(y) \rightarrow x = y) \tag{2}$$

$$\exists x(O(x) \wedge \neg Z(x)) \wedge \forall xy(O(x) \wedge O(y) \rightarrow x = y) \tag{3}$$

The idea is that every element of a model encodes a configuration of M in its relation to tuples of bits of size n. Such a tuple of bits $\bar{b}$ can be naturally read as a number in the range $[0, \ldots, 2^n - 1]$. Let us think that $A_i(c, \bar{b})$ means that in the configuration encoded by c, tape cell $\bar{b}$ contains a_i, $H(c, \bar{b})$ denotes that this tape cell is scanned by the head and, for a cell observed by the head, $S_i(c, \bar{b})$ means that M is in state s_i.

To be able to speak about properties of configurations of M in TGF_1 we introduce a predicate C of arity $1 + 2n$, which will be made true at least for all tuples consisting of an arbitrary element of a model followed by $2n$ bits. We first say that, for any $0 \leq i < 2n$, C holds for some tuple consisting of i ones and $2n - i$ zeros, and then propagate C to all relevant tuples, using the fact that the pair of permutations $(2, 1, 3, \ldots, 2n)$ and $(2, 3, \ldots, 2n, 1)$ generates the whole permutation group S_{2n}. Below $\bar{z} = z_{2n-1}, z_{2n-2}, \ldots, z_1, z_0$.

$$\begin{aligned}\forall x \exists t_1 t_0 (O(t_1) \wedge Z(t_0) \wedge & C(x, t_0, t_0, t_0, \ldots, t_0) \wedge \\ & C(x, t_1, t_0, t_0 \ldots, t_0) \wedge \\ & C(x, t_1, t_1, t_0, \ldots, t_0) \wedge \ldots \wedge \\ & C(x, t_1, t_1, t_1, \ldots, t_1))\end{aligned} \tag{4}$$

$$\forall x \bar{z} (C(x, \bar{z}) \rightarrow C(x, z_{2n-2}, z_{2n-1}, z_{2n-3}, \ldots, z_0) \wedge C(x, z_{2n-2}, z_{2n-3}, \ldots, z_0, z_{n-1})) \tag{5}$$

We use a convention that $\bar{u}, \bar{v}, \bar{w}$ are tuples of variables of size n, $\bar{u} = u_{n-1}, \ldots, u_0$ and analogously for $\bar{v}$ and $\bar{w}$. We introduce abbreviations, $\lambda^{\neq}(\bar{u}, \bar{v})$ and $\lambda^{+1}(\bar{u}, \bar{v})$ for quantifier-free formulas of size polynomial in n. The former is intended to say that the numbers encoded by $\bar{u}$ and $\bar{v}$ differ, the latter – that the number encoded by $\bar{v}$ is greater by one than the number encoded by $\bar{u}$. *E.g.*, $\lambda^{+1}(\bar{u}, \bar{v})$ can be defined as

$$\bigvee_{0 \leq i < n} (Z(u_i) \wedge O(v_i) \wedge \bigwedge_{j<i} (O(u_j) \wedge Z(v_j)) \wedge \bigwedge_{j>i} (O(u_j) \leftrightarrow Z(v_j))) \tag{6}$$

Analogously, we use $\lambda^i(\bar{u})$ and $\lambda^{\geq i}(\bar{u})$ for formulas saying that the number encoded by $\bar{u}$ is, resp., equal to i and greater or equal i. Again, they can be defined in a standard way by quantifier-free, polynomially bounded formulas.

Now we ensure that every element properly encodes a configuration. The following formulas say that, resp., there is a tape cell scanned by the head, there is at most one such cell, this cell carries also information about the state, and every tape cell contains precisely a single letter. Below $\dot{\bigvee}_i \psi_i$ is an easily definable shorthand for "exactly one of the ψ_i holds".

$$\forall x \exists \bar{u} (H(x, \bar{u}) \wedge \bigwedge_i (O(u_i) \vee Z(u_i)) \tag{7}$$

$$\forall x \bar{u} \bar{v} (C(x, \bar{u}, \bar{v}) \to H(x, \bar{u}) \wedge \lambda^{\neq}(\bar{u}, \bar{v}) \to \neg H(x, \bar{v})) \tag{8}$$

$$\forall x \bar{u} (H(x, \bar{u}) \to \dot{\bigvee}_i S_i(x, \bar{u})) \tag{9}$$

$$\forall x \bar{u} (C(x, \bar{u}, \bar{u}) \to \dot{\bigvee}_i A_i(x, \bar{u})) \tag{10}$$

We then say that every element has two successors, and, using the trick with permutations prepare appropriate guards. Predicates $Succ_i$ are of arity $2 + 3n$. For $i = 1, 2$ we write:

$$\begin{aligned} \forall x \exists y t_1 t_0 (O(t_1) \wedge Z(t_0) \wedge & Succ_i(x, y, t_0, t_0, t_0, \ldots, t_0) \wedge \\ & Succ_i(x, y, t_1, t_0, t_0 \ldots, t_0) \wedge \\ & Succ_i(x, y, t_1, t_1, t_0, \ldots, t_0) \wedge \ldots \wedge \\ & Succ_i(x, y, t_1, t_1, t_1, \ldots, t_1)) \end{aligned} \tag{11}$$

$$\begin{aligned} & \forall x y \bar{t} (Succ_i(x, y, \bar{t}) \to \\ & \quad Succ_i(x, y, t_{3n-2}, t_{3n-1}, t_{3n-3}, \ldots, t_0) \wedge Succ_i(x, y, t_{3n-2}, t_{3n-3}, \ldots, t_0, t_{3n-1})) \end{aligned} \tag{12}$$

We next describe the computations of M on $\bar{a}$. First we say that the letter at a tape cell not scanned by the head does not change in the successor configurations. For $i = 1, 2$:

$$\forall x y \bar{u} ((Succ_i(x, y, \bar{u}, \bar{u}, \bar{u}) \to \neg H(x, \bar{u}) \to \bigwedge_i (A_i(x, \bar{u}) \to A_i(y, \bar{u}))) \tag{13}$$

Consider now existential moves. Assume that in an existential state s_i, reading a letter a_j the machine has two possible transitions: $(s_{i'}, a_{j'}, \to)$ and $(s_{i''}, a_{j''}, \leftarrow)$. Then we write:

$$\begin{aligned} & \forall x y \bar{u} \bar{v} \bar{w} (Succ_1(x, y, \bar{u}, \bar{v}, \bar{w}) \to H(x, \bar{u}) \wedge S_i(x, \bar{u}) \wedge A_j(x, \bar{u}) \wedge \lambda^{+1}(\bar{u}, \bar{v}) \wedge \lambda^{+1}(\bar{w}, \bar{u}) \to \\ & \quad (H(y, \bar{v}) \wedge S_{i'}(y, \bar{v}) \wedge A_{j'}(y, \bar{u})) \vee (H(y, \bar{w}) \wedge S_{i''}(y, \bar{w}) \wedge A_{j''}(y, \bar{u}))) \end{aligned} \tag{14}$$

Similarly, assume that M has moves as above in a universal state s_i. We write:

$$\begin{aligned} & \forall x y \bar{u} \bar{v} \bar{v} (Succ_1(x, y, \bar{u}, \bar{v}, \bar{v}) \to \\ & \quad H(x, \bar{u}) \wedge S_i(x, \bar{u}) \wedge A_j(x, \bar{u}) \wedge \lambda^{+1}(\bar{u}, \bar{v}) \to H(y, \bar{v}) \wedge S_{i'}(y, \bar{v}) \wedge A_{j'}(y, \bar{u})) \end{aligned} \tag{15}$$

$$\begin{aligned} & \forall x y \bar{u} \bar{w} \bar{w} (Succ_2(x, y, \bar{u}, \bar{w}, \bar{w}) \to \\ & \quad H(x, \bar{u}) \wedge S_i(x, \bar{u}) \wedge A_j(x, \bar{u}) \wedge \lambda^{+1}(\bar{w}, \bar{u}) \to H(y, \bar{w}) \wedge S_{i'}(y, \bar{w}) \wedge A_{j'}(y, \bar{u})) \end{aligned} \tag{16}$$

We finally say that a model does not contain a configuration with the rejecting state and impose the existence of an element encoding the initial configuration.

$$\neg \exists x S_k(x) \wedge \exists x Init(x) \tag{17}$$

$$\begin{aligned} \forall x \bar{u} (C(x, \bar{u}) \to Init(x) \to (& \lambda^{=0}(\bar{u}) \to H(x, \bar{u}) \wedge S_0(x, \bar{u}) \wedge A_{i_0}(x, \bar{u})) \wedge \\ & (\lambda^{=1}(\bar{u}) \to A_{i_1}(x, \bar{u})) \wedge \ldots \wedge \\ & (\lambda^{=n-1}(\bar{u}) \to A_{i_{n-1}}(x, \bar{u})) \wedge \\ & (\lambda^{\geq n}(\bar{u}) \to A_0(x, \bar{u}))) \end{aligned} \tag{18}$$

Showing that M accepts $\bar{a}$ iff the constructed formula has a model is routine.

Upper bounds in (ii) and (iii). In both cases we have proved an exponential model property. Thus, to test satisfiability it suffices to guess an exponentially bounded structure and verify that it indeed is a model. More precisely, given a formula φ we nondeterministically convert it into normal form φ'. We guess an exponentially bounded model $\mathfrak{B}$ of φ' (again we remark that not only the universe of $\mathfrak{B}$ is bounded exponentially, but also the description of $\mathfrak{B}$, since we are dealing only with at most $|\varphi'|$ relations of arity at most $|\varphi'|$), and verify that it is indeed a model. The last task can be carried out in an exhaustive way: for each $b \in B$ and each conjunct of φ' of the form $\forall x \exists \bar{y} \psi'_i(x, \bar{y})$ guess which elements form a witness structure for b and this conjunct and check that they indeed form a required witness structure; for each conjunct $\forall \bar{x}(\gamma_i(\bar{x}) \to \psi_i(\bar{x}))$ enumerate all tuples $\bar{b}$ of elements of B such that $|\bar{b}| = |\bar{x}|$ and check that $\mathfrak{B} \models \gamma_i(\bar{b}) \to \psi_i(\bar{b})$. Proceed analogously with the conjunct $\forall xy \psi''(x, y)$.

Lower bounds in (ii) and (iii). It suffices to show NExpTime-lower bound for GF$_1$ without equality. As advertised in the Introduction, we even strengthen this result using only uniform formulas, that is we show NExpTime-hardness of the *uniform* one-dimensional guarded fragment being the intersection of GF and UF$_1$. For our current purposes it is sufficient to say that conjunctions of sentences $\exists \bar{x} \psi(\bar{x})$ and $\forall \bar{x} \psi(\bar{x})$ with quantifier-free ψ are uniform if all atoms of φ use either all variables of $\bar{x}$ or just one of them. We use only formulas of such kind. For a general definition of UF$_1$ see [9] or [13]. Our proof goes by an encoding of an exponential tiling problem and is given in the full version of this paper.

5 Variations on extensions of the guarded fragment

Let us see what happens when the ideas of one-dimensionality, tri-guardedness and their combination are applied to two extensions of the guarded fragment: the loosely guarded fragment, LGF, introduced by van Benthem [17], and the guarded negation fragment, GNFO, introduced by Bárány, ten Cate and Segoufin [3]. LGF is defined similarly to GF, but the notion of the guard is more liberal: in subformulas of the form $\exists \bar{y}(\gamma(\bar{x}, \bar{y}) \wedge \varphi(\bar{x}, \bar{y}))$ and $\forall \bar{y}(\gamma(\bar{x}, \bar{y}) \to \varphi(\bar{x}, \bar{y}))$ we do not require that γ is atomic but allow it to be a conjunction of atoms such that for every variable from $\bar{y}$ and every variable from $\bar{y} \cup \bar{x}$ there is an atom in γ containing both of them. In GNFO (atomic) guards are required not for quantifiers but for negated subformulas. For a more detailed definition of GNFO see [3].

One-dimensionality. First, let us see that the one-dimensionality decreases the complexity of LGF, similarly as in the case of GF, but does not affect the complexity of GNFO.

▶ **Theorem 4.**

(i) *The satisfiability (= finite satisfiability) problem for the one-dimensional* LGF, LGF$_1$, *is* NExpTime*-complete.* LGF$_1$ *has an exponential model property.*

(ii) *The satisfiability (= finite satisfiability) problem for the one-dimensional* GNFO *is* 2-ExpTime*-complete.*

To prove (i) we adjust the small model construction from the proof of Thm. 2, by using more copies of witness structures and refining the strategy of providing witnesses. The construction from the proof of Thm. 2 cannot be applied without any changes to the current scenario, as it may accidentally form some cliques of cardinality greater than 2 in the Gaifmann graph of the constructed model which then could work as loose guards and lead to a violation of some universal conjuncts of the input formula.

To see (ii) note that GNFO contains the unary negation fragment, UNFO, whose satisfiability problem is already 2-ExpTime-hard. UNFO is not one-dimensional but can be polynomially translated to its equivalent UN-normal form (ten Cate, Segoufin [16]), which is one-dimensional. The upper bound is inherited from the upper bound for full GNFO [3].

Tri-guardedness. Unfortunately, allowing for unguarded binary subformulas leads to undecidability already in the case of one-dimensional variants of LGF and GNFO.

▶ **Theorem 5.** *The (finite) satisfiability problems for the one-dimensional* LGF *or* GNFO*, with unguarded subformulas with two variables, even without equality, are undecidable.*

In the case of LGF_1, unguarded binary subformulas give the power of full one-dimensional fragment F_1. Indeed by adding a conjunct $\forall xy G^*(x, y)$ we would be able to guard any tuple of variables $x_1, \ldots, x_k$ by the conjunction $\bigwedge_{i \neq j} G^*(x_i, x_j)$. (A similar observation is present also in [15].) As the satisfiability problem for F_1 is undecidable [9] this gives the undecidability of the considered variation of LGF. For the one-dimensional GNFO, using unguarded negations of binary atoms one can express transitivity of binary relations: $\neg\exists xyz(Rxy \wedge Ryz \wedge \neg Rxz)$. One-dimensional GNFO contains the two-variable guarded fragment which becomes undecidable when extended by transitive relations (Kieroński [12], Kazakov [11]). Thus the claim follows.

6 Adding constants

Finally, we study the satisfiability problem for GF_1 and TGF_1 with constants. It turns out that in the presence of constants we lose neither the decidability nor the finite model property, however, the complexity increases. The following theorem completes Table 1.

▶ **Theorem 6.**

(i) *Every satisfiable formula in* TGF_1 *with constants has a finite model of size bounded doubly exponentially in its length.*

(ii) *The satisfiability (= finite satisfiability) problem for* GF_1 *with constants (with or without equality) is 2-*ExpTime*-complete.*

(iii) *The satisfiability (= finite satisfiability) problem for* TGF_1 *with constants (with or without equality) is 2-*NExpTime*-complete.*

It is not difficult to see that Lemma 1 holds for formulas with constants. Thus, to show (i) we can use a minor adaptation of our small model construction from the proof of Thm. 2. Indeed, interpretations of constants may be treated as kings. The number of $\mathfrak{K}$-types remains doubly exponential. The construction works then essentially without changes, we only remark that in step *Completing the structure*, when a 2-type for a pair of elements is chosen, we need to define the truth-values of all relations on tuples built out of these elements and constants. This way we get a doubly exponential bound on the size of models.

The upper bound in (ii) follows from the fact that full GF with constants is in 2-ExpTime [7].

The upper bound in (iii) follows from the fact that full TGF with constants is in 2-NExpTime [15]. We remark, however, that this upper bound for TGF is obtained without proving the finite model property, thus to justify the upper bound for finite satisfiability of TGF_1 we must refer to part (i) of Thm. 6.

The corresponding lower bounds in (ii) and (iii) are proved in the full version of this paper.

References

1 H. Andréka, J. van Benthem, and I. Németi. Modal Languages and Bounded Fragments of Predicate Logic. *Journal of Philosophical Logic*, 27:217–274, 1998.

2 V. Bárány, G. Gottlob, and M. Otto. Querying the Guarded Fragment. *Logical Methods in Computer Science*, 10(2), 2014.

3 V. Bárány, B. ten Cate, and L. Segoufin. Guarded Negation. *J. ACM*, 62(3):22, 2015.

4 P. Bourhis, M. Morak, and A. Pieris. Making Cross Products and Guarded Ontology Languages Compatible. In *International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 880–886, 2017.

5 A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

6 W. D. Goldfarb. The unsolvability of the Gödel class with identity. *J. Symb. Logic*, 49:1237–1252, 1984.

7 E. Grädel. On The Restraining Power of Guards. *J. Symb. Log.*, 64(4):1719–1742, 1999.

8 E. Grädel, P. Kolaitis, and M. Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

9 L. Hella and A. Kuusisto. One-dimensional Fragment of First-order Logic. In *Proceedings of Advances in Modal Logic, 2014*, pages 274–293, 2014.

10 A.S. Kahr, E.F. Moore, and H. Wang. Entscheidungsproblem reduced to the ∀∃∀ case. *Proc. Nat. Acad. Sci. U.S.A.*, 48:365–377, 1962.

11 Y. Kazakov. *Saturation-based decision procedures for extensions of the guarded fragment.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2006.

12 E. Kieroński. Results on the Guarded Fragment with Equivalence or Transitive Relations. In *Computer Science Logic*, volume 3634 of *LNCS*, pages 309–324. Springer, 2005.

13 E. Kieronski and A. Kuusisto. Complexity and Expressivity of Uniform One-Dimensional Fragment with Equality. In *MFCS. Proceedings, Part I*, pages 365–376, 2014.

14 M. Mortimer. On languages with two variables. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:135–140, 1975.

15 Sebastian Rudolph and Mantas Šimkus. The Triguarded Fragment of First-Order Logic. In *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 57 of *EPiC Series in Computing*, pages 604–619, 2018.

16 B. ten Cate and L. Segoufin. Unary negation. *Logical Methods in Comp. Sc.*, 9(3), 2013.

17 J. van Benthem. Dynamic bits and pieces. *ILLC Research Report*, 1997.

Finite Satisfiability of Unary Negation Fragment with Transitivity

Daniel Danielski
University of Wrocław, Poland

Emanuel Kieroński
University of Wrocław, Poland
kiero@cs.uni.wroc.pl

Abstract

We show that the finite satisfiability problem for the unary negation fragment with an arbitrary number of transitive relations is decidable and 2-ExpTime-complete. Our result actually holds for a more general setting in which one can require that some binary symbols are interpreted as arbitrary transitive relations, some as partial orders and some as equivalences. We also consider finite satisfiability of various extensions of our primary logic, in particular capturing the concepts of nominals and role hierarchies known from description logic. As the unary negation fragment can express unions of conjunctive queries, our results have interesting implications for the problem of finite query answering, both in the classical scenario and in the description logics setting.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases unary negation fragment, transitivity, finite satisfiability, finite open-world query answering, description logics

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.17

Related Version A full version of this paper is available at https://arxiv.org/abs/1809.03245.

Funding Supported by Polish National Science Centre grant No 2016/21/B/ST6/01444.

1 Introduction

Decidable fragments and unary negation. Searching for attractive fragments of first-order logic is an important theme in theoretical computer science. Successful examples of such fragments, with numerous applications, are modal and description logics. They have their own syntax, but naturally translate to first-order logic, via the *standard translation.* Several seminal decidable fragments of first-order logic were identified by preserving one particular restriction obeyed by this translation and dropping all the others. Important examples of such fragments are two-variable logic, FO^2, [25], the guarded fragment, GF, [2], and the fluted fragment, FF, [24, 22]. They restrict, respectively, the number of variables, the quantification pattern and the order of variables in which they appear as arguments of predicates. A more recent proposal [27] is the *unary negation fragment*, UNFO. This time we restrict the use of negations, allowing them only in front of subformulas with at most one free variable. UNFO turns out to retain many good algorithmic and model theoretic properties of modal logic, including the finite model property, a tree-like model property and the decidability of the satisfiability problem. We remark here that UNFO and GF have a common decidable generalization, *the guarded negation fragment*, GNFO, [5].

To justify the attractiveness of UNFO let us look at one of the crucial problems in database theory, open-world query answering. Given an (incomplete) set of facts $\mathfrak{D}$, a set of constraints $\mathcal{T}$ and a query q, check if $\mathfrak{D} \wedge \mathcal{T}$ entails q. Generally, this problem is undecidable, and to make it decidable one needs to restrict the class of queries and constraints. Widely investigated class of queries are (unions of) *conjunctive queries* – (disjunctions of) sentences of the form $\exists \bar{x}\psi(\bar{x})$ where ψ is a conjunction of atoms. An important class of constraints are *tuple generating dependencies*, TGDs, of the form $\forall \bar{x}\bar{y}(\psi(\bar{x},\bar{y}) \rightarrow \exists \bar{z}\psi'(\bar{y},\bar{z}))$, where ψ and ψ' are, again, conjunctions of atoms. Conjunctive query answering against arbitrary TGDs is still undecidable (see, e.g., [6]), so TGDs need to be restricted further. Several classes of TGDs making the problem decidable have been proposed. One interesting such class are *frontier-one* TGDs, in which the *frontier* of each dependency, $\bar{y}$, consists just of a single variable [4]. Frontier-one TGDs are a special case of frontier-guarded TGDs [3]. Checking whether $\mathfrak{D}$ and $\mathcal{T}$ entail q boils down to verifying (un)satisfiability of the formula $\mathfrak{D} \wedge \mathcal{T} \wedge \neg q$. It turns out that if $\mathcal{T}$ is a conjunction of frontier-one TGDs and q is a disjunction of conjunctive queries then the resulting formula belongs to UNFO.

Transitivity. A serious weakness of the expressive power of UNFO is that it cannot express transitivity of a binary relation, nor related properties like being an equivalence, a partial order or a linear order. This limitation becomes particularly important when database or knowledge representation applications are considered, as transitivity is a natural property in many real-life situations. Just consider relations like *greater-than* or *part-of*. This weakness is shared by FO^2, GF and FF. Thus, it is natural to think about their extensions, in which some distinguished binary symbols may be explicitly required to be interpreted as transitive relations. It turns out that FO^2, GF and FF do not cope well with transitivity, and the satisfiability problems for the obtained extensions are undecidable [15, 13, 23] (see also [10, 18, 17]). Some positive results were obtained for FO^2, GF and FF only when one transitive relation is available [21, 18, 23] or when some further syntactic restrictions are imposed [26].

UNFO is an exception here, since its satisfiability problem remains decidable in the presence of arbitrarily many transitive relations. This has been explicitly stated in [16], as a corollary from a stronger result that UNFO is decidable when extended by regular path expressions. Independently, the decidability of UNFO with transitivity, UNFO+$\mathcal{S}$, follows from [1], which deals with the decidability of a richer logic, the guarded negation fragment with transitive relations restricted to non-guard positions, which embeds UNFO+$\mathcal{S}$. From both papers the 2-ExpTime-completeness of UNFO+$\mathcal{S}$ can be inferred.

Our main results. A problem related to satisfiability is *finite satisfiability*, in which we ask about the existence of *finite* models. In computer science, the importance of decision procedures for finite satisfiability arises from the fact that most objects about which we may want to reason using logic, e.g., databases, are finite. Thus the ability of solving only general satisfiability may not be fully satisfactory. Both the above-mentioned decidability results implying the decidability of UNFO+$\mathcal{S}$ are obtained by employing tree-like model properties of the logics and then using automata techniques. Since tree-like unravelings of models are infinite, this approach works only for general satisfiability, and gives little insight into the decidability/complexity of finite satisfiability. In this paper we consider the finite satisfiability problem for UNFO+$\mathcal{S}$. Actually, we made a step in this direction already in our previous paper [7] (see [8] for its longer version) where we proved a related result that UNFO with equivalence relations, UNFO+EQ, has the finite model property and thus that its satisfiability and finite satisfiability problems coincide, both being 2-ExpTime-complete. Some ideas developed in [7] are extended and applied also here, even though UNFO+$\mathcal{S}$ does not have the finite model property which becomes evident when looking at the following formula with transitive T, $\forall x \exists y Txy \wedge \forall x \neg Txx$, satisfiable only in infinite models.

Our main contribution is demonstrating the decidability of finite satisfiability for UNFO+$\mathcal{S}$ and establishing its 2-ExpTime-completeness. En route we obtain a triply exponential bound on the size of minimal models of finitely satisfiable UNFO+$\mathcal{S}$ formulas. Actually, our results hold for a more general setting, in which some relations may be required to be interpreted as equivalences, some as partial orders, and some just as arbitrary transitive relations. Returning to database motivations, we get this way the decidability of the *finite* open-world query answering for unions of conjunctive queries against frontier-one TGDs with equivalences, partial orders and arbitrary transitive relations. By *finite open-world query answering* we mean the question if for given $\mathfrak{D}$, $\mathcal{T}$ and q, $\mathfrak{D}$ and $\mathcal{T}$ entail q over *finite* structures.

To the best of our knowledge, UNFO+$\mathcal{S}$ is the first logic which allows one to use arbitrarily many transitive relations, and, at the same time, to speak non-trivially about relations of arbitrary arities, whose finite satisfiability problem is shown decidable. In the case of related logics of this kind, like the guarded fragment with transitive guards [26], and the guarded negation fragment with transitive relations outside guards [1], the decidability was shown only for general satisfiability, and its finite version is open. (Finite satisfiability was shown decidable only for the *two-variable* guarded fragment with transitive guards [20]).

We believe that moving from UNFO+EQ from [7] to UNFO+$\mathcal{S}$ is an important improvement. Besides the fact that this requires strengthening our techniques and employing some new ideas, general transitive relations have stronger motivations than equivalences. In particular, it opens natural connections to the realm of description logics, DLs.

UNFO and expressive description logics. UNFO, via the above-mentioned standard translation, embeds the DL $\mathcal{ALC}$, as well as its extension by inverse roles ($\mathcal{I}$) and role intersections ($\sqcap$). Thus, having the ability of expressing conjunctive queries, we can use our results to solve the so-called *(finite) ontology mediated query answering* problem, (F)OMQA, for some DLs. This problem is a counterpart of (finite) open-world query answering: given a conjunctive query (or a union of conjunctive queries) and a knowledge base specified in a DL, check whether the query holds in every (finite) model of this knowledge base.

While there are quite a lot of results for OMQA, not much is known about FOMQA. In particular, for DLs with transitive roles ($\mathcal{S}$) the only positive results we are aware of are the ones obtained recently in [12], where the decidability and 2-ExpTime-completeness of FOMQA for the logics $\mathcal{SOI}$, $\mathcal{SIF}$ and $\mathcal{SOF}$ is shown. This is orthogonal to our results described above, since UNFO+$\mathcal{S}$ captures neither nominals ($\mathcal{O}$) nor functional roles ($\mathcal{F}$). On the other hand, we are able to express any positive boolean combinations of roles, including their intersection ($\sqcap$), which allows us to solve FOMQA, e.g., for the logic $\mathcal{SI}^{\sqcap}$. Moreover we can use non-trivially relations of arity greater than two.

It is an interesting question if our decidability result can be extended to capture some more expressive DLs. Unfortunately, we cannot hope for *number restrictions* ($\mathcal{Q}$ or $\mathcal{N}$) or even *functional roles* ($\mathcal{F}$), as satisfiability and finite satisfiability of UNFO (even without transitive relations) and two binary functional relations are undecidable. This is implicit in [27] (see the full version of this paper for an explicit proof). On the positive side, we show the decidability and 2-ExpTime-completeness of finite satisfiability of UNFO+$\mathcal{SOH}$, extending UNFO+$\mathcal{S}$ by constants (corresponding to *nominals* ($\mathcal{O}$)) and inclusions of binary relations (capturing *role hierarchies* ($\mathcal{H}$)). This is sufficient, in particular, to imply the decidability of FOMQA for the description logic $\mathcal{SHOI}^{\sqcap}$, which, up to our knowledge, is a new result.

Towards guarded negation fragment. We propose also another decidable extension of our basic logic, the *one-dimensional base-guarded negation fragment* with transitive relations on non-guard positions, BGNFO$_1$+$\mathcal{S}$. This is a non-trivial fragment of the already mentioned logic from [1]. After some rather easy adjustments, our constructions cover this bigger logic, however, it becomes undecidable when extended with inclusions of binary relations.

Organization of the paper. The rest of this paper is organized as follows. Section 2 contains definitions, basic facts and a high-level description of our decidability proof. As our constructions are rather complex, in the main body of the paper, Section 3, we explicitly process the restricted, two-variable case of our logic, for which our ideas can be presented more transparently. In Section 4 we just formulate the remaining results, leaving the details for the full version of this paper, which also contains the missing proofs from Sections 2 and 3. In Section 5 we conclude the paper.

2 Preliminaries

2.1 Logics, structures, types and functions

We employ standard terminology and notation from model theory. We refer to structures using Fraktur capital letters, and their domains using the corresponding Roman capitals. For a structure $\mathfrak{A}$ and $A' \subseteq A$ we use $\mathfrak{A}{\restriction}A'$ or $\mathfrak{A}'$ to denote the restriction of $\mathfrak{A}$ to A'.

The *unary negation fragment of first-order logic*, UNFO is defined by the following grammar [27]: $\varphi = B\bar{x} \mid x = y \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x \varphi \mid \neg\varphi(x)$, where, in the first clause, B represents any relational symbol, and, in the last clause, φ has no free variables besides (at most) x. An example formula not expressible in UNFO is $x \neq y$. We formally do not have universal quantification. However we allow ourselves to use $\forall \bar{x} \neg \varphi$ as an abbreviation for $\neg \exists \bar{x} \varphi$, for an UNFO formula φ. Note that frontier-one TGDs $\forall \bar{x} y(\psi(\bar{x}, y) \to \exists \bar{z} \psi'(y, \bar{z}))$ are in UNFO as they can be rewritten as $\neg \exists \bar{x} y(\psi(\bar{x}, y) \wedge \neg \exists \bar{z} \psi'(y, \bar{z}))$.

We mostly work with purely relational signatures (admitting constants only in some extensions of our main results) of the form $\sigma = \sigma_{\text{base}} \cup \sigma_{\text{dist}}$, where σ_{base} is the *base signature*, and σ_{dist} is the *distinguished signature*. We assume that $\sigma_{\text{dist}} = \{T_1, \ldots, T_{2k}\}$, with all the T_u binary, and intension that T_{2u} is interpreted as the inverse of T_{2u-1}. For every $1 \leq u \leq k$ we sometimes write T_{2u}^{-1} for T_{2u-1}, and T_{2u-1}^{-1} for T_{2u}. We say that a subset $\mathcal{E}$ of σ_{dist} is *closed under inverses* if, for every $1 \leq u \leq 2k$, we have $T_u \in \mathcal{E}$ iff $T_u^{-1} \in \mathcal{E}$. Note that $\mathcal{E}$ is closed under inverses iff $\sigma_{\text{dist}} \setminus \mathcal{E}$ is closed under inverses. Given a formula φ we denote by σ_φ the signature induced by φ, *i.e.*, the minimal signature, with its distinguished part closed under inverses, containing all symbols from φ.

The *unary negation fragment with transitive relations*, UNFO+$\mathcal{S}$, is defined by the same grammar as UNFO, however when satisfiability of its formulas is considered, we restrict the class of admissible models to those that interpret all symbols from σ_{dist} as transitive relations and, additionally, for each u, interpret T_{2u} as the inverse of T_{2u-1}. The latter condition is intended to simplify the presentation, and is imposed without loss of generality. In our constructions we sometimes consider some auxiliary structures in which symbols from σ_{dist} are not necessarily interpreted as transitive relations (but the pairs T_{2u-1}, T_{2u} are always interpreted as inverses of each other).

An *(atomic) k-type* over a signature σ is a maximal satisfiable set of literals (atoms and negated atoms) over σ with variables $x_1, \ldots, x_k$. We often identify a k-type with the conjunction of its elements. We are mostly interested in 1- and 2-types. Given a σ-structure $\mathfrak{A}$ and $a, b \in A$ we denote by $\text{atp}^{\mathfrak{A}}(a)$ the 1-type *realized* by a, that is the unique 1-type $\alpha(x_1)$ such that $\mathfrak{A} \models \alpha(a)$, and by $\text{atp}^{\mathfrak{A}}(a, b)$ the unique 2-type $\beta(x_1, x_2)$ such that $\mathfrak{A} \models \beta(a, b)$.

We use various functions in our paper. Given a function $f : A \to B$ we denote by $\text{Rng} f$ its range, by $\text{Dom} f$ its domain, and by $f{\restriction}A_0$ the restriction of f to $A_0 \subseteq A$.

2.2 Normal form, witnesses and basic facts

We say that an UNFO+$\mathcal{S}$ formula is in Scott-normal form if it is of the shape

$$\forall x_1, \ldots, x_t \neg\varphi_0(\bar{x}) \wedge \bigwedge_{i=1}^{m} \forall x \exists \bar{y} \varphi_i(x, \bar{y}) \tag{1}$$

where each φ_i is a UNFO+$\mathcal{S}$ quantifier-free formula and φ_0 is additionally in negation normal form (NNF). A similar normal form for UNFO was introduced in the bachelor's thesis [9]. By a straightforward adaptation of Scott's translation for FO^2 [25] one can translate in polynomial time any UNFO+$\mathcal{S}$ formula to a formula in normal form, in such a way that both are satisfiable over the same domains. This allows us, when dealing with decidability/complexity issues for UNFO+$\mathcal{S}$, or when considering the size of minimal finite models of formulas, to restrict attention to normal form formulas.

Given a structure $\mathfrak{A}$, a normal form formula φ as in (1) and elements $a, \bar{b}$ of A such that $\mathfrak{A} \models \varphi_i(a, \bar{b})$ we say that the elements of $\bar{b}$ are *witnesses* for a and φ_i and that $\mathfrak{A}\restriction\{a, \bar{b}\}$ is a *witness structure* for a and φ_i. Fix an element a. For every φ_i choose a witness structure $\mathfrak{W}_i$. Then the structure $\mathfrak{W} = \mathfrak{A}\restriction\{W_1 \cup \ldots \cup W_m\}$ is called a φ-*witness structure* for a.

We are going to present a construction which given an arbitrary finite model of a normal form UNFO+$\mathcal{S}$ formula φ builds a finite model of φ of a bounded size. The construction goes via several intermediate steps in which some tree-like models are produced. To argue that that they are still models of φ we use the following basic observation (we recall that t is the number of variables of the $\forall$-conjunct of φ).

▶ **Lemma 1.** *Let $\mathfrak{A}$ be a model of a normal form UNFO+$\mathcal{S}$ formula φ. Let $\mathfrak{A}'$ be a structure in which all symbols from σ_{dist} are interpreted as transitive relations, such that*

(a1) *for every $a' \in A'$ there is a φ-witness structure for a' in $\mathfrak{A}'$,*

(a2) *for every tuple $a'_1, \ldots, a'_t \in A'$ there is a homomorphism $\mathfrak{h} : \mathfrak{A}'\restriction\{a'_1, \ldots, a'_t\} \to \mathfrak{A}$ which preserves 1-types of elements.*

Then $\mathfrak{A}' \models \varphi$.

2.3 Plan of the small model construction

Our main goal is to show that finite satisfiability of UNFO+$\mathcal{S}$ formulas can be checked in 2-ExpTime. To this end we will introduce a natural notion of tree-like structures and a measure associating with transitive paths of such structures their so-called *ranks*. Intuitively, for a transitive relation T_i and a T_i-path π, the T_i-rank of π is the number of one-directional T_i-edges in π (a precise definition is given in Section 3.1). Then we show that having the following forms of models is equivalent for a normal form formula φ:

(f1) finite;

(f2) tree-like, with bounded ranks of transitive paths;

(f3) tree-like, with ranks of transitive paths bounded doubly exponentially in $|\varphi|$;

(f4) tree-like, with ranks of paths bounded doubly exponentially in $|\varphi|$, and regular (with doubly exponentially many non-isomorphic subtrees);

(f5) finite of size triply exponential in $|\varphi|$.

We will make the following steps: (f1) $\rightsquigarrow$ (f2), (f2) $\rightsquigarrow$ (f3), (f3) $\rightsquigarrow$ (f4), (f4) $\rightsquigarrow$ (f5). The step closing the circle, (f5) $\rightsquigarrow$ (f1) is trivial. In the two-variable case, we will omit the form (f4) and directly show (f3) $\rightsquigarrow$ (f5). Our 2-ExpTime-algorithm will look for models of the form (f3). Showing transitions leading from (f3) to (f5) justifies that its answers coincide indeed with the existence of *finite* models.

This scheme is similar to the one we used to show the finite model property for UNFO+EQ in [7]. In the main part of the construction from [7] we build bigger and bigger substructures in which some equivalence relations are total. The induction goes, roughly speaking, by the number of non-total equivalences in the substructure. Here we extend this approach to handle one-way transitive connections. It may be useful to briefly compare the case of UNFO+$\mathcal{S}$ and the case of UNFO+EQ.

First of all, if a given formula φ is from UNFO+EQ then we can start our constructions leading to a small finite model of φ from its arbitrary model, while if φ is in UNFO+$\mathcal{S}$ we start from a *finite* model of φ. A very simple step (f1) $\rightsquigarrow$ (f2) in both papers is, essentially, identical. The counterpart of step (f3) $\rightsquigarrow$ (f4) in the case of equivalences is slightly simpler, but the main differences lie in steps (f2) $\rightsquigarrow$ (f3) and (f4) $\rightsquigarrow$ (f5). The former, clearly, is not present at all in [7]. While the general idea in this step is quite standard, as we just use a kind of tree pruning, the details are rather delicate due to possible interactions among different transitive relations, and this step is, by no means, trivial. We refine here, in particular, the apparatus of *declarations* introduced in [7]. Regarding step (f4) $\rightsquigarrow$ (f5), the main construction there, in its single inductive step, has two phases: building the so-called *components* and then arranging them into a bigger structure. It is this first phase which is more complicated than in the corresponding step in [7]. Having components prepared we join them similarly as in [7].

3 The two-variable case

As in the case of unbounded number of variables we can restrict attention to normal form formulas, which in the two-variable case simplify to the standard Scott-normal form [25]:

$$\forall xy \neg\varphi_0(x,y) \wedge \bigwedge_{i=1}^{m} \forall x \exists y \varphi_i(x,y), \tag{2}$$

where all φ_i are quantifier-free UNFO^2+$\mathcal{S}$ formulas (in this restricted case it is not important whether φ_0 is in NNF or not). As is typical for two-variable logics we assume that formulas do not use relational symbols of arity greater than 2 (cf. [14]).

3.1 Tree pruning in the two-variable case

We use a standard notion of a (finite or infinite) rooted tree and related terminology. Additionally, any set consisting of a node and all its children is called a *family*. Any node b, except for the root and the leaves, belongs to two families: the one containing its parent, and the one containing its children, the latter called the *downward family of b*.

We say that a structure $\mathfrak{A}$ over a signature consisting of unary and binary symbols is a *light tree-like structure* if its nodes can be arranged into a rooted tree in such a way that if $\mathfrak{A} \models Baa'$ for some non-transitive relation symbol B then one of three conditions holds: $a = a'$, a is the parent of a' or a is a child of a', and if $\mathfrak{A} \models T_u aa'$ for some T_u then either $a = a'$ or there is a sequence of distinct nodes $a = a_0, a_1, \ldots, a_k = a'$ such that a_i and a_{i+1} are joined by an edge of the tree and $\mathfrak{A} \models T_u a_i a_{i+1}$. In other words, distant nodes in a light tree-like structure can be joined only by transitive connections, moreover, these transitive connections are just the transitive closures of connections inside families. For a light tree-like structure $\mathfrak{A}$ and $a \in A$ we denote by A_a the set of all nodes in the subtree rooted at a and by $\mathfrak{A}_a$ the corresponding substructure.

Let $\mathfrak{A}$ be a light tree-like structure. A sequence of nodes $a_1, \ldots, a_N \in A$ is a *downward path* in $\mathfrak{A}$ if for each i a_{i+1} is a child of a_i. A *downward-T_u-path* is a downward path such that for each i we have $\mathfrak{A} \models T_u a_i a_{i+1}$. The *$T_u$-rank* of a downward-$T_u$-path $\vec{a}$, $\mathfrak{r}_u^{\mathfrak{A}}(\vec{a})$, is the cardinality of the set $\{i : \mathfrak{A} \models \neg T_u a_{i+1} a_i\}$. The T_u-*rank* of an element $a \in A$ is defined as $\mathfrak{r}_u^{\mathfrak{A}}(a) = \sup\{\mathfrak{r}_u^{\mathfrak{A}}(\vec{a}) : \vec{a} = a, a_2, \ldots, a_N; \vec{a}$ is a downward-T_u-path$\}$. For an integer M, we say that $\mathfrak{A}$ has downward-T_u-paths *bounded by* M when for all $a \in A$ we have $\mathfrak{r}_u^{\mathfrak{A}}(a) \leq M$, and that $\mathfrak{A}$ has *transitive paths bounded by* M if it has downward-T_u-paths bounded by M for all u. Note that a downward-T_u-path bounded by M may have more than M nodes, as the symmetric T_u-connections do not increase the rank.

Given an arbitrary model $\mathfrak{A}$ of a normal form UNFO^2+$\mathcal{S}$ formula φ we can simply construct its light tree-like model of degree bounded by $|\varphi|$. We define a *light-φ-tree-like unraveling* $\mathfrak{A}'$ of $\mathfrak{A}$ and an associated function $\mathfrak{h} : A' \to A$ in the following way. $\mathfrak{A}'$ is divided into levels $L_0, L_1, \ldots$. Choose an arbitrary element $a \in A$ and add to level L_0 of A' an element a' such that $\text{atp}^{\mathfrak{A}'}(a') = \text{atp}^{\mathfrak{A}}(a)$; set $\mathfrak{h}(a') = a$. The element a' will be the only element of L_0 and will become the root of $\mathfrak{A}'$. Having defined L_i repeat the following for every $a' \in L_i$. For every j, if $\mathfrak{h}(a')$ is not a witness for φ_j and itself then choose in $\mathfrak{A}$ a witness b for $\mathfrak{h}(a')$ and φ_j. Add a fresh copy b' of b to L_{i+1}, make $\mathfrak{A}' \restriction \{a', b'\}$ isomorphic to $\mathfrak{A} \restriction \{\mathfrak{h}(a'), b\}$ and set $\mathfrak{h}(b') = b$. Complete the definition of $\mathfrak{A}'$ transitively closing all relations from σ_{dist}.

▶ **Lemma 2** ((f1) ⇝ (f2), light)**.** *Let $\mathfrak{A}$ be a finite model of a normal form UNFO^2+$\mathcal{S}$ formula φ. Let $\mathfrak{A}'$ be a light-φ-tree-like unraveling of $\mathfrak{A}$. Then $\mathfrak{A}' \models \varphi$ and $\mathfrak{A}'$ is a light tree-like structure of degree bounded by $|\varphi|$, and transitive paths bounded by $|A|$.*

Our next task is making the transition (f2) ⇝ (f3). For this purpose we introduce a notion of light declarations. It is closely related to a notion of declarations which will be used in the general case, but simpler than the latter. Fix a signature and let $\boldsymbol{\alpha}$ be the set of 1-types over this signature.

For $\mathcal{T} \subseteq \{T_1, \ldots, T_{2k}\}$ we write $\mathfrak{A} \models \mathcal{T}ab$ iff $\mathfrak{A} \models T_u ab$ for all $T_u \in \mathcal{T}$. A *light declaration* is a function of type $\mathcal{P}(\{T_1, \ldots, T_{2k}\}) \to \mathcal{P}(\boldsymbol{\alpha})$. Given a light tree-like structure $\mathfrak{A}$ and its node a we say that a *respects* a light declaration $\mathfrak{d}$ if for every $\mathcal{T}$, for every $\alpha \in \mathfrak{d}(\mathcal{T})$ there is **no** node $b \in A$ of 1-type α such that $\mathfrak{A} \models \mathcal{T}ab$. We denote by $\text{ldec}^{\mathfrak{A}}(a)$ the maximal light declaration respected by a. Formally, for every $\mathcal{T} \subseteq \{T_1, \ldots, T_{2k}\}$, $\text{ldec}^{\mathfrak{A}}(a)(\mathcal{T}) = \{\alpha :$ for every node b of type α we have $\neg\mathfrak{A} \models \mathcal{T}ab\}$. Intuitively, $\text{ldec}^{\mathfrak{A}}(a)$ says, for any combination of transitive relations, which 1-types have **no** realizations to which a is connected by this combination in $\mathfrak{A}$. Note that if a respects a light declaration $\mathfrak{d}$ then for any $\mathcal{T}$ we have $\mathfrak{d}(\mathcal{T}) \subseteq \text{ldec}^{\mathfrak{A}}(a)(\mathcal{T})$. We remark that it would be equivalent to define the light declarations without the negations, listing the 1-types that a given node **is** connected with, however we choose a version with negations to make them uniform with the corresponding (more complicated) notion in the general case, where negations are more convenient.

Now we define the *local consistency conditions (LCCs)* for a system of light declarations $(\mathfrak{d}_a)_{a \in A}$ assigned to all nodes of a tree-like structure $\mathfrak{A}$. Let F be the downward family of some node a. We say that the system *satisfies LCCs at* a if for every $a_1, a_2 \in F$ and for every $\mathcal{T}$ such that $\mathfrak{A} \models \mathcal{T}a_1a_2$ the following two conditions hold: **(ld1)** for every $\alpha \in \boldsymbol{\alpha}$, if $\alpha \in \mathfrak{d}_{a_1}(\mathcal{T})$ then $\alpha \in \mathfrak{d}_{a_2}(\mathcal{T})$, **(ld2)** $\text{atp}^{\mathfrak{A}}(a_2) \notin \mathfrak{d}_{a_1}(\mathcal{T})$. Given a light tree-like structure $\mathfrak{A}$ we say that a system of light declarations $(\mathfrak{d}_a)_{a \in A}$ is *locally consistent* if it satisfies LCCs at each $a \in A$ and is *globally consistent* if $\mathfrak{d}_a(\mathcal{T}) \subseteq \text{ldec}^{\mathfrak{A}}(a)(\mathcal{T})$ for each $a \in A$ and each $\mathcal{T}$. Note that the global consistency means that all nodes a respect their light declarations $\mathfrak{d}_a$. It is not difficult to see that local and global consistency play along in the following sense.

▶ **Lemma 3** (Local-global, light)**.** *Let $\mathfrak{A}$ be a light tree-like structure. Then, (i) if a system of light declarations $(\mathfrak{d}_a)_{a \in A}$ is locally consistent then it is globally consistent; and (ii) the* canonical *system of light declarations, $(\text{ldec}^{\mathfrak{A}}(a))_{a \in A}$, is locally consistent.*

Given a light tree-like structure $\mathfrak{A}$, by the *generalized type* of a node a of $\mathfrak{A}$ we will mean a pair $(\mathrm{ldec}^{\mathfrak{A}}(a), \mathrm{atp}^{\mathfrak{A}}(a))$, and denote it as $\mathrm{gtp}^{\mathfrak{A}}(a)$. We introduce a concept of top-down tree pruning. Let $\mathfrak{A}$ be a light tree-like structure. A *top-down tree pruning process* on $\mathfrak{A}$ has countably many steps $0, 1, 2, \ldots$, each of them producing a new light tree-like structure by removing some nodes from the previous one and naturally stitching together the surviving nodes. We emphasise that the universes of all structures build in this process are subsets of the universe of the original structure $\mathfrak{A}$. More specifically, we take $\mathfrak{A}_0 := \mathfrak{A}$, and having constructed $\mathfrak{A}_i$, $i \geq 0$ construct $\mathfrak{A}_{i+1}$ as follows. For every node a of $\mathfrak{A}_i$ of depth $i+1$ (we assume that the root has depth 0) either leave the subtree rooted at a untouched or replace it by a subtree rooted at some descendant b of a having in the original structure $\mathfrak{A}$ the same generalized type as a, and then transitively close all transitive relations. The result of the process is a naturally defined limit structure $\mathfrak{A}'$, in which the pair of elements a, b, of depth d_a and d_b respectively, has its 2-type taken from $\mathfrak{A}_{\max(d_a, d_b)}$. Note that this 2-type is not modified in the subsequent structures, so the definition is sound.

▶ **Lemma 4** (Tree-pruning, light). *Let $\mathfrak{A}$ be a light tree-like structure. Let $(\mathfrak{d}_a)_{a \in A}$ be the canonical system of light declarations on $\mathfrak{A}$, $\mathfrak{d}_a := \mathrm{ldec}^{\mathfrak{A}}(a)$. Let $\mathfrak{A}'$ be the result of a top-town tree pruning process on $\mathfrak{A}$. Then (i) the system of light declarations $(\mathfrak{d}_a)_{a \in A'}$ (the canonical declarations from $\mathfrak{A}$ of the nodes surviving the pruning process) in $\mathfrak{A}'$ is locally consistent, (ii) for any pair of elements $a, a' \in A'$ there is a homomorphism $\mathfrak{A} \restriction \{a, a'\} \to A$ preserving the 1-types; it also follows that (iii) for a normal form φ, if $\mathfrak{A}$ is a model of φ such that any node a has all its witnesses in its downward family then $\mathfrak{A}' \models \varphi$.*

It is not difficult to devise a strategy of top-down tree pruning leading to a model with short transitive paths in a simple scenario where only one transitive relation is present. With several transitive relations, however, a quite intricate strategy seems to be required. The main obstacle is that when decreasing the T_u-rank of an element a, for some u, we may accidentally increase the T_v-rank of a for some $v \neq u$. Nevertheless, an appropriate strategy exists (see the full version of this paper), which allows us to state:

▶ **Lemma 5** ((f2) ⇝ (f3), light). *Let φ be a normal form UNFO^2+$\mathcal{S}$ formula. Let $\mathfrak{A} \models \varphi$ be a light tree-like structure over signature σ_φ, with transitive paths bounded by some natural number M, such that each element has all the required witnesses in its downward family. Then φ has a light tree-like model with transitive paths bounded doubly exponentially in $|\varphi|$.*

3.2 Finite model construction in the two-variable case

In this section we show the following small model property. To this end, in particular, we will make the transition (f3) ⇝ (f5).

▶ **Theorem 6.** *Every finitely satisfiable two-variable UNFO+$\mathcal{S}$ formula φ has a finite model of size bounded triply exponentially in $|\varphi|$.*

Let us fix a finitely satisfiable normal form UNFO+$\mathcal{S}$ formula φ over a signature $\sigma_\varphi = \sigma_{\text{base}} \cup \sigma_{\text{dist}}$ for $\sigma_{\text{dist}} = \{T_1, \ldots, T_{2k}\}$. Denote by $\boldsymbol{\alpha}$ the set of 1-types over this signature. Fix a light tree-like model $\mathfrak{A} \models \varphi$, with linearly bounded degree and doubly exponentially bounded transitive paths (in this section we denote this bound by $\hat{M}_\varphi$), as guaranteed by Lemma 5. We show how to build a "small" finite model $\mathfrak{A}' \models \varphi$. For a set $\mathcal{E} \subseteq \sigma_{\text{dist}}$, closed under inverses, and $a \in A$ we denote by $[a]_{\mathcal{E}}$ the set consisting of a and all elements $b \in A$ such that $\mathfrak{A} \models T_u ab$ for all $T_u \in \mathcal{E}$. Note that $[a]_{\mathcal{E}}$ is either a singleton or each of the $T_u \in \mathcal{E}$ is total on $[a]_{\mathcal{E}}$, that is, for each $b_1, b_2 \in [a]_{\mathcal{E}}$ we have $\mathfrak{A} \models T_u b_1 b_2$ for all $T_u \in \mathcal{E}$. We note that $[a]_{\emptyset} = A$.

In our construction we inductively produce finite fragments of $\mathfrak{A}'$ corresponding to some (potentially infinite) classes $[a]_{\mathcal{E}}$ of $\mathfrak{A}$. Essentially, the induction goes downward on the size of $\mathcal{E}$. Intuitively, if a relation is total then it plays no important role, so we may forget about it during the construction. Every such fragment will be obtained by an appropriate arrangement of some number of basic building blocks, called *components*. Each of the components is obtained by some number of applications of the inductive assumption to situations in which a new pair of relations T_{2u-1}, T_{2u} is added to $\mathcal{E}$.

Let us formally state our inductive lemma. In this statement we do not explicitly include any bound on the size of promised finite models, but such a bound will be implicit in the proof and will be presented later. Recall that $\mathfrak{A}$ is the model fixed at the beginning of this subsection.

▶ **Lemma 7** (Main construction, light). *Let $a_0 \in A$ and let $\mathcal{E}_0 \subseteq \sigma_{dist}$ be closed under inverses, let $\mathcal{E}_{tot} := \sigma_{dist} \setminus \mathcal{E}_0$. Let $\mathfrak{A}_0 = \mathfrak{A}_{a_0} \restriction [a_0]_{\mathcal{E}_{tot}}$. Then there exist a finite structure $\mathfrak{A}'_0$, a function $\mathfrak{p} : A'_0 \to A_0$ and an element $a'_0 \in A'_0$, called the* origin *of $\mathfrak{A}'_0$, such that*

(b1) *A'_0 is a singleton or every symbol from $\mathcal{E}_{tot}$ is interpreted as the total relation on $\mathfrak{A}'_0$.*

(b2) *$\mathfrak{p}(a'_0) = a_0$.*

(b3) *For each $a' \in A'_0$ and each i, if $\mathfrak{p}(a')$ has a child being its witness for φ_i in $\mathfrak{A}_0$ then a' has a witness for φ_i in $\mathfrak{A}'_0$. Moreover, $\mathrm{atp}^{\mathfrak{A}'_0}(a') = \mathrm{atp}^{\mathfrak{A}_0}(\mathfrak{p}(a'))$.*

(b4) *For every pair $a', b' \in A'_0$ there exists a homomorphism $\mathfrak{h} : \mathfrak{A}'_0 \restriction \{a', b'\} \to \mathfrak{A}$ preserving* 1*-types such that $\mathfrak{h}(a') = \mathfrak{p}(a')$, and for any 1-type α and $\mathcal{T} \subseteq \{1, \ldots, 2k\}$, if $\mathfrak{A}'_0 \models \mathcal{T}a'b'$ and $\alpha \notin \mathrm{ldec}^{\mathfrak{A}}(\mathfrak{p}(b'))(\mathcal{T})$ then $\alpha \notin \mathrm{ldec}^{\mathfrak{A}}(\mathfrak{p}(a'))(\mathcal{T})$.*

Observe first that Lemma 7 indeed allows us to build a particular finite model of φ. Apply it to $\mathcal{E}_0 = \sigma_{\text{dist}}$ (which means that $\mathcal{E}_{tot} = \emptyset$ and $[a_0]_{\mathcal{E}_{tot}} = A$) and a_0 being the root of $\mathfrak{A}$ (which means that $\mathfrak{A}_0 = \mathfrak{A}$) and use Lemma 1 to see that the obtained structure $\mathfrak{A}'_0$ is a model of φ. Indeed, Condition (a1) of Lemma 1 follows directly from Condition (b3), as in this case $\mathfrak{p}(a')$ has all witnesses in $\mathfrak{A}_0$. Condition (a2) is directly implied by Condition (b4).

The proof of Lemma 7 goes by induction on l, where $l = |\mathcal{E}_0|/2$. In the base of induction, $l = 0$, we have $\mathcal{E}_{tot} = \sigma_{\text{dist}}$. Without loss of generality we may assume that the classes $[a]_{\mathcal{E}_{tot}}$ are singletons for all $a \in A$. (If this is not the case, we just add artificial transitive relations T_{2k+1} and T_{2k+2} both interpreted as the identity in $\mathfrak{A}$.) We simply take $\mathfrak{A}'_0 := \mathfrak{A}_0 = \mathfrak{A} \restriction \{a_0\}$ and set $\mathfrak{p}(a_0) = a_0$. It is readily verified that the conditions (b1)–(b4) are then satisfied.

For the inductive step assume that Lemma 7 holds for arbitrary $\mathcal{E}_0$ closed under inverses, of size $2(l-1) < 2k$. We show that then it holds for $\mathcal{E}_0$ of size $2l$. Take such $\mathcal{E}_0$, and assume, w.l.o.g., that $\mathcal{E}_0 = \{T_1, \ldots, T_{2l}\}$. In the next two subsections we present a construction of $\mathfrak{A}'_0$. We argue that it is correct in the full version of this paper. Finally we estimate the size of the produced models and establish the complexity of the finite satisfiability problem.

3.2.1 Pattern components

We plan to construct $\mathfrak{A}'_0$ out of basic building blocks called *components*. Each component will be an isomorphic copy of some pattern component.

Let $\boldsymbol{\gamma}[A_0]$ be the set of the generalized types realized in $\mathfrak{A}_0$. For every $\gamma \in \boldsymbol{\gamma}[A_0]$ we construct two pattern structures, a *pattern component* $\mathfrak{C}^\gamma$ and an *extended pattern component* $\mathfrak{G}^\gamma$. $\mathfrak{C}^\gamma$ is a finite structure whose universe is divided into $2l$ *layers* $L_1, \ldots, L_{2l}$. $\mathfrak{G}^\gamma$ extends $\mathfrak{C}^\gamma$ by an additional, *interface layer*, denoted L_{2l+1}. See the left part of Fig. 1. We now define $\mathfrak{G}^\gamma$, obtaining then $\mathfrak{C}^\gamma$ just by the restriction of $\mathfrak{G}^\gamma$ to non-interface layers.

Each non-interface layer L_i is further divided into *sublayers* $L_i^1, L_i^2, \ldots, L_i^{\hat{M}_\varphi+1}$. Additionally, in each sublayer L_i^j its initial part $L_i^{j,init}$ is distinguished. In particular, $L_1^{1,init}$ consists

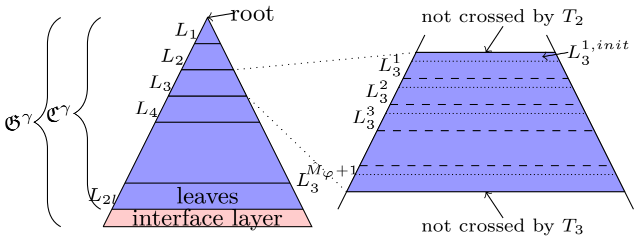


Figure 1 A schematic view of a component in the two-variable case.

of a single element called the *root*. The interface layer L_{2l+1} has no internal division but, for convenience, is sometimes referred to as $L_{2l+1}^{1,init}$. The elements of L_{2l} are called *leaves* and the elements of L_{2l+1} are called *interface elements*. See Fig. 1.

$\mathfrak{G}^\gamma$ will have a shape resembling a tree, with structures obtained by the inductive assumption as nodes, though it will not be tree-like in the sense of Section 3.1 (in particular, the internal structure of nodes may be complicated). All elements of $\mathfrak{G}^\gamma$, except for the interface elements, will have appropriate witnesses (those required by (b3)) provided. The crucial property we want to enforce is that the root of $\mathfrak{G}^\gamma$ will not be joined to its interface elements by any transitive path.

We remark that during the process of building a pattern component we do not yet apply the transitive closure to the distinguished relations. Postponing this step is not important from the point of view of the correctness of the construction, but will allow us for a more precise presentation of the proof of this correctness. Given a component $\mathfrak{C}$ (extended component $\mathfrak{G}$) we will sometimes denote by $\mathfrak{C}_+$ ($\mathfrak{G}_+$) the structure obtained from $\mathfrak{C}$ ($\mathfrak{G}$) by applying all the appropriate transitive closures.

The role of every non-interface layer L_u is, speaking informally, to *kill* T_u, that is to ensure that there will be no T_u-connections from L_u to L_{u+1}. See the right part of Fig. 1. The role of sublayers of L_u, on the other hand, is to decrease the T_u-rank of the patterns of elements. The purpose of the interface layer, L_{2l+1}, will be to connect the component with other components.

If γ is the generalized type of a_0 then take $a := a_0$; otherwise take as a any element of A_0 of generalized type γ. We begin the construction of $\mathfrak{G}^\gamma$ by defining $L_1^{1,init} = \{a'\}$ for a fresh a', setting $\text{atp}^{\mathfrak{G}^\gamma}(a') = \text{atp}^{\mathfrak{A}}(a)$ and $\mathfrak{p}(a') = a$.

Construction of a layer. Let $1 \le u \le 2l$. Assume we have defined layers $L_1, \ldots, L_{u-1}$, the initial part of sublayer L_u^1, $L_u^{1,init}$, and both the structure of $\mathfrak{G}^\gamma$ and the values of $\mathfrak{p}$ on $L_1 \cup \ldots \cup L_{u-1} \cup L_u^{1,init}$. We are going to kill T_u. We now expand $L_u^{1,init}$ to a full layer L_u.

Step 1: Subcomponents. Assume that we have defined sublayers $L_u^1, \ldots, L_u^{j,init}$, and both the structure of $\mathfrak{G}^\gamma$ and the values of $\mathfrak{p}$ on $L_1 \cup \ldots \cup L_{u-1} \cup L_u^1 \cup \ldots \cup L_u^{j,init}$. For each $b \in L_u^{j,init}$ perform independently the following procedure. Apply the inductive assumption to $\mathfrak{p}(b)$ and the set $\mathcal{E}_0 \setminus \{T_u, T_u^{-1}\}$ obtaining a structure $\mathfrak{B}_0$, its origin b_0 and a function $\mathfrak{p}_b : B_0 \to A_{\mathfrak{p}(b)} \cap [\mathfrak{p}(b)]_{\mathcal{E}_{tot} \cup \{T_u, T_u^{-1}\}} \subseteq A_0$ with $\mathfrak{p}_b(b_0) = \mathfrak{p}(b)$. Identify b_0 with b and add the remaining elements of $\mathfrak{B}_0$ to L_u^j, retaining the structure. Substructures $\mathfrak{B}_0$ of this kind will be called *subcomponents* (note that all appropriate relations are transitively closed in subcomponents). Extend $\mathfrak{p}$ so that $\mathfrak{p}{\restriction}B_0 = \mathfrak{p}_b$. This finishes the definition of L_u^j.

Step 2: Providing witnesses. For each $b \in L_u^j$ and $1 \leq s \leq m$ independently perform the following procedure. Let $\mathfrak{B}_0$ be the subcomponent created inductively in *Step 1*, such that $b \in B_0$. If $\mathfrak{p}(b)$ has a witness for $\varphi_s(x, y)$ in A_0 then we want to reproduce such a witness for b. Choose one such witness c (being a child of $\mathfrak{p}(b)$) for $\mathfrak{p}(b)$. Let us denote $\beta = \mathrm{atp}^{\mathfrak{A}}(\mathfrak{p}(b), c)$. If $\{T_u xy, T_u^{-1} xy\} \subseteq \beta$ then by Condition (b3) of the inductive assumption b already has an appropriate witness in the subcomponent $\mathfrak{B}_0$. So we do nothing in this case. If $T_u xy \in \beta$ and $T_u^{-1} xy \notin \beta$ then we add a copy c' of c to $L_u^{j+1,init}$; if $T_u xy \notin \beta$ then we add a copy c' of c to $L_{u+1}^{1,init}$. We join b with c' by β and set $\mathfrak{p}(c') = c$.

An attentive reader may be afraid that when adding witnesses for elements of the last sublayer $L_u^{\hat{M}_\varphi+1}$ of L_u we may want to add one of them to the non-existing layer $L_u^{\hat{M}_\varphi+2}$. There is however no such danger, which follows from the following claim.

▷ Claim 8. (i) Let $b \in L_u^{j,init}$ and let $\mathfrak{B}_0$ be the subcomponent created for b in *Step 1*. Then for all $b' \in \mathfrak{B}_0$ we have $\mathfrak{r}_u^{\mathfrak{A}}(\mathfrak{p}(b)) \geq \mathfrak{r}_u^{\mathfrak{A}}(\mathfrak{p}(b'))$. (ii) Let $b \in L_u^j$ and let $c' \in L_u^{j+1}$ be a witness created for b in *Step* 2. Then $\mathfrak{r}_u^{\mathfrak{A}}(\mathfrak{p}(b)) > \mathfrak{r}_u^{\mathfrak{A}}(\mathfrak{p}(c'))$.

Hence, when moving from L_u^j to L_u^{j+1} the T_u-ranks of pattern elements for the elements of these sublayers strictly decrease. Since these ranks are bounded by $\hat{M}_\varphi$, then, even if the T_u-ranks of the patterns of some elements of L_u^1 are equal to $\hat{M}_\varphi$, then, if $L_u^{\hat{M}_\varphi+1}$ is non-empty, the T_u-ranks of the patterns of its elements must be 0, which means that they cannot have witnesses connected to them one-directionally by T_u.

The construction of $\mathfrak{G}^\gamma$ is finished when layer L_{2l} is fully processed. We have added some elements to the interface layer, L_{2l+1}. Recall that it has only its "initial part".

3.2.2 Joining the components

In this section we take some number of copies of pattern components and arrange them into the desired structure $\mathfrak{A}'_0$, identifying interface elements of some components with the roots of some other. Some care is needed in this process in order to avoid any modifications of the internal structure of closures $\mathfrak{C}_+$ of components $\mathfrak{C}$, which could potentially result from the transitivity of relations. In particular we need to ensure that if for some u a pair of elements of a component $\mathfrak{C}$ is not connected by T_u inside $\mathfrak{C}$, then it will not become connected by a chain of T_u-edges external to $\mathfrak{C}$.

We create a pattern component $\mathfrak{C}^\gamma$ and its extension $\mathfrak{G}^\gamma$ for every $\gamma \in \boldsymbol{\gamma}[A_0]$. Let γ_{a_0} be the generalized type of a_0. Let *max* be the maximal number of interface elements across all the $\mathfrak{G}^\gamma$. For each $\mathfrak{G}^\gamma$ arbitrarily number its interface elements from 1 up to, maximally, *max*.

For each γ we take copies $\mathfrak{G}_{i,\gamma'}^{\gamma,g}$ of $\mathfrak{G}^\gamma$ for $g \in \{0, 1\}$, $1 \leq i \leq max$ and $\gamma' \in \boldsymbol{\gamma}[A_0]$. The parameter g is sometimes called a *color* (*red* or *blue*); it is convenient to think that the non-interface elements of $G_{i,\gamma'}^{\gamma,g}$ are of color g, but its interface elements have color $1-g$, cf. the left part of Fig. 1, as the latter will be later identified with the roots of some components of color $1 - g$. We import the numbering of the interface elements to these copies. We also take an additional copy $\mathfrak{G}_{\perp,\perp}^{\gamma_{a_0},0}$ of $\mathfrak{G}^{\gamma_{a_0}}$. Its root will become the origin of the whole $\mathfrak{A}'_0$. By $\mathfrak{C}_{i,\gamma'}^{\gamma,g}$ we denote the restriction of $\mathfrak{G}_{i,\gamma'}^{\gamma,g}$ to its non-interface elements.

For each γ, g consider extended components of the form $\mathfrak{G}_{\cdot,\cdot}^{\gamma,g}$, where the placeholders $\cdot$ can be substituted with any combination of proper indices. Perform the following procedure for each $1 \leq i \leq max$. Let b be the i-th interface element of any such extended component, let γ' be the generalized type of $\mathfrak{p}(b)$. Identify the i-th interface elements of all $\mathfrak{G}_{\cdot,\cdot}^{\gamma,g}$ with the root c_0 of $\mathfrak{G}_{i,\gamma}^{\gamma',1-g}$. Note that the values of $\mathfrak{p}(c_0)$ and $\mathfrak{p}(b)$ may differ. However, by construction, they have identical generalized types γ'. For the element c^* obtained in this identification step we define $\mathfrak{p}(c^*) = \mathfrak{p}(c_0)$.

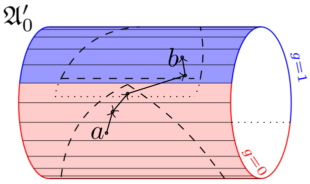


Figure 2 Viewing $\mathfrak{A}^0_0$ and $\mathfrak{A}'_0$ as placed on a cylindrical surface.

Define the graph of components used in the above construction, G^{comp}, by joining two components by an edge iff we identified an interface element of the extended version of one of them with the root of the other. Let A^0_0 be the union of the components accessible from $\mathfrak{C}^{\gamma_{a_0},0}_{\perp,\perp}$ in G^{comp} and let $\mathfrak{A}^0_0$ be the induced structure. Note that in $\mathfrak{A}^0_0$ we still do not take the transitive closures of relations. We define $\mathfrak{A}'_0$ by transitively closing all relations from $\sigma_{\rm dist}$ in $\mathfrak{A}^0_0$. Finally, we choose as the origin a'_0 of $\mathfrak{A}'_0$ the root of the pattern component $\mathfrak{C}^{\gamma_0,0}_{\perp,\perp}$.

We remark that it is sufficient to take as the universe of $\mathfrak{A}'_0$ the union of the universes of some components $\mathfrak{C}^{\cdot,\cdot}_{\cdot,\cdot}$, and not of their extended versions $\mathfrak{G}^{\cdot,\cdot}_{\cdot,\cdot}$ from which we started our construction, since the interface elements from these extended components were identified with some roots of other components.

For the correctness proof of our construction see the full version of this paper. In this proof it is helpful to think about $\mathfrak{A}^0_0$ and $\mathfrak{A}'_0$ as the structures placed on a cylindrical surface and divided into $4l$ *levels*, see Fig. 2. What is crucial, any transitive path in $\mathfrak{A}^0_0$ can cross at most one of the two borders between colors.

3.2.3 Size of models and complexity

By a rather routine calculation we can show that models produced in the proof of Lemma 7 are of size bounded triply exponentially in the length of input formulas. This finishes the proof of Thm. 6, which immediately gives the decidability of the finite satisfiability problem for UNFO^2+$\mathcal{S}$ and suggests a simple 3-NExpTime-procedure: guess a finite structure of size bounded triply exponentially in the size of input φ and verify that it is indeed a model of φ. We can however do better and show a doubly exponential upper bound matching the known complexity of the general satisfiability problem. For this we design an alternating exponential space algorithm searching for models of the form (f3). The lower bound can be obtained for the two-variable UNFO^2+$\mathcal{S}$ in the presence of one transitive relation by a straightforward adaptation of the lower bound proof for GF^2 with transitive guards [19].

▶ **Theorem 9.** *The finite satisfiability problem for UNFO^2+$\mathcal{S}$ is 2-*ExpTime*-complete.*

4 The general case and its further extensions

In the full version of this paper we generalize the ideas from Section 3 to show:

▶ **Theorem 10.** *The finite satisfiability problem for UNFO+$\mathcal{S}$ is 2-*ExpTime*-complete.*

We also obtain a triply exponential upper bound on the size of minimal finite models of finitely satisfiable formulas. The structure of the proofs is similar to the two-variable case, though some details are more complicated. In particular, we need to go through form (f4) of models: regular trees with bounded ranks of transitive paths. We also explain that in addition to general transitive relations we can use also equivalences and partial orders.

We further extend Thm. 10 by considering an extension, UNFO+$\mathcal{SOH}$, of UNFO+$\mathcal{S}$ by constants and inclusion of binary relations of the form $B_1 \subseteq B_2$, interpreted in a natural way: $\mathfrak{A} \models B_1 \subseteq B_2$ iff $\mathfrak{A} \models \forall xy(B_1xy \rightarrow B_2xy)$.

▶ **Theorem 11.** *The finite satisfiability problem for UNFO+$\mathcal{SOH}$ is 2-*ExpTime*-complete.*

As mentioned in the Introduction, UNFO+$\mathcal{SOH}$ captures several interesting description logics. This implies that we can solve FOMQA problem for them. In particular, we have the following corollary, which, up to our knowledge is the first decidability result for FOMQA in the case of a description logic with both transitive roles and role hierarchies.

▶ **Corollary 12.** *Finite ontology mediated query answering, FOMQA, for the description logic $\mathcal{SHOI}^{\sqcap}$ is decidable and 2-*ExpTime*-complete.*

$\mathcal{SHOI}^{\sqcap}$ and some related logics are considered, e.g., in [11]. For more about FOMQA for description logics with transitivity see [12]. For more about OMQA for description logics see, e.g., references in [12].

Somewhat orthogonally to the extensions motivated by description logics we consider the *base-guarded negation fragment with transitivity*, BGNFO+$\mathcal{S}$, for which the general satisfiability problem was shown decidable in [1]. We do not solve its finite satisfiability problem here, but, analogously to the extension with equivalence relations, UNFO+EQ [7], we are able to lift our results to its one-dimensional restriction, BGNFO$_1$+$\mathcal{S}$, admitting only formulas in which every maximal block of quantifiers leaves at most one variable free.

▶ **Theorem 13.** *The finite satisfiability problem for BGNFO$_1$+$\mathcal{S}$ is 2-*ExpTime*-complete.*

Surprisingly, in contrast to UNFO+$\mathcal{S}$, BGNFO$_1$+$\mathcal{S}$ becomes undecidable when extended by inclusions of binary relations.

5 Conclusions

We proved that the finite satisfiability problem for the unary negation fragment with transitive relations, UNFO+$\mathcal{S}$, is decidable and 2-ExpTime-complete, complementing this way the analogous result for the general satisfiability problem for this logic implied by two other papers. Further, we identified some decidable extensions of our base logic capturing the concepts of nominals and role hierarchies from description logics. We noted that our work has some interesting implications on the finite query answering problem both under the classical (open-world) database scenario as well as in the description logics setting.

One open question is the decidability of the finite satisfiability problem for the full logic BGNFO+$\mathcal{S}$ from [1]. We made a step in this direction here, by solving this problem for the one-dimensional restriction of that logic. Another question is if our techniques can be adapted to a setting in which we do not assert that some distinguished relations are transitive but where we can talk about the transitive closure of the binary relations, or, more generally, to the extension of UNFO with regular path expressions from [16].

We finally remark that we do not know if our small model construction, producing finite models of size bounded triply exponentially in the size of the input formulas, is optimal with respect to the size of models. The best we can do for the lower bound is to enforce models of doubly exponential size (actually, this can be done in UNFO even without transitive relations).

References

1 A. Amarilli, M. Benedikt, P. Bourhis, and M. Vanden Boom. Query Answering with Transitive and Linear-Ordered Data. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence, IJCAI 2016*, pages 893–899, 2016.

2 H. Andréka, J. van Benthem, and I. Németi. Modal Languages and Bounded Fragments of Predicate Logic. *Journal of Philosophical Logic*, 27:217–274, 1998.

3 J.-F. Baget, M. LeClere, and M.-L. Mugnier. Walking the Decidability Line for Rules with Existential Variables. In *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning, KR 2010*, pages 466–476, 2010.

4 J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. Extending Decidable Cases for Rules with Existential Variables. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, pages 677–682, 2009.

5 V. Bárány, B. ten Cate, and L. Segoufin. Guarded Negation. *J. ACM*, 62(3):22, 2015.

6 A. Calì, G. Gottlob, and M. Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.

7 D. Danielski and E. Kieroński. Unary negation fragment with equivalence relations has the finite model property. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 285–294, 2018.

8 D. Danielski and E. Kieronski. Unary negation fragment with equivalence relations has the finite model property. *CoRR*, abs/1802.01318, 2018. `arXiv:1802.01318`.

9 M. Dzieciołowski. Satisfability issues for unary negation logic. Bachelor's thesis, University of Wrocław, 2017.

10 H. Ganzinger, Ch. Meyer, and M. Veanes. The Two-Variable Guarded Fragment with Transitive Relations. In *14th Annual IEEE Symposium on Logic in Computer Science, LICS 1999*, pages 24–34, 1999.

11 B. Glimm and Y. Kazakov. Role Conjunctions in Expressive Description Logics. In *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008*, pages 391–405, 2008.

12 T. Gogacz, Y. A. Ibáñez-García, and F. Murlak. Finite Query Answering in Expressive Description Logics with Transitive Roles. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018.*, pages 369–378, 2018.

13 E. Grädel. On The Restraining Power of Guards. *J. Symb. Log.*, 64(4):1719–1742, 1999.

14 E. Grädel, P. Kolaitis, and M. Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

15 E. Grädel, M. Otto, and E. Rosen. Undecidability results on two-variable logics. *Archiv für Mathematische Logik und Grundlagenforschung*, 38(4-5):313–354, 1999.

16 J. Ch. Jung, C. Lutz, M. Martel, and T. Schneider. Querying the Unary Negation Fragment with Regular Path Expressions. In *International Conference on Database Theory, ICDT 2018*, pages 15:1–15:18, 2018.

17 Y. Kazakov. *Saturation-based decision procedures for extensions of the guarded fragment.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2006.

18 E. Kieroński. Results on the Guarded Fragment with Equivalence or Transitive Relations. In *Computer Science Logic*, volume 3634 of *LNCS*, pages 309–324. Springer, 2005.

19 E. Kieroński. On the complexity of the two-variable guarded fragment with transitive guards. *Inf. Comput.*, 204(11):1663–1703, 2006.

20 E. Kieroński and L. Tendera. Finite Satisfiability of the Two-Variable Guarded Fragment with Transitive Guards and Related Variants. *ACM Trans. Comput. Logic*, 19(2):8:1–8:34, 2018.

21 I. Pratt-Hartmann. The Finite Satisfiability Problem for Two-Variable, First-Order Logic with one Transitive Relation is Decidable. *Mathematical Logic Quarterly*, 2018.

22 I. Pratt-Hartmann, W. Szwast, and L. Tendera. The Fluted Fragment Revisited. *Journal of Symbolic Logic*, Forthcoming, 2019.

23 I. Pratt-Hartmann and L. Tendera. The Fluted Fragment with Transitivity. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019*, pages 18:1–18:15, 2019.

24 W. V. Quine. On the limits of decision. In *Proceedings of the 14th International Congress of Philosophy*, volume III, pages 57–62, 1969.

25 D. Scott. A decision method for validity of sentences in two variables. *Journal Symbolic Logic*, 27:477, 1962.

26 W. Szwast and L. Tendera. The guarded fragment with transitive guards. *Annals of Pure and Applied Logic*, 128:227–276, 2004.

27 B. ten Cate and L. Segoufin. Unary negation. *Logical Methods in Comp. Sc.*, 9(3), 2013.

The Fluted Fragment with Transitivity

Ian Pratt-Hartmann
University of Warsaw, Poland
University of Opole, Poland
University of Manchester, UK
ipratt@cs.man.ac.uk

Lidia Tendera
University of Opole, Poland
tendera@math.uni.opole.pl

Abstract

We study the satisfiability problem for the fluted fragment extended with transitive relations. We show that the logic enjoys the finite model property when only one transitive relation is available. On the other hand we show that the satisfiability problem is undecidable already for the two-variable fragment of the logic in the presence of three transitive relations.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Finite Model Theory

Keywords and phrases First-Order logic, Decidability, Satisfiability, Transitivity, Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.18

Related Version An extended version of the paper is available at `https://arxiv.org/abs/1906.09131`.

Funding This work is supported by the Polish National Science Centre grant 2018/31/B/ST6/03662.

1 Introduction

The *fluted fragment*, here denoted $\mathcal{FL}$, is a fragment of first-order logic in which, roughly speaking, the order of quantification of variables coincides with the order in which those variables appear as arguments of predicates. The allusion is presumably architectural: we are invited to think of arguments of predicates as being "lined up" in columns. The following formulas are sentences of $\mathcal{FL}$

$$\begin{array}{l}\text{No student admires every professor}\\ \forall x_1(\text{student}(x_1) \rightarrow \neg\forall x_2(\text{prof}(x_2) \rightarrow \text{admires}(x_1, x_2)))\end{array} \tag{1}$$

$$\begin{array}{l}\text{No lecturer introduces any professor to every student}\\ \forall x_1(\text{lecturer}(x_1) \rightarrow \neg\exists x_2(\text{prof}(x_2) \wedge \forall x_3(\text{student}(x_3) \rightarrow \text{intro}(x_1, x_2, x_3)))),\end{array} \tag{2}$$

with the "lining up" of variables illustrated in Fig. 1. By contrast, none of the formulas

$$\forall x_1.r(x_1, x_1)$$
$$\forall x_1\forall x_2(r(x_1, x_2) \rightarrow r(x_2, x_1))$$
$$\forall x_1\forall x_2\forall x_3(r(x_1, x_2) \wedge r(x_2, x_3) \rightarrow r(x_1, x_3)),$$

expressing, respectively, the reflexivity, symmetry and transitivity of the relation r, is fluted, as the atoms involved cannot be arranged so that their argument sequences "line up" in the fashion of Fig. 1.

The history of this fragment is somewhat tortuous. The basic idea of *fluted logic* can be traced to a paper given by W.V. Quine to the 1968 *International Congress of Philosophy* [19], in which the author defined the *homogeneous m-adic formulas*. Quine later relaxed this

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 18; pp. 18:1–18:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$\begin{array}{r} \forall x_1 \\ (\text{student}(x_1) \\ \to \neg\forall x_2 \\ (\text{prof}(x_2) \\ \to \text{admires}(x_1, x_2))) \end{array} \qquad\qquad \begin{array}{r} \forall x_1 \\ (\text{lecturer}(x_1) \\ \to \neg\exists x_2 \\ (\text{prof}(x_2) \\ \land\forall x_3 \\ (\text{student}(x_3) \end{array}$$

Figure 1 The "lining up" of variables in the fluted formulas (1) and (2); all quantification is executed on the right-most available column.

fragment, in the context of a discussion of predicate-functor logic, to what he called "fluted" quantificational schemata [20], claiming that the satisfiability problem for the relaxed fragment is decidable. The viability of the proof strategy sketched by Quine was explicitly called into question by Noah [12], and the subject then taken up by W.C. Purdy [17], who gave his own definition of "fluted formulas", proving decidability. It is questionable whether Purdy's reconstruction is faithful to Quine's intentions: the matter is clouded by differences in the definitions of predicate functors between between [12] and [20], both of which Purdy cites. In fact, Quine's original definition of "fluted" quantificational schemata appears to coincide with a logic introduced – apparently independently – by A. Herzig [6]. Rightly of wrongly, however, the name "fluted fragment" has now attached itself to Purdy's definition in [17]; and we shall continue to use it in that way in the present article. See Sec. 2 for a formal definition.

To complicate matters further, Purdy claimed in [18] that $\mathcal{FL}$ (i.e. the fluted fragment, in our sense, and his) has the exponential-sized model property: if a fluted formula φ is satisfiable, then it is satisfiable over a domain of size bounded by an exponential function of the number of symbols in φ. Purdy concluded that the satisfiability problem for $\mathcal{FL}$ is NExpTime-complete. These latter claims are false. It was shown in [14] that, although $\mathcal{FL}$ has the finite model property, there is no elementary bound on the sizes of the models required, and the satisfiability problem for $\mathcal{FL}$ is non-elementary. More precisely, define $\mathcal{FL}^m$ to be the subfragment of $\mathcal{FL}$ in which at most m variables (free or bound) appear. Then the satisfiability problem for $\mathcal{FL}^m$ is $\lfloor m/2 \rfloor$-NExpTime-hard for all $m \geq 2$ and in $(m-2)$-NExpTime for all $m \geq 3$ [15]. It follows that the satisfiability problem for $\mathcal{FL}$ is Tower-complete, in the framework of [21]. These results fix the exact complexity of satisfiability of $\mathcal{FL}^m$ for small values of m. Indeed, the satisfiability problem for FO^2, the two-variable fragment of first-order logic, is known to be NExpTime-complete [5], whence the corresponding problem for $\mathcal{FL}^2$ is certainly in NExpTime. Moreover, for $0 \leq m \leq 1$, $\mathcal{FL}^m$ coincides with the m-variable fragment of first-order logic, whence its satisfiability problem is NPTime-complete. Thus, taking 0-NExpTime to mean NPTime, we see that the satisfiability problem for $\mathcal{FL}^m$ is $\lfloor m/2 \rfloor$-NExpTime-complete, at least for $m \leq 4$.

The focus of the present paper is what happens when we add to the fluted fragment the ability to stipulate that certain designated binary relations are *transitive*, or are *equivalence relations*. The motivation comes from analogous results obtained for other decidable fragments of first-order logic. Consider basic propositional modal logic K. Under the standard translation into first-order logic (yielded by Kripke semantics), we can regard K as a fragment of first-order logic – indeed as a fragment of $\mathcal{FL}^2$. From basic modal logic K, we obtain the logic K4 under the supposition that the accessibility relation on possible worlds is transitive, and the logic S5 under the supposition that it is an equivalence relation: it is well-known that the satisfiability problems for K and K4 are PSpace-complete, whereas that for S5 is NPTime-complete [11]. (For analogous results on *graded* modal logic, see [7].) Closely related are also description logics (cf. [2]) with *role hierarchies* and *transitive roles*. In particular, the description logic $\mathcal{SH}$, which has the finite model property, is an ExpTime-complete fragment of $\mathcal{FL}$ with transitivity. Similar investigations have been carried out in respect of FO^2, which has the finite model property and whose satisfiability problem, as just mentioned, is NExpTime-complete. The finite model property is lost when one transitive relation or two equivalence relations are allowed. For equivalence, everything is known: the (finite) satisfiability problem for FO^2 in the presence of a single equivalence relation remains NExpTime-complete, but this increases to 2-NExpTime-complete in the presence of two equivalence relations [8, 9], and becomes undecidable with three. For transitivity, we have an incomplete picture: the *finite* satisfiability problem for FO^2 in the presence with a single transitive relation in decidable in 3-NExpTime [13], while the decidability of the satisfiability problem remains open (cf. [23]); the corresponding problems with two transitive relations are both undecidable [10].

Adding equivalence relations to the fluted fragment poses no new problems. Existing results on of FO^2 with two equivalence relations can be used to show that the satisfiability and finite satisfiability problems for $\mathcal{FL}$ (not just $\mathcal{FL}^2$) with *two* equivalence relations are decidable. Furthermore, the proof that the corresponding problems for FO^2 in the presence of *three* equivalence relations are undecidable can easily be seen to apply also to $\mathcal{FL}^2$. On the other hand, the situation with transitivity is much less clear.

We show in the sequel that $\mathcal{FL}$ in the presence of a single transitive relation has the finite model property. On the other hand, $\mathcal{FL}$ with three transitive relations admits axioms of infinity and the corresponding satisfiability problem is undecidable even for the intersection of $\mathcal{FL}^2$ with the *guarded fragment* [1] (and the same holds even when one of these transitive relations is the identity). The status of $\mathcal{FL}$ with just two transitive relations remains open. These can be contrasted with DLs where, to lose the finite model property, one needs to add to $\mathcal{S}$ either both *inverses* and *number restrictions*, or the *self* operator (none expressible in $\mathcal{FL}$). We also want to point to another paper in this volume [4] and the references therein where the impact of adding transitivity to the *unary negation fragment* is discussed.

2 Preliminaries

Unless explicitly stated to the contrary, the fragments of first-order logic considered here do not contain equality. We employ purely relational signatures, i.e. no individual constants or function symbols. We do, however, allow 0-ary relations (proposition letters).

Let $\bar{x}_\omega = x_1, x_2, \ldots$ be a fixed sequence of variables. We define the sets of formulas $\mathcal{FL}^{[m]}$ (for $m \geq 0$) by structural induction as follows: (i) any atom $\alpha(x_\ell, \ldots, x_m)$, where $x_\ell, \ldots, x_m$ is a contiguous subsequence of $\bar{x}_\omega$, is in $\mathcal{FL}^{[m]}$; (ii) $\mathcal{FL}^{[m]}$ is closed under boolean combinations; (iii) if φ is in $\mathcal{FL}^{[m+1]}$, then $\exists x_{m+1}\varphi$ and $\forall x_{m+1}\varphi$ are in $\mathcal{FL}^{[m]}$. The set of *fluted formulas* is defined as $\mathcal{FL} = \bigcup_{m\geq 0} \mathcal{FL}^{[m]}$. A *fluted sentence* is a fluted formula with no free variables. Thus, when forming Boolean combinations in the fluted fragment, all the combined formulas must have as their free variables some suffix of some prefix $x_1, \ldots, x_m$ of $\bar{x}_\omega$; and, when quantifying, only the last variable in this sequence may be bound. Note also that proposition letters (0-ary predicates) may be combined freely with formulas: if φ is in $\mathcal{FL}^{[m]}$, then so, for example, is $\varphi \wedge P$, where P is a proposition letter.

Denote by $\mathcal{FL}^m$ the sub-fragment of $\mathcal{FL}$ consisting of those formulas featuring at most m variables, free or bound. Do not confuse $\mathcal{FL}^m$ (the set of fluted formulas with at most m variables, free or bound) with $\mathcal{FL}^{[m]}$ (the set of fluted formulas with *free* variables $x_\ell, \ldots, x_m$).

These are, of course, quite different. For example, (1) is in $\mathcal{FL}^2$, and (2) is in $\mathcal{FL}^3$, but they are both in $\mathcal{FL}^{[0]}$. Note that $\mathcal{FL}^m$ cannot include predicates of arity greater than m.

For $m \geq 2$, denote by $\mathcal{FL}^m k\mathrm{T}$ the m-variable fluted fragment $\mathcal{FL}^m$ together with k distinguished transitive relations. In addition, denote by $\mathcal{FL}^2 k\mathrm{T}^u$ the sub-fragment of $\mathcal{FL}^2 k\mathrm{T}$ in which no binary predicates occur except the k distinguished transitive ones.

3 The decidability of fluted logic with one transitive relation

In this section, we show that the logic $\mathcal{FL}1\mathrm{T}$, the fluted fragment together with a single distinguished transitive relation $\mathfrak{t}$, has the finite model property. We proceed in stages. First, we show that $\mathcal{FL}^2 1\mathrm{T}^u$ has a doubly exponential-sized model property. Next, we show that $\mathcal{FL}^2 1\mathrm{T}$ has a triply exponential-sized model property, via an exponential-sized reduction to $\mathcal{FL}^2 1\mathrm{T}^u$. Finally, for $m \geq 2$, we provide an exponential-sized reduction of the satisfiability problem for $\mathcal{FL}^{m+1} 1\mathrm{T}$ to the corresponding problem for $\mathcal{FL}^m 1\mathrm{T}$, showing that, if the target of the reduction has a model of size N, then the source has a model of size $O(2^N)$. The satisfiability problems considered here will all have at least exponential complexity. Therefore, we may assume without loss of generality in this section that all signatures feature no 0-ary predicates, since their truth values can simply be guessed.

3.1 The logic $\mathcal{FL}^2 1\mathrm{T}^u$

Fix some signature Σ of unary predicates. We consider $\mathcal{FL}^2 1\mathrm{T}^u$-formulas over the signature $\Sigma \cup \{\mathfrak{t}\}$, where $\mathfrak{t}$ is the distinguished transitive predicate. (Thus, $\mathfrak{t} \notin \Sigma$.) By a 1-*type over* Σ, we mean a maximal consistent conjunction of literals $\pm p(x)$, where $p \in \Sigma$. If $\mathfrak{A}$ is a structure interpreting $\Sigma \cup \{\mathfrak{t}\}$, any element $a \in A$ satisfies a unique 1-type over Σ; we denote it $\mathrm{tp}^{\mathfrak{A}}[a]$. Since Σ will not vary, we typically omit reference to it when speaking of 1-types. We use the letters π and π' always to range over 1-types and μ always to range over arbitrary quantifier-free Σ-formulas involving just the variable x. We write $\pi(y)$ to indicate the result of substituting y everywhere for x in π, and similarly for π' and μ.

Call a $\mathcal{FL}^2 1\mathrm{T}^u$-formula over $\Sigma \cup \{\mathfrak{t}\}$ *basic* if it is of one of the forms

$$\exists x.\mu \qquad \forall x.\mu \qquad \forall x(\pi \to \exists y(\mu(y) \wedge \pm\mathfrak{t}(x,y))) \qquad \forall x(\pi \to \forall y(\pi'(y) \to \pm\mathfrak{t}(x,y))).$$

The following Lemma is a version of the familiar “Scott normal form” for $\mathcal{FL}^2$ from [22].

▶ **Lemma 1.** *Let φ be a $\mathcal{FL}^2 1\mathrm{T}^u$-sentence. There exists a set Ψ of basic $\mathcal{FL}^2 1\mathrm{T}^u$-formulas with the following properties: (i) $\models (\bigwedge \Psi) \to \varphi$; (ii) if φ has a model, then so has Ψ; (iii) $\|\Psi\|$ is bounded by a polynomial function of $\|\varphi\|$.*

We say that a *super-type over* Σ is a pair $\langle \pi, \Pi \rangle$, where π is a 1-type over Σ and Π a set of 1-types over Σ. If $\mathfrak{A}$ is a structure interpreting the signature $\Sigma \cup \{\mathfrak{t}\}$ and $a \in A$, the super-type of a in $\mathfrak{A}$, denoted $\mathrm{stp}^{\mathfrak{A}}[a]$, is the pair $\langle \mathrm{tp}^{\mathfrak{A}}[a], \Pi \rangle$, where $\Pi = \{\mathrm{tp}^{\mathfrak{A}}[b] \mid \mathfrak{A} \models \mathfrak{t}[a,b]\}$. Intuitively, a super-type is a description of an element in a structure specifying that element’s 1-type together with the 1-types of those elements to which it is related by $\mathfrak{t}$. If S is a set of super-types, we write $\mathrm{tp}(S) = \{\pi \mid \langle \pi, \Pi \rangle \in S \text{ for some } \Pi\}$. We usually omit Σ when speaking of super-types. By a *certificate*, we mean a pair $C = (S, \ll)$, where S is a set of super-types and $\ll$ is a transitive relation on $\mathrm{tp}(S)$ satisfying the following conditions:

(C1) if $\langle \pi, \Pi \rangle \in S$ and $\pi' \in \Pi$, then there exists $\langle \pi', \Pi' \rangle \in S$ with $\Pi' \subseteq \Pi$;

(C2) if $\pi \ll \pi'$, $\langle \pi, \Pi \rangle \in S$ and $\langle \pi', \Pi' \rangle \in S$, then $\{\pi'\} \cup \Pi' \subseteq \Pi$.

For a structure $\mathfrak{A}$, the *certificate of* $\mathfrak{A}$, denoted $C(\mathfrak{A})$, is the pair $(S, \ll)$, where $S = \{\mathrm{stp}^{\mathfrak{A}}[a] \mid a \in A\}$ is the set of super-types realized in $\mathfrak{A}$, and $\pi \ll \pi'$ if and only if π and π' are realized in $\mathfrak{A}$ and $\mathfrak{A} \models \forall x(\pi \to \forall y(\pi'(y) \to \mathfrak{t}(x,y)))$. Intuitively, a certificate is a description of a structure listing the realized super-types and containing a partial order which specifies when all elements realizing one 1-type are related by $\mathfrak{t}$ to all elements realizing another 1-type.

▶ **Lemma 2.** *If $\mathfrak{A}$ is any structure interpreting $\Sigma \cup \{\mathfrak{t}\}$, $C(\mathfrak{A})$ is a certificate.*

Proof. Write $C(\mathfrak{A}) = (S, \ll)$. Obviously $\ll$ is transitive. We must check (**C1**) and (**C2**).
(**C1**): Suppose $\langle \pi, \Pi \rangle \in S$ and $\pi' \in \Pi$. Let a be such that $\mathrm{stp}^{\mathfrak{A}}[a] = \langle \pi, \Pi \rangle$. Then there exists a $b \in A$ such that $\mathrm{tp}^{\mathfrak{A}}[b] = \pi'$ and $\mathfrak{A} \models \mathfrak{t}[a,b]$. Let $\mathrm{stp}^{\mathfrak{A}}[b] = \langle \pi', \Pi' \rangle$.
(**C2**): Suppose $\langle \pi, \Pi \rangle \in S$ and $\langle \pi', \Pi' \rangle \in S$ with $\pi \ll \pi'$, and let $a, b \in A$ be such that $\mathrm{stp}^{\mathfrak{A}}[a] = \langle \pi, \Pi \rangle$ and $\mathrm{stp}^{\mathfrak{A}}[b] = \langle \pi', \Pi' \rangle$. Since $\pi \ll \pi'$, by construction of $C(\mathfrak{A})$, we have $\mathfrak{A} \models \forall x(\pi \to \forall y(\pi'(y) \to \mathfrak{t}(x,y)))$, whence it is immediate that $\pi' \in \Pi$ and $\Pi' \subseteq \Pi$. ◀

If $C = (S, \ll)$ is a certificate, and ψ a basic $\mathcal{FL}^2 1\mathrm{T}^u$-formula, we define the relation $C \models \psi$ to hold provided the following six conditions are satisfied. The motivation for this definition is provided by Lemmas 3 and 4.

(i) if ψ is of the form $\forall x(\pi \to \exists y(\mu(y) \wedge \mathfrak{t}(x,y)))$, then, for all Π such that $\langle \pi, \Pi \rangle \in S$, there exists $\pi' \in \Pi$ such that $\models \pi' \to \mu$;
(ii) if ψ is of the form $\forall x(\pi \to \forall y(\pi'(y) \to \mathfrak{t}(x,y)))$ and $\pi, \pi' \in \mathrm{tp}(S)$, then $\pi \ll \pi'$;
(iii) if ψ is of the form $\forall x(\pi \to \exists y(\mu(y) \wedge \neg\mathfrak{t}(x,y)))$, then, for all $\langle \pi, \Pi \rangle \in S$, there exists $\langle \pi', \Pi' \rangle \in S$ such that $\models \pi' \to \mu$ and there exists no $\alpha \in \{\pi\} \cup \Pi$ such that $\alpha \ll \pi'$;
(iv) if ψ is of the form $\forall x(\pi \to \forall y(\pi'(y) \to \neg\mathfrak{t}(x,y)))$, then, for all $\langle \pi, \Pi \rangle \in S$, $\pi' \notin \Pi$;
(v) if ψ is of the form $\exists x.\mu$, then there exists $\langle \pi, \Pi \rangle \in S$ such that $\models \pi \to \mu$;
(vi) if ψ is of the form $\forall x.\mu$, then, for all $\langle \pi, \Pi \rangle \in S$, $\models \pi \to \mu$.

▶ **Lemma 3.** *Let $\mathfrak{A}$ be a structure interpreting $\Sigma \cup \{\mathfrak{t}\}$ and let ψ be a basic $\mathcal{FL}^2 1\mathrm{T}^u$-formula over $\Sigma \cup \{\mathfrak{t}\}$. If $\mathfrak{A} \models \psi$, then $C(\mathfrak{A}) \models \psi$.*

Proof. We write $C(\mathfrak{A}) = (S, \ll)$ and consider the possible forms of ψ in turn.

- $\psi = \forall x(\pi \to \exists y(\mu(y) \wedge \mathfrak{t}(x,y)))$: Suppose $\langle \pi, \Pi \rangle \in S$. Then there exists $a \in A$ with $\mathrm{stp}^{\mathfrak{A}}[a] = \langle \pi, \Pi \rangle$. Since $\mathfrak{A} \models \psi$, choose $b \in A$ such that $\mathfrak{A} \models \mu[b]$ and $\mathfrak{A} \models \mathfrak{t}[a,b]$, and let $\mathrm{tp}^{\mathfrak{A}}[b] = \pi'$. Then $\models \pi' \to \mu$ and $\pi' \in \Pi$, as required.
- $\psi = \forall x(\pi \to \forall y(\pi'(y) \to \mathfrak{t}(x,y)))$: It is immediate by the construction of $C(\mathfrak{A})$ that, if $\pi, \pi' \in \mathrm{tp}(S)$, then $\pi \ll \pi'$;
- $\psi = \forall x(\pi \to \exists y(\mu(y) \wedge \neg\mathfrak{t}(x,y)))$: Suppose $\langle \pi, \Pi \rangle \in S$. Then there exists $a \in A$ with $\mathrm{stp}^{\mathfrak{A}}[a] = \langle \pi, \Pi \rangle$. Since $\mathfrak{A} \models \psi$, choose $b \in A$ such that $\mathfrak{A} \models \mu[b]$ and $\mathfrak{A} \not\models \mathfrak{t}[a,b]$, and let $\mathrm{tp}^{\mathfrak{A}}[b] = \pi'$, so that $\models \pi' \to \mu$. We require only to show that there exists no $\alpha \in \{\pi\} \cup \Pi$ such that $\alpha \ll \pi'$. Suppose, for contradiction, that such an α exists. By (**C1**), $\alpha \in \mathrm{tp}(S)$. If $\alpha = \pi$, then, by the definition of $\ll$, we have $\mathfrak{A} \models \forall x(\pi \to \forall y(\pi'(y) \to \mathfrak{t}(x,y)))$, which contradicts the supposition that $\mathfrak{A} \not\models \mathfrak{t}[a,b]$. If $\alpha \in \Pi$, then, by the definition of Π and $\ll$, we have an element $a' \in A$ such that $\mathrm{tp}^{\mathfrak{A}}[a'] = \alpha$, $\mathfrak{A} \models \mathfrak{t}[a,a']$ and $\mathfrak{A} \models \forall x(\alpha \to \forall y(\pi'(y) \to \mathfrak{t}(x,y)))$, which again contradicts the supposition that $\mathfrak{A} \not\models \mathfrak{t}[a,b]$.
- $\psi = \forall x(\pi \to \forall y(\pi'(y) \to \neg\mathfrak{t}(x,y)))$: Suppose $\langle \pi, \Pi \rangle \in S$ and let $a \in A$ be such that $\mathrm{stp}^{\mathfrak{A}}[a] = \langle \pi, \Pi \rangle$. Since $\mathfrak{A} \models \psi$, we have $\pi' \notin \Pi$.
- $\psi = \exists x.\mu$ or $\psi = \forall x.\mu$. Immediate by construction of S. ◀

▶ **Lemma 4.** *If $C = (S, \ll)$ is a certificate, then there exists a structure $\mathfrak{A}$ over a domain of cardinality $2|S|$ such that, for any basic $\mathcal{FL}^2 1\mathrm{T}^u$-formula ψ over Σ, $C \models \psi$ implies $\mathfrak{A} \models \psi$.*

Proof. Define $A^+ = \{a^+_{\pi,\Pi} \mid \langle \pi, \Pi \rangle \in S\}$ and $A^- = \{a^-_{\pi,\Pi} \mid \langle \pi, \Pi \rangle \in S\}$, where the various $a^+_{\pi,\Pi}$ and $a^-_{\pi,\Pi}$ are some objects (assumed distinct), and set $A = A^+ \cup A^-$. Define the binary relations $T_1 = \{\langle a^\pm_{\pi,\Pi}, a^+_{\pi',\Pi'} \rangle \mid \{\pi'\} \cup \Pi' \subseteq \Pi\}$ and $T_2 = \{\langle a^\pm_{\pi,\Pi}, a^\pm_{\pi',\Pi'} \rangle \mid \pi \ll \pi'\}$, and let T be the transitive closure of $T_1 \cup T_2$. Intuitively, we may think of the elements $a^+_{\pi',\Pi'}$ as witnessing existential formulas of the form $\exists y(\mu(y) \wedge \mathfrak{t}(x,y))$, where $\models \pi' \rightarrow \mu$, and of the elements $a^-_{\pi',\Pi'}$ as witnessing existential formulas of the form $\exists y(\mu(y) \wedge \neg\mathfrak{t}(x,y))$. Now define $\mathfrak{A}$ on the domain A by setting $\text{tp}^{\mathfrak{A}}[a^\pm_{\pi,\Pi}] = \pi$ for all $\langle \pi, \Pi \rangle \in S$, and by setting $\mathfrak{t}^{\mathfrak{A}} = T$.

We observe that if $a = a^\pm_{\pi,\Pi}$ and $b = a^\pm_{\pi',\Pi'}$ are elements of A such that a is related to b by either T_1 or T_2, then $\{\pi'\} \cup \Pi' \subseteq \Pi$. Indeed, for T_1, this is immediate by definition; and for T_2, it follows from property (**C2**) of certificates. It follows by induction that, if a is related to b by T, then $\{\pi'\} \cup \Pi' \subseteq \Pi$. To prove the lemma, we consider the possible forms of ψ in turn.

- $\psi = \forall x(\pi \rightarrow \exists y(\mu(y) \wedge \mathfrak{t}(x,y)))$: Suppose $a = a^\pm_{\pi,\Pi}$. Since $C \models \psi$, there exists $\pi' \in \Pi$ such that $\pi' \rightarrow \mu$. By (**C1**), there exists $\langle \pi', \Pi' \rangle \in S$ such that $\Pi' \subseteq \Pi$. Letting $b = a^+_{\pi',\Pi'}$, we have that a is related to b by T_1. But then $\mathfrak{A} \models \mathfrak{t}[a,b]$ and $\mathfrak{A} \models \mu[b]$ by construction of $\mathfrak{A}$.
- $\psi = \forall x(\pi \rightarrow \forall y(\pi'(y) \rightarrow \mathfrak{t}(x,y)))$: Since $C \models \psi$, we have $\pi \ll \pi'$. Suppose now $a = a^\pm_{\pi,\Pi}$ and $b = a^\pm_{\pi',\Pi'}$. Thus, a is related to b by T_2, and so by construction of $\mathfrak{A}$, $\mathfrak{A} \models \mathfrak{t}[a,b]$.
- $\psi = \forall x(\pi \rightarrow \exists y(\mu(y) \wedge \neg\mathfrak{t}(x,y)))$: Suppose $a = a^\pm_{\pi,\Pi}$. Since $C \models \psi$, there exists $\langle \pi', \Pi' \rangle \in S$ such that $\pi' \rightarrow \mu$, and such that there is no $\alpha \in \{\pi\} \cup \Pi$ with $\alpha \ll \pi'$. Now let $b = a^-_{\pi',\Pi'}$. By construction of $\mathfrak{A}$, $\mathfrak{A} \models \mu[b]$. It suffices to show that $\mathfrak{A} \not\models \mathfrak{t}[a,b]$. For otherwise, by the definition of T, there exists a chain of elements $a = a_1, \ldots, a_m = b$ with each related to the next by either T_1 or T_2 and with a_{m-1} related to a_m by T_2. (Notice that nothing can be related by T_1 to $b = a^-_{\pi',\Pi'}$.) Writing $a_{m-1} = a^\pm_{\alpha,\Pi''} \in S$, we see that $\alpha \ll \pi'$, and, moreover, that a is either identical to a_{m-1}, or related to it by T. As we observed above, if $a^\pm_{\pi,\Pi}$ is related to $a^\pm_{\alpha,\Pi''}$ by T, then $\alpha \in \Pi$. Thus, either way, $\alpha \in \{\pi\} \cup \Pi$. But we are supposing that no such α exists.
- $\psi = \forall x(\pi \rightarrow \forall y(\pi'(y) \rightarrow \neg\mathfrak{t}(x,y)))$: Suppose $a = a^\pm_{\pi,\Pi}$ and $b = a^\pm_{\pi',\Pi'}$ are elements of A. We observed above that, if a is related to b by T, then $\pi' \in \Pi$, contradicting the assumption that $C \models \psi$. Thus, by construction of $\mathfrak{A}$, $\mathfrak{A} \not\models \mathfrak{t}[a,b]$.
- $\psi = \exists x.\mu$ or $\psi = \forall x.\mu$. Immediate by construction of $\mathfrak{A}$. ◀

Since the number of super-types over Σ is bounded by $2^{(2^{|\Sigma|}+|\Sigma|)}$, and a structure $\mathfrak{A}$ can be guessed and verified to be a model of any m-variable first-order formula φ in time $O(|\varphi| \cdot |A|^m)$ [25], Lemmas 1–4 yield:

▶ **Lemma 5.** *If φ is a satisfiable formula of $\mathcal{FL}^2 1\mathrm{T}^u$, then φ has a model of size at most doubly exponential in $\|\varphi\|$. Hence the satisfiability problem for $\mathcal{FL}^2 1\mathrm{T}^u$ is in* 2-NExpTime.

3.2 The logics $\mathcal{FL}^m 1\mathrm{T}$ for $m \geq 2$

Let Σ be a signature of predicates of positive arity, excluding $\mathfrak{t}$. An atomic formula of $\mathcal{FL}^m 1\mathrm{T}$ involving a predicate from $\Sigma \cup \{\mathfrak{t}\}$ will be called a *fluted m-atom over* $\Sigma \cup \{\mathfrak{t}\}$. A *fluted m-literal* is a fluted m-atom or the negation thereof. A *fluted m-clause* is a disjunction of fluted m-literals. We allow the absurd formula $\bot$ (i.e. the empty disjunction) to count as a fluted m-clause. Thus, any literal of a fluted m-clause has arguments $x_h, \ldots, x_m$, in that order, for some h ($1 \leq h \leq m$). When writing fluted m-clauses, we silently remove bracketing, re-order literals and delete duplicated literals as necessary. A *fluted m-type* is a maximal consistent set of fluted m-literals; where convenient, we identify fluted m-types with their conjunctions. If $\mathfrak{A}$ is a structure interpreting $\Sigma \cup \{\mathfrak{t}\}$, any tuple $a_1, \ldots, a_m$ from A satisfies a unique fluted m-type; we denote it $\text{ftp}^{\mathfrak{A}}[a_1, \ldots, a_m]$. Note that a fluted 1-type over

$\Sigma \cup \{\mathfrak{t}\}$ coincides with what we earlier called a 1-type over Σ. Reference to the signature $\Sigma \cup \{\mathfrak{t}\}$ will as usual be suppressed when clear from context. Predicates in Σ will be referred to as *non-distinguished*. Our strategy will be to reduce the (finite) satisfiability problem for $\mathcal{FL}^m1\mathrm{T}$ to that for $\mathcal{FL}^21\mathrm{T}$ (Lemma 11), and thence to that for $\mathcal{FL}^21\mathrm{T}^u$ (Lemma 9), which we have already dealt with (Lemma 5).

A $\mathcal{FL}^m1\mathrm{T}$-formula φ $(m \geq 2)$ is in *clause normal form* if it is of the form

$$\forall x_1 \cdots x_m.\Omega \ \wedge \bigwedge_{i=1}^{s} \forall x_1 \cdots x_{m-1} \, (\alpha_i \rightarrow \exists x_m.\Gamma_i) \wedge \bigwedge_{j=1}^{t} \forall x_1 \cdots x_{m-1}(\beta_j \rightarrow \forall x_m.\Delta_j), \tag{3}$$

where $\Omega, \Gamma_1, \ldots, \Gamma_s, \Delta_1, \ldots, \Delta_t$ are sets of fluted m-clauses, and $\alpha_1, \ldots, \alpha_s$, $\beta_1, \ldots, \beta_t$ fluted $(m-1)$-atoms. We refer to $\forall x_1 \cdots x_m.\Omega$ as the *static conjunct* of φ, to conjuncts of the form $\forall x_1 \cdots x_{m-1} \, (\alpha_i \rightarrow \exists x_m \Gamma_i)$ as the *existential conjuncts* of φ, and to conjuncts of the form $\forall x_1 \cdots x_{m-1}(\beta_j \rightarrow \forall x_m.\Delta_j)$ as the *universal conjuncts* of φ.

Using the same techniques as for Lemma 1, we can transform any $\mathcal{FL}^m1\mathrm{T}$-formula into clause normal form.

▶ **Lemma 6.** *Let φ be an $\mathcal{FL}^m1\mathrm{T}$-formula, $m \geq 2$. There exists an $\mathcal{FL}^m1\mathrm{T}$-formula ψ in clause normal form such that: (i) $\models \psi \rightarrow \varphi$; and (ii) if φ has a model then so has ψ; (iii) $\|\psi\|$ is bounded by a polynomial function of $\|\varphi\|$.*

For fragments of first-order logic not involving equality, we are free to duplicate any element a in a structure $\mathfrak{A}$. More formally, we have the following lemma, which will be used as a step in the ensuing argument.

▶ **Lemma 7.** *Let $\mathfrak{A}$ be any structure, and let $z > 0$. There exists a structure $\mathfrak{B}$ such that (i) if φ is any first-order formula without equality, then $\mathfrak{A} \models \varphi$ if and only if $\mathfrak{B} \models \varphi$; (ii) $|B| = z \cdot |A|$; and (iii) if $\psi(x_1, \ldots, x_{m-1}) = \exists x_m.\chi(x_1, \ldots, x_m)$ is a first-order formula without equality, and $\mathfrak{B} \models \psi[b_1, \ldots, b_{m-1}]$, then there exist at least z distinct elements b of B such that $\mathfrak{B} \models \chi[b_1, \ldots, b_{m-1}, b]$.*

Keeping the signature Σ fixed, we employ the standard apparatus of resolution theorem-proving to eliminate non-distinguished predicates of arity 2 or more. Suppose $p \in \Sigma$ is a predicate of arity m, and let γ' and δ' be fluted m-clauses over Σ. Then, $\gamma = p(x_1, \ldots, x_m) \vee \gamma'$ and $\delta = \neg p(x_1, \ldots, x_m) \vee \delta'$ are also fluted m-clauses, as indeed is $\gamma' \vee \delta'$. In that case, we call $\gamma' \vee \delta'$ a *fluted resolvent* of γ and δ, and we say that $\gamma' \vee \delta'$ is *obtained by fluted resolution* from γ and δ *on* $p(x_1, \ldots, x_m)$. Thus, fluted resolution is simply a restriction of the familiar resolution rule from first-order logic to the case where the resolved-on literals have maximal arity, m, and (in the case $m = 2$) do not feature the distinguished predicate $\mathfrak{t}$. It may be helpful to note the following at this point: (i) if γ and δ resolve to form ϵ, then $\models \forall x_1 \cdots \forall x_m(\gamma \wedge \delta \rightarrow \epsilon)$; (ii) the fluted resolvent of two fluted m-clauses may or may not involve predicates of arity m; (iii) in fluted resolution, the arguments of the literals in the fluted m-clauses undergo no change when forming the resolvent; (iv) if the fluted m-clause γ involves no predicates of arity m, then it cannot undergo fluted resolution at all.

If Γ is a set of fluted m-clauses, denote by Γ^* the smallest set of fluted m-clauses including Γ and closed under fluted resolution. If $\Gamma = \Gamma^*$, we say that it is *closed under fluted resolution*. We further denote by Γ° the result of deleting from Γ^* any clause involving a *non-distinguished* predicate of arity m. Observe that, since all fluted m-atoms involving predicates of *non*-maximal arity are of the form $p(x_h, \ldots, x_m)$ for some $h \geq 2$, it follows that Γ° features the variable x_1 *only* in the case $m = 2$, and even then only in literals of the form $\pm\mathfrak{t}(x_1, x_2)$.

The following lemma is, in effect, nothing more than the familiar completeness theorem for (ordered) propositional resolution. The proof is omitted due to space limits.

▶ **Lemma 8.** *Let Γ be a set of fluted m-clauses over a signature $\Sigma \cup \{\mathfrak{t}\}$, let Σ' be the result of removing all predicates of maximal arity m from Σ, and let τ^- be a fluted m-type over $\Sigma' \cup \{\mathfrak{t}\}$. If τ^- is consistent with Γ°, then there exists a fluted m-type τ over the signature $\Sigma \cup \{\mathfrak{t}\}$ such that $\tau \supseteq \tau^-$ and τ is consistent with Γ.*

The following lemma employs a technique from [13] to eliminate binary predicates.

▶ **Lemma 9.** *Let φ be an $\mathcal{FL}^2 1\mathrm{T}$-formula in clause normal form over a signature $\Sigma \cup \{\mathfrak{t}\}$, and suppose that φ has s existential and t universal conjuncts. Then there exists a clause normal form $\mathcal{FL}^2 1\mathrm{T}^u$-formula φ' over a signature $\Sigma' \cup \{\mathfrak{t}\}$ such that: (i) φ' has at most $2^t s$ existential and 2^t universal conjuncts; (ii) $|\Sigma'| \leq |\Sigma| + 2^t(s+1)$; (iii) if φ has a model, so does φ'; and (iv) if φ' has a model of size M, then φ has a model of size at most sM.*

Proof. Let $\varphi = \forall x_1 x_2.\Omega \ \wedge \bigwedge_{i=1}^s \forall x_1 \left(p_i(x_1) \to \exists x_2.\Gamma_i\right) \wedge \bigwedge_{j=1}^t \forall x_1 (q_j(x_1) \to \forall x_2.\Delta_j)$, where $\Omega, \Gamma_1, \ldots, \Gamma_s, \Delta_1, \ldots, \Delta_t$ are sets of fluted 2-clauses, and $p_1, \ldots, p_s$, $q_1, \ldots, q_t$ unary predicates. Write $T = \{1, \ldots, t\}$. For all i $(1 \leq i \leq s)$ and all $J \subseteq T$, let $p_{i,J}$ and q_J be new unary predicates. The intended interpretation of $p_{i,J}(x_1)$ is "x_1 satisfies p_i, and also satisfies q_j for every $j \in J$;" and the intended interpretation of $q_J(x_1)$ is "x_1 satisfies q_j for every $j \in J$." Let φ' be the conjunction of the sentences:
(a) $\bigwedge_{i=1}^s \bigwedge_{J \subseteq T} \forall x_2((p_i(x_2) \wedge \bigwedge_{j \in J} q_j(x_2)) \to p_{i,J}(x_2))$,
(b) $\bigwedge_{J \subseteq T} \forall x_2((\bigwedge_{j \in J} q_j(x_2)) \to q_J(x_2))$,
(c) $\bigwedge_{i=1}^s \bigwedge_{J \subseteq T} \forall x_1 \left(p_{i,J}(x_1) \to \exists x_2 \left(\Gamma_i \cup \Omega \cup \bigcup\{\Delta_j \mid j \in J\}\right)^\circ\right)$, and
(d) $\bigwedge_{J \subseteq T} \forall x_1 \left(q_J(x_1) \to \forall x_2 \left(\Omega \cup \bigcup\{\Delta_j \mid j \in J\}\right)^\circ\right)$.
Observe that φ' contains no non-distinguished binary predicates, and hence is in $\mathcal{FL}^2 1\mathrm{T}^u$. Clearly, φ' satisfies properties (i) and (ii). To show (iii), suppose $\mathfrak{A} \models \varphi$, and let $\mathfrak{A}'$ be the structure obtained by interpreting the predicates $p_{i,J}$ and q_J as suggested above. To show (iv), suppose φ' has a model of size M. By Lemma 7, φ' has a model $\mathfrak{B}$ of size sM in which witnesses for all the conjuncts in (c) are duplicated s times. We need to show that $\mathfrak{B}$ can be expanded to a model of φ. Fix $a \in B$ and suppose a satisfies p_1. Let J be the set of indices j such that a satisfies q_j. By (a), putting $i = 1$, a satisfies $p_{1,J}$, whence, by (c), there exists b such that the pair $\langle a, b\rangle$ satisfies $(\Gamma_1 \cup \Omega \cup \bigcup\{\Delta_j \mid j \in J\})^\circ$. But Lemma 8 guarantees that we can expand $\mathfrak{B}$ by interpreting the non-distinguished binary predicates so that $\langle a, b\rangle$ satisfies $\Gamma_1 \cup \Omega \cup \bigcup\{\Delta_j \mid j \in J\}$. Because of the duplication of witnesses, we can repeat with $p_2, \ldots, p_s$, choosing a fresh witness each time, so as to avoid clashes. Do this for all elements a. At the end of the process, the partially defined expansion of $\mathfrak{B}$ satisfies all the existential conjuncts of φ, and violates none of the universal or static conjuncts. A precisely similar argument shows that we may complete the expansion so that no universal or static conjuncts of φ are violated. ◀

Thus, at the expense of an exponentially larger signature, we have reduced the (finite) satisfiability problem for $\mathcal{FL}^2 1\mathrm{T}$ to that for $\mathcal{FL}^2 1\mathrm{T}^u$. By Lemmas 5 and 9, we obtain

▶ **Lemma 10.** *Let φ be a $\mathcal{FL}^2 1\mathrm{T}$-formula. If φ is satisfiable, then φ has a model of size at most triply exponential in $\|\varphi\|$. Hence the satisfiability problem for $\mathcal{FL}^2 1\mathrm{T}$ is in 3-NExpTime.*

We now establish the finite model property for the whole of $\mathcal{FL}1\mathrm{T}$ by eliminating variables from $\mathcal{FL}^{m+1}1\mathrm{T}$, where $m \geq 2$, one at a time. The proof of the following Lemma is similar to the proof of Lemma 9 and is omitted due to space limits.

▶ **Lemma 11.** *Let φ be a clause normal form $\mathcal{FL}^{m+1}1\mathrm{T}$-formula $(m \geq 2)$ over a signature $\Sigma \cup \{\mathrm{t}\}$, and suppose that φ has s existential conjuncts and t universal conjuncts. Then there exists a clause normal form $\mathcal{FL}^{m}1\mathrm{T}$-formula φ' over a signature $\Sigma' \cup \{\mathrm{t}\}$ such that the following hold: (i) φ' has at most $2^t s$ existential and 2^t universal conjuncts; (ii) $|\Sigma'| \leq |\Sigma| + 2^t(s+1)$; (iii) if φ has a model, so does φ'; and (iv) if φ' has a model of size M, then φ has a model of size at most sM.*

▶ **Theorem 12.** *Let φ be a $\mathcal{FL}^{m}1\mathrm{T}$-formula for $m \geq 2$. If φ is satisfiable, then φ has a model of size at most $(m+1)$-tuply exponential in $\|\varphi\|$. Hence the satisfiability problem for $\mathcal{FL}^{m}1\mathrm{T}$ is in non-deterministic $(m+1)$-tuply exponential time.*

Proof. Induction on m. The case $m = 2$ is Lemma 10. The inductive step is Lemma 11. ◀

We mentioned in Sec. 1 that [14] establishes a lower bound of $\lfloor m/2 \rfloor$-NExpTime-hard for the satisfiability problem for $\mathcal{FL}^{m}$. For $m \geq 3$, this appears to be the best available lower bound on the corresponding problem for $\mathcal{FL}^{m}1\mathrm{T}$. Thus, a gap remains between the best available upper and lower complexity bounds. Certainly, it follows that the satisfiability problem for $\mathcal{FL}1\mathrm{T}$ is Tower-complete, as for $\mathcal{FL}$.

4 Fluted Logic with more Transitive Relations

In this section we show two undecidability results for the fluted fragment with two variables, $\mathcal{FL}^2$, extended with more transitive relations, that have been informally announced in [24]. We employ the apparatus of tiling systems.

A *tiling system* is a tuple $\boldsymbol{\mathcal{C}} = (\mathcal{C}, \mathcal{C}_H, \mathcal{C}_V)$, where $\mathcal{C}$ is a finite set of *tiles*, and $\mathcal{C}_H$, $\mathcal{C}_V \subseteq \mathcal{C} \times \mathcal{C}$ are the *horizontal* and *vertical* constraints.

Let S be any of the spaces $\mathbb{N} \times \mathbb{N}$, $\mathbb{Z} \times \mathbb{Z}$ or $\mathbb{Z}_t \times \mathbb{Z}_t$. A tiling system $\boldsymbol{\mathcal{C}}$ *tiles* S, if there exists a function $\rho : S \to \mathcal{C}$ such that for all $(p,q) \in S$: $(\rho(p,q), \rho(p+1,q)) \in \mathcal{C}_H$ and $(\rho(p,q), \rho(p,q+1)) \in \mathcal{C}_V$. The following problems are known to be undecidable (cf. e.g. [3]):

- Given a tiling system $\boldsymbol{\mathcal{C}}$ determine if $\boldsymbol{\mathcal{C}}$ tiles $\mathbb{Z} \times \mathbb{Z}$, or $\mathbb{N} \times \mathbb{N}$.
- Given a tiling system $\boldsymbol{\mathcal{C}}$ determine if $\boldsymbol{\mathcal{C}}$ tiles $\mathbb{Z}_t \times \mathbb{Z}_t$, for some $t \geq 1$.

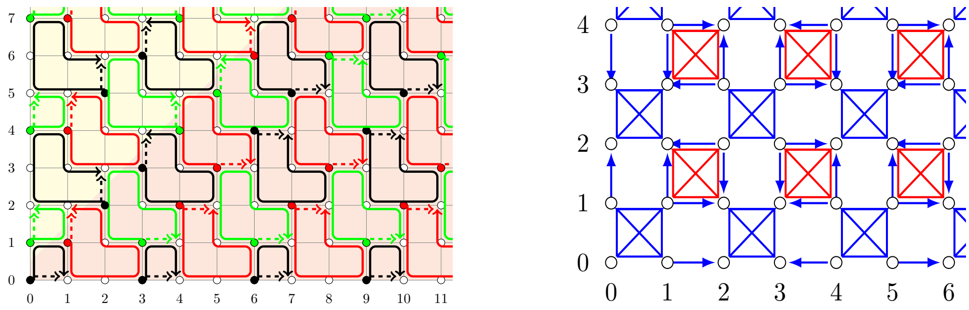
In this section we first prove the following theorem.

▶ **Theorem 13.** *The satisfiability problem for $\mathcal{FL}^2 3\mathrm{T}$, the two-variable fluted fragment with three transitive relations, is undecidable.*

Proof. Suppose the signature contains transitive relations b (black), g (green) and r (red), and additional unary predicates e, e', f, l, $c_{i,j}$ $(0 \leq i \leq 5,\ 0 \leq j \leq 2)$ and $d_{i,j}$ $(0 \leq i \leq 2,\ 0 \leq j \leq 5)$; we refer to the $c_{i,j}$'s and to the $d_{i,j}$'s as *colours.*

We reduce from the $\mathbb{N} \times \mathbb{N}$ tiling problem. We first write a formula φ_{grid} that captures several properties of the intended expansion of the $\mathbb{N} \times \mathbb{N}$ grid as shown in Fig. 2a. There the predicates $c_{i,j}$ and $d_{i,j}$ together define a partition of the universe as follows: an element (k,k') with $k' > k$ (i.e. in the yellow region, above the diagonal) satisfies $c_{i,j}$ with $i = k \mod 6$, $j = k' \mod 3$, and an element (k,k') with $k \geq k'$ (i.e. in the pink region, on or below the diagonal) satisfies $d_{i,j}$ with $i = k \mod 3, j = k' \mod 6$. Paths of the same transitive relation have length at most 7 and follow one of four designated patterns. Remaining unary predicates mark the following elements: l – left column, f – bottom row, e – main diagonal, and e' – elements with coordinates $(k, k+1)$.

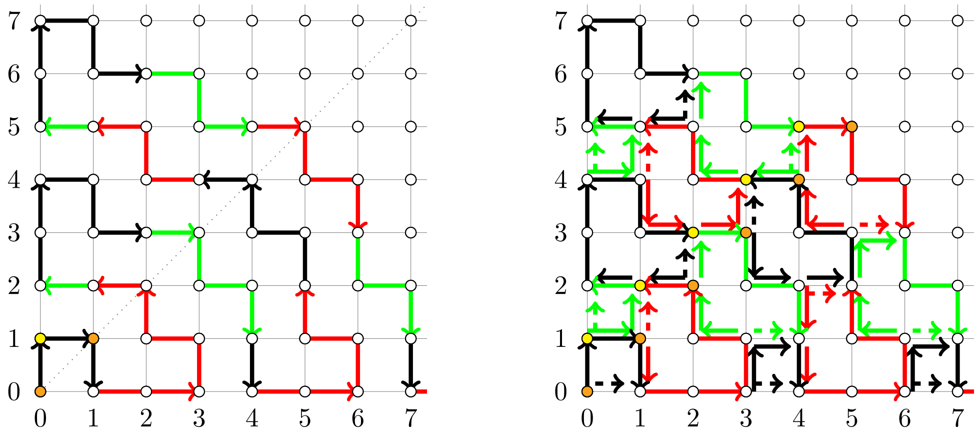
The formula φ_{grid} comprises a large number of conjuncts. We have organized these conjuncts into groups, each of which secures a particular property (or collection of properties) exhibited by its models. The first two properties are very simple:



(a) Three transitive relations: b, g and r. Filled nodes depict the beginning of a transitive path of the same colour; dotted lines connect the first element with the last element on such path.

(b) Two transitive relations: b and r. Edges without arrows depict connections in both direction.

Figure 2 Expansions of the $\mathbb{N} \times \mathbb{N}$ grid in the proofs of Theorem 13 (a) and Theorem 16 (b).



(a) Path starting with an initial element and generated by witnesses for conjuncts of group (3).

(b) Additional edges arising from conjuncts of group (4) (solid lines drawn inside grid cells) and transitivity (dashed lines). Nodes on the diagonals are marked orange (e) and yellow (e').

Figure 3 Construction of the intended model of φ_{grid} in the proof of Theorem 13.

(1) There is an "initial" element satisfying $d_{00}(x) \wedge e(x) \wedge l(x) \wedge f(x)$.

(2) The predicates $c_{i,j}$ and $d_{i,j}$ together partition the universe.

The third property generates the path shown in Fig. 3a:

(3) Each element has a b- r- or g- successor as shown in the path shown in Fig. 3a, and satisfying the appropriate predicates $c_{i,j}$ or $d_{i,j}$. Specifically, if a node in this path has coordinates (x, y) with $y > x$, then it satisfies $c_{i,j}$ where $i = x \mod 3$ and $j = y \mod 6$; and when $y \leq x$, then the node satisfies $d_{i,j}$ where $i = x \mod 3$ and $j = y \mod 6$.

The conjuncts enforcing this property have the form

$$\forall x\big(colour(x) \wedge diag(x) \wedge border(x) \rightarrow \exists y(t(x,y) \wedge colour'(y))\big), \tag{3a}$$

where $colour$ and $colour'$ stand for one of the predicate letters $c_{i,j}$ or $d_{i,j}$, $diag(x)$ stands for one of the literals $e'(x)$, $\neg e'(x)$, $e(x)$, $\neg e(x)$ or $\top$ (i.e. the logical constant true), $border(x)$ stands for one of the literals $l(x)$, $\neg l(x)$, $f(x)$, $\neg f(x)$ or $\top$, and t stands for one of the transitive predicate letters b, r or g. The precise combinations of the literals and predicate letters in these conjuncts can be read from Fig. 3a (cf. [16] for details).

To connect all pairs of elements that are neighbours in the standard grid we require a fourth property, which we give in schematic form as follows:

(4) *Certain* pairs of elements connected by *one* transitive relation are also connected by *another*, as indicated in Fig. 3b.

Here are some examples of the conjuncts enforcing this property:

$$\forall x(c_{01}(x) \rightarrow \forall y(b(x,y) \wedge (c_{11}(y) \vee d_{11}(y)) \rightarrow g(x,y))), \tag{4a}$$

$$\forall x(d_{11}(x) \rightarrow \forall y(b(x,y) \wedge d_{10}(y) \rightarrow r(x,y))), \tag{4b}$$

$$\forall x(d_{11}(x) \rightarrow \forall y(r(x,y) \wedge (c_{12}(y) \vee d_{12}(y)) \rightarrow g(x,y))), \tag{4c}$$

The role of these conjuncts can be explained referring to Fig. 3b. For example, employing (4b) for the element $(1,1)$ in the intended model $\mathfrak{G}$, we get $\mathfrak{G} \models r((1,1),(1,0))$; hence by transitivity of r, also $\mathfrak{G} \models r((1,1),(1,2))$. This, applying (4c), implies $\mathfrak{G} \models g((1,1),(1,2))$. By (4a), we get $\mathfrak{G} \models g((0,1),(1,1))$ and, by transitivity of g, $\mathfrak{G} \models g((0,1),(0,2))$. The process is illustrated in Fig. 3b; when carried on along the zig-zag path, it constructs a grid-like structure.

These conjuncts depend on having available the predicates marking the borders and the diagonals. Specifically, we require the following property:

(5) the predicates l, f, e and e' are distributed to mark the left-most column, the first row, the diagonal and the "super-diagonal" of the grid, as indicated above. To secure this property, we add to φ_{grid} several conjuncts, for instance:

$$\bigwedge_{0 \leq i \leq 2,\ 0 \leq j \leq 5} \forall x\big(d_{i,j}(x) \wedge \pm e(x) \rightarrow \forall y((b(x,y) \vee g(x,y) \vee r(x,y)) \wedge d_{i+1,j+1}(y) \rightarrow \pm e(y))\big), \tag{5a}$$

where $\pm e(x)$ denotes uniformly $e(x)$ or $\neg e(x)$. Similar conjuncts are added for the super-diagonal, left column and bottom row; and also for the connection with and between e and e'. The conjuncts ensuring properties (4) and (5) work in tandem. For instance, applying (5a) to (1,1) we get e is true at $(2,2)$; then, following the zig-zag path and applying more conjuncts from the group (4), we get that $g((2,2),(3,3))$ holds, so the node $(3,3)$ will be marked by e; this will propagate along the main diagonal.

The structure $\mathfrak{G}$ depicted in Fig. 2a is a model of φ_{grid}. In fact, φ_{grid} is an infinity axiom. To see this, let $\mathfrak{A} \models \varphi_{grid}$ and define an injective embedding ρ of the standard grid on $\mathbb{N} \times \mathbb{N}$ into $\mathfrak{A}$ as follows. Let $next : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{N} \times \mathbb{N}$ be the successor function defined on $\mathbb{N} \times \mathbb{N}$ as depicted by the zig-zag path in the left-hand picture of Fig. 3 starting at $(0,0)$ (ignoring any colours). Denote $s_0 = (0,0)$, $s_n = next(s_{n-1})$ and $S_n = \{s_0, \ldots, s_n\}$. Let $a_0 \in A$ be an element such that $\mathfrak{A} \models d_{00}(a) \wedge e(a) \wedge l(a) \wedge f(a)$ that exists by condition (1). Define $\rho(s_0) = a_0$. Now, we proceed inductively: suppose $\rho(s_{n-1})$ has already been defined in step $n-1$ of the induction and $\rho(s_{n-1}) = a_{n-1}$. Let a_n be the witness of a_{n-1} for the appropriate conjunct from the group (3), i.e. where the unary literals for x agree with the unary literals satisfied by a_{n-1} in $\mathfrak{A}$. Define $\rho(s_n) = a_n$. Using induction one can prove that ρ is indeed injective: in the inductive step we assume that $\mathfrak{A} \restriction \{a_0, \ldots, a_{n-1}\}$ is isomorphic to $\mathfrak{G} \restriction S_{n-1}$, and we show that $a_n \notin \{a_0, \ldots, a_{n-1}\}$ and $\mathfrak{A} \restriction \{a_0, \ldots, a_n\}$ and $\mathfrak{G} \restriction S_n$ are again isomorphic. In the proof one considers several cases depending on the 1-type realized by a_n. The formula φ_{grid} ensures that $a_0, \ldots, a_{18}$ are all distinct, and any eight consecutive elements of the sequence $a_0, \ldots, a_n$ are always distinct. Consider $a_{18} = \rho(4,2)$ that requires a witness $b \in A$ for a conjunct from the group (3) such that $\mathfrak{A} \models r(a_{18}, b) \wedge d_{11}(b)$. Suppose, $b = a_2 = \rho(1,1)$, since $\mathfrak{A} \models d_{11}(a_2)$. Then, by transitivity of g, $\mathfrak{A} \models g(a_{18}, a_{10})$, which is a contradiction with $\mathfrak{G} \models \neg g((4,2),(1,1))$. Other cases are similar and due to page limits have been omitted.

We are now ready to define the horizontal and vertical successors in models of of φ_{grid}. In fact, instead of defining the horizontal grid successor h as one binary relation, we define two disjoint binary relations $\mathsf{rt}(x, y)$ and $\mathsf{lt}(x, y)$ such that rt and the inverse of lt together give the expected horizontal grid successor; they are defined respecting the "direction" of the transitive edges in the models. In the intended model $\mathsf{rt}((x_1, y_1), (x_2, y_2))$ holds if $x_2 = x_1 + 1$, $y_2 = y_1$ and (x_1, y_1) and (x_2, y_2) are connected by b, g or r; and for $\mathsf{lt}((x_1, y_1), (x_2, y_2))$ to hold we require $x_2 = x_1 - 1$ instead of $x_2 = x_1 + 1$. We present the definition of $\mathsf{rt}(x, y)$ in detail below[1]

$$\begin{aligned}\mathsf{rt}(x,y) :=& (b(x,y) \vee g(x,y) \vee r(x,y)) \quad \wedge \\ & (c_{01}(x) \wedge d_{11}(y)) \vee (c_{20}(x) \wedge d_{03}(y)) \vee (c_{42}(x) \wedge d_{25}(y)) \quad \vee \\ & \bigvee_{(i,j)\notin\{(0,2),(1,2),(2,1),(3,1),(4,0),(5,0)\}} (c_{ij}(x) \wedge c_{i+1,j}(y)) \quad \vee \bigvee_{(i,j)\notin\{(2,1),(1,3),(0,5)\}} (d_{ij}(x) \wedge d_{i+1,j}(y))\end{aligned}$$

The relation rt connects elements that are connected by b, g or r and satisfy one of the possible combinations of colours: in the second line the combinations for crossing the diagonal are listed, in the third line the left disjunction describes combinations when both elements are located above the diagonal, and in the right disjunction – when both elements are located on and below the diagonal. The definition of $\mathsf{lt}(x, y)$ complements that of rt. Analogously, we define relations up and dn that together define the vertical grid successor.

Now we are ready to write formulas that properly assign tiles to elements of the model. We do this with a formula φ_{tile}, which again features several conjuncts enforcing various properties of its models. Fortunately, the properties in question are much simpler this time:

(6) Each node encodes precisely one tile: $\forall x\big(\bigvee_{C\in\mathcal{C}} C(x) \wedge \bigwedge_{C\neq D}(\neg C(x) \vee \neg D(x))\big)$.

(7) Adjacent tiles respect $\mathcal{C}_H$. This is secured by the conjuct

$$\bigwedge_{C\in\mathcal{C}} \forall x\big(C(x) \to \forall y\big((\mathsf{rt}(x,y) \to \bigvee_{C':(C,C')\in\mathcal{C}_H} C'(y)) \quad \wedge \quad (\mathsf{lt}(x,y) \to \bigvee_{C':(C',C)\in\mathcal{C}_H} C'(y))\big)\big).$$

(8) Adjacent tiles respect $\mathcal{C}_V$ (written as above using up and dn).

We remark that these latter formulas are not strictly fluted but can be rewritten as fluted using classical tautologies.

Finally, let $\eta_{\boldsymbol{\mathcal{C}}}$ be the conjunction of φ_{grid} and φ_{tile}. We complete the proof showing

▷ Claim 14. $\eta_{\boldsymbol{\mathcal{C}}}$ is satisfiable iff $\boldsymbol{\mathcal{C}}$ tiles $\mathbb{N} \times \mathbb{N}$.

Proof. ($\Leftarrow$) If $\boldsymbol{\mathcal{C}}$ tiles $\mathbb{N} \times \mathbb{N}$ then to show that $\eta_{\boldsymbol{\mathcal{C}}}$ is satisfiable we can expand our intended model $\mathfrak{G}$ for φ_{grid} assigning to every element of the grid a unique $C \in \mathcal{C}$ given by the tiling.

($\Rightarrow$) Let $\mathfrak{A} \models \eta_{\boldsymbol{\mathcal{C}}}$. Let ρ be the embedding of the standard $\mathbb{N} \times \mathbb{N}$ grid into $\mathfrak{A}$ defined above. One can inductively show that ρ maps neighbours in the grid to elements connected by one of the relations $\mathsf{lt}, \mathsf{rt}, \mathsf{up}, \mathsf{dn}$ as follows ($i, j \geq 0$):

$$\mathfrak{A} \models \mathsf{rt}(\rho(i,j), \rho(i+1,j)) \dot\vee \mathsf{lt}(\rho(i+1,j), \rho(i,j)) \wedge \mathsf{up}(\rho(i,j), \rho(i,j+1)) \dot\vee \mathsf{dn}(\rho(i,j+1), \rho(i,j)).$$

(Here, $\dot\vee$ is exclusive disjunction.) So, we can define a tiling of the standard grid assigning to every node (i, j) the unique tile C such that $\mathfrak{A} \models C(\rho(i, j))$. Conditions (7) and (8) together with the above observation ensure that this assignment satisfies the tiling conditions. ◁

◀

[1] Addition in subscripts of the $c_{i,j}$'s is always understood modulo 6 in the first position, and modulo 3 in the second position, i.e. $c_{i+a,j+b}$ denotes $c_{(i+a) \bmod 6,(j+b) \bmod 3}$. Similarly, addition in subscripts of the $d_{i,j}$'s is understood modulo 3 in the first position, and modulo 6 in the second position.

We remark that the formula φ_{grid} in the proof of Theorem 13 is an axiom of infinity, hence the satisfiability and the finite satisfiability problems do not coincide. Moreover, all formulas used in the proof are either guarded or can easily be rewritten as guarded. Furthermore, in the proof it would suffice to assume that b, g and r are interpreted as equivalence relations. Hence, we can strengthen the above theorem as follows.

▶ **Corollary 15.** *The satisfiability problem for the intersection of the fluted fragment with the two-variable guarded fragment is undecidable in the presence of three transitive relations (or three equivalence relations).*

Now we improve the undecidability result to the case of $\mathcal{FL}^2$2T with equality.

▶ **Theorem 16.** *The (finite) satisfiability problem for the two-variable fluted fragment with equality is undecidable in the presence of two transitive relations.*

Proof. We write a formula φ_{grid} over a signature consisting of transitive relations b and r, and unary predicates $c_{i,j}$ $(0 \leq i, j \leq 3)$. The formula φ_{grid} captures several properties of the intended expansion of the $\mathbb{Z} \times \mathbb{Z}$ grid as shown Fig. 2b:

(1) there is an initial element: $\exists x.c_{00}(x)$.
(2) the predicates $c_{i,j}$ partition the universe.
(3) transitive paths do not connect distinct elements of the same colour: $\bigwedge_{0 \leq i,j \leq 3} \forall x(c_{ij}(x) \rightarrow \forall y((b(x, y) \vee r(x, y)) \wedge c_{ij}(y) \rightarrow x = y))$
(4) each element belongs to a 4-element blue clique and to a 4-element red clique.
(5) *certain* pairs of elements connected by r are also connected by b, and *certain* pairs of elements connected by b are also connected by r.

We have given property (5) only schematically, of course; its role is analogous to that of property (4) in the proof of Theorem 13. The remainder of the proof is similar to the one presented for Theorem 13 and is omitted due to space limits. We note that φ_{grid} has also finite models expanding a toroidal grid structure $\mathbb{Z}_{4m} \times \mathbb{Z}_{4m}$ $(m > 0)$ obtained by identifying elements from columns 0 and $4m$ and from rows 0 and $4m$. Hence, the proof gives undecidability for both the satisfiability and the finite satisfiability problems. ◀

Again, the formulas used in the above proof are guarded or can be rewritten as guarded. Also it suffices to assume that r is an equivalence relation. Hence we get the following

▶ **Corollary 17.** *The (finite) satisfiability problem for the intersection of the fluted fragment with equality with the two-variable guarded fragment is undecidable in the presence of two transitive relations (or one transitive and one equivalence relation).*

5 Conclusions

In this paper, we considered the (m-variable) fluted fragment in the presence of different numbers of transitive relations. We showed that $\mathcal{FL}$1T has the finite model property, but $\mathcal{FL}^2$3T admits axioms of infinity and the satisfiability problem for $\mathcal{FL}^2$3T is undecidable. This contrasts with known results for other decidable fragments, in particular, FO^2, where the satisfiability and finite satisfiability problems are undecidable in the presence of two transitive relations, and where the *finite* satisfiability problem is decidable in the presence of one transitive relation. It is open whether the (finite) satisfiability problem for $\mathcal{FL}$ in the presence of *two* transitive relations, $\mathfrak{t}_1$ and $\mathfrak{t}_2$, is decidable. We point out that Lemma 11 in Section 3 could be generalized to normal form formulas from $\mathcal{FL}^{m+1}$2T. Hence, the (finite) satisfiability problem for $\mathcal{FL}$ in the presence of two transitive relations is decidable if and only if the corresponding problem for $\mathcal{FL}^2$ with two transitive relations is decidable. Unfortunately neither the method of Sec. 3 (to show decidability) nor that of Sec. 4 (to show undecidability) appears to apply here. The barrier in the former case is that pairs of elements can be related by both $\mathfrak{t}_1$ and $\mathfrak{t}_2$ via divergent $\mathfrak{t}_1$- and $\mathfrak{t}_2$-chains, so that simple certificates of the kind employed for $\mathcal{FL}^2 1\mathrm{T}^u$ do not guarantee the existence of models. The barrier in the latter case is that the grid construction has to build models featuring transitive paths of *bounded* length, and this seems not to be achievable with just two transitive relations. Finally, we expect that the undecidability result for $\mathcal{FL}^2 3\mathrm{T}$ can be extended to get undecidability of the corresponding finite satisfiability problem.

References

1 H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

2 F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

3 E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* Springer, 1997.

4 D. Danielski and E. Kieroński. Finite Satisfiability of Unary Negation Fragment with Transitivity. In *Proceedings of MFCS 2019*, volume 138, pages 17:1–17:15, 2019.

5 E. Grädel, P. Kolaitis, and M. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

6 A. Herzig. A new decidable fragment of first order logic. In *Abstracts of the 3rd Logical Biennial Summer School and Conference in Honour of S. C.Kleene*, June 1990.

7 Y. Kazakov and I. Pratt-Hartmann. A note on the complexity of the satisfiability problem for graded modal logic. In *Logic in Computer Science*, pages 407–416. IEEE, 2009.

8 E. Kieroński, J. Michaliszyn, I. Pratt-Hartmann, and L. Tendera. Two-variable first-order logic with equivalence closure. *SIAM Journal on Computing*, 43(3):1012–1063, 2014.

9 E. Kieroński and M. Otto. Small Substructures and Decidability Issues for First-Order Logic with Two Variables. *Journal of Symbolic Logic*, 77:729–765, 2012.

10 E. Kieroński and L. Tendera. On finite satisfiability of two-variable first-order logic with equivalence relations. In *Logic in Computer Science*, pages 123–132. IEEE, 2009.

11 R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6:467–480, 1980.

12 A. Noah. Predicate-functors and the limits of decidability in logic. *Notre Dame Journal of Formal Logic*, 21(4):701–707, 1980.

13 I. Pratt-Hartmann. Finite satisfiability for two-variable, first-order logic with one transitive relation is decidable. *Mathematical Logic Quarterly*, 64(3):218–248, 2018.

14 I. Pratt-Hartmann, W. Szwast, and L. Tendera. Quine's fluted fragment is non-elementary. In *25th EACSL Annual Conference on Computer Science Logic, CSL*, volume 62 of *LIPIcs*, pages 39:1–39:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

15 I. Pratt-Hartmann, W. Szwast, and L. Tendera. The fluted fragment revisited. *Journal of Symbolic Logic*, 2019. (Forthcoming).

16 I. Pratt-Hartmann and L. Tendera. The fluted fragment with transitivity. *ArXiv*, 2019. extended version of MFCS'19 paper. URL: https://arxiv.org/abs/1906.09131.

17 W. C. Purdy. Fluted formulas and the limits of decidability. *Journal of Symbolic Logic*, 61(2):608–620, 1996.

18 W. C. Purdy. Complexity and nicety of fluted logic. *Studia Logica*, 71:177–198, 2002.

19 W. V. Quine. On the limits of decision. In *Proceedings of the 14th International Congress of Philosophy*, volume III, pages 57–62. University of Vienna, 1969.

20 W. V. Quine. The variable. In *The Ways of Paradox*, pages 272–282. Harvard University Press, revised and enlarged edition, 1976.

21 S. Schmitz. Complexity hierarchies beyond Elementary. *ACM Transactions on Computation Theory*, 8(1:3):1–36, 2016.

22 D. Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27:477, 1962.

23 W. Szwast and L. Tendera. On the satisfiability problem for fragments of the two-variable logic with one transitive relation. *Journal of Logic and Computation*, 2019. Forthcoming.

24 L. Tendera. Decidability frontier for fragments of first-order logic with transitivity. In *Proceedings of the 31st International Workshop on Description Logics co-located with 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018)*, 2018. URL: `http://ceur-ws.org/Vol-2211/paper-02.pdf`.

25 M. Vardi. On the complexity of bounded-variable queries. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 266–276, 1995.

Counting of Teams in First-Order Team Logics

Anselm Haak
Theoretische Informatik, Leibniz Universität Hannover, Appelstraße, D-30167, Germany
haak@thi.uni-hannover.de

Juha Kontinen
Department of Mathematics and Statistics, University of Helsinki,
Pietari Kalmin katu 5, 00014, Finland
juha.kontinen@helsinki.fi

Fabian Müller
Theoretische Informatik, Leibniz Universität Hannover, Appelstraße, D-30167, Germany
fabian.mueller@thi.uni-hannover.de

Heribert Vollmer
Theoretische Informatik, Leibniz Universität Hannover, Appelstraße, D-30167, Germany
vollmer@thi.uni-hannover.de

Fan Yang
Department of Mathematics and Statistics, University of Helsinki,
Pietari Kalmin katu 5, 00014, Finland
fan.yang@helsinki.fi

Abstract

We study descriptive complexity of counting complexity classes in the range from #P to $\# \cdot \mathrm{NP}$. A corollary of Fagin's characterization of NP by existential second-order logic is that #P can be logically described as the class of functions counting satisfying assignments to free relation variables in first-order formulae. In this paper we extend this study to classes beyond #P and extensions of first-order logic with team semantics. These team-based logics are closely related to existential second-order logic and its fragments, hence our results also shed light on the complexity of counting for extensions of first-order logic in Tarski's semantics. Our results show that the class $\# \cdot \mathrm{NP}$ can be logically characterized by independence logic and existential second-order logic, whereas dependence logic and inclusion logic give rise to subclasses of $\# \cdot \mathrm{NP}$ and #P, respectively. We also study the function class generated by inclusion logic and relate it to the complexity class $\mathrm{TotP} \subsetneq \#\mathrm{P}$. Our main technical result shows that the problem of counting satisfying assignments for monotone Boolean Σ_1-formulae is $\# \cdot \mathrm{NP}$-complete with respect to Turing reductions as well as complete for the function class generated by dependence logic with respect to first-order reductions.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Complexity classes; Theory of computation → Complexity theory and logic

Keywords and phrases team-based logics, counting classes, finite model theory, descriptive complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.19

Related Version A full version of the paper is available at `https://arxiv.org/abs/1902.00246`.

Funding *Anselm Haak, Fabian Müller, Heribert Vollmer*: Supported by DFG VO 630/8-1 and DAAD grant 57348395.
Juha Kontinen, Fan Yang: Supported by grants 308712 and 308099 of the Academy of Finland.

Acknowledgements We thank the anonymous referees for helpful comments.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 19; pp. 19:1–19:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The question of the power of counting arises in propositional and predicate logic in a number of contexts. Counting the number of satisfying assignments for a given propositional formula, #SAT, is complete for Valiant's class #P of functions counting accepting paths of nondeterministic polynomial-time Turing machines [32]. Valiant also proved that #SAT even remains complete when restricted to monotone 2CNF-formulae.

The class #P can be seen as the counting analogue of NP, which was shown by Fagin [11] to correspond to existential second order logic, where the quantified relation encodes accepting computation paths of NP-machines. Hence, if we define $\#\mathrm{FO}^{\mathrm{rel}}$ to count satisfying assignments to free relational variables in first-order (FO-) formulae, we obtain $\#\mathrm{FO}^{\mathrm{rel}} = \#\mathrm{P}$. This result has been refined to prefix classes of FO showing, e.g., that $\#\Pi_2^{\mathrm{rel}} = \#\mathrm{P}$ [30].

If we define $\#\mathrm{FO}^{\mathrm{func}}$ in the same fashion as $\#\mathrm{FO}^{\mathrm{rel}}$ except that we count assignments to function variables instead of relation variables, then obviously $\#\mathrm{FO}^{\mathrm{func}} = \#\mathrm{P}$. The situation changes for the prefix classes, though. In particular, unlike for $\#\Pi_1^{\mathrm{rel}}$, it holds that $\#\Pi_1^{\mathrm{func}} = \#\mathrm{P}$, and, remarkably, also arithmetic circuit classes like $\#\mathrm{AC}^0$ can be characterized in this context [7].

In this paper we consider a different model-theoretic approach to the study of counting processes using so-called team-based logics. In these logics, formulae with free variables are evaluated not for single assignments to these variables but for *sets* of such assignments (called *teams*). Logics with team semantics have been developed for the study of various dependence and independence concepts important in many areas such as (probabilistic) databases and Bayesian networks (see, e.g. [16, 5, 15]) for which model counting is an important inference task (see, e.g., [3, 27]). In addition, team-based logics have interesting connections to a wide range of areas such as formal semantics of natural language [4], social choice theory [28], and quantum information theory [18].

In team semantics, a first-order formula is satisfied by a team if and only if all its members satisfy the formula individually. Interest in teams stems from the introduction of different logical atoms describing properties of teams, called team atoms, such as the value of a variable being functionally dependent on other variables (characterized by the *dependence* atom $=\!(\bar{x}, y)$), a variable being independent from other variables (characterized by the *independence* atom $\bar{y}\perp_{\bar{x}}\bar{z}$), and the values of a variable occurring as values of some other variable (characterized by the *inclusion* atom $\bar{x} \subseteq \bar{y}$), etc. ([31, 14, 12]).

We initiate in this paper the study of counting for team-based logics. In our proofs we utilize the known correspondences between team-based logics and existential second-order logic (Σ_1^1) and its fragments (see Theorem 2). We want to stress that our results are also novel for existential second-order logic and its fragments, and that there is no natural way to carry out the study of the function class generated by inclusion logic, that is, FO extended by the inclusion atom, in Tarskian semantics.

We define $\#\mathrm{FO}^{\mathrm{team}}$ to be the class of functions counting teams that satisfy a given FO-formula, and similarly for extensions of FO by team atoms. Making use of different team atoms, we give a characterization of $\# \cdot \mathrm{NP}$. While it is relatively easy to see that with every finite set A of NP-definable team atoms, the class $\#\mathrm{FO}(A)^{\mathrm{team}}$ stays a subclass of $\# \cdot \mathrm{NP}$ (Toda's generalization of #P, see [17] for a survey of counting classes like these), we show that FO extended with the independence atom is actually sufficient to characterize the full class $\# \cdot \mathrm{NP}$:

$$\#\mathrm{FO}(\perp)^{\mathrm{team}} = \#\Sigma_1^1 = \# \cdot \mathrm{NP}.$$

The situation with inclusion logic and dependence logic is more complex due to their strong closure properties: satisfaction of formulae is closed under union for inclusion logic and is closed downwards for dependence logic. We show that $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$ is a subclass of TotP, which is a class of counting problems with easy decision versions. Note that TotP is a strict subclass of #P unless P = NP and consequently the same holds for $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$. Furthermore, $\#\mathrm{FO}(=(\dots))^{\mathrm{team}}$ is a subclass of $\#\cdot\mathrm{NP}$, which we believe to be strict as well. Interestingly, both classes contain complete problems from their respective superclasses. In establishing this result for dependence logic, we introduce an interesting class of monotone quantified Boolean formulae and show that the corresponding counting problem where the all-0-assignment is not counted, $\#\Sigma_1\mathrm{CNF}^-_*$, is $\#\cdot\mathrm{NP}$-complete. In order to prove $\#\cdot\mathrm{NP}$-completeness we also show that the more natural problem of counting all satisfying assignments of the same class of formulae is $\#\cdot\mathrm{NP}$-complete by introducing a new technique of simultaneous reductions between pairs of counting problems, which we hope will also be useful in other contexts.

For inclusion logic we show that the well-known #P-complete problem $\#2\mathrm{CNF}^+$ is in $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$ and that the problem of counting assignments for existentially quantified dual-Horn formulae is hard for $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$.

In related previous work, so-called weighted logics have been used to logically characterize counting complexity classes [1], and the decision-problem analogue PP of #P and the counting hierarchy have been logically characterized in [21, 23, 6]. Counting classes from circuit complexity beyond $\#\mathrm{AC}^0$ have been logically characterized in [8].

Due to space restrictions, we only give proof sketches for some theorems, and detailed proofs for all results throughout the paper are deferred to the full version of this paper.

2 Definitions and Preliminaries

First-order Logic and Team Semantics

Let us start by recalling the syntax of first-order logic (FO). In this work, we only consider relational vocabularies (i.e., vocabularies with no function or constant symbols), and thus the only first-order terms are variables. Formulae of first-order logic are defined by the following grammar:

$$\varphi ::= \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x\varphi \mid \forall x\varphi \mid R(\overline{x}) \mid \neg R(\overline{x}) \mid x = y \mid x \neq y \qquad (\star)$$

where x, y are variables, R is a relation symbol, and $\overline{x}$ is a tuple of the appropriate number of variables.

The set of *free variables* of a formula φ is defined as usual, and we sometimes write $\varphi(x_1, \dots, x_k)$ to emphasize that the free variables of φ are among $x_1, \dots, x_k$. A formula with no free variable is called a *sentence*. For any k, the fragment Σ_k of FO consists of all formulae of the form $\exists\overline{x_1}\forall\overline{x_2}\dots Q\overline{x_k}\varphi$, where φ is quantifier-free and Q is either $\exists$ or $\forall$ depending on whether k is odd or even; similarly, the fragment Π_k is defined as the class of all formulae $\forall\overline{x_1}\exists\overline{x_2}\dots Q\overline{x_k}\varphi$ in prenex normal form with a quantifier prefix with k alternations starting with universal quantifiers.

We only consider finite structures with a finite relational vocabulary σ. Denote the class of all such structures by $\mathrm{STRUC}[\sigma]$, and let $\mathrm{dom}(\mathcal{A})$ denote the universe of a σ-structure $\mathcal{A}$. We will always use structures with universe $\{0, 1, \dots, n-1\}$ for some $n \in \mathbb{N} \setminus \{0\}$. We assume that our structures contain a *built-in binary relation* $\leq$ and *ternary relations* $+, \times$ with the usual interpretation, i.e., $\leq$ is interpreted in a model of any size as the "less than or equal to" relation on $\mathbb{N}$, $+$ is interpreted as addition and $\times$ as multiplication. We write $\mathrm{enc}_\sigma(\mathcal{A})$ for the standard binary encoding of a σ-structure $\mathcal{A}$ (see e.g., [20]): Relations are encoded row by row by listing their truth values as 0's and 1's. The whole structure is encoded by the concatenation of the encodings of its relations.

We assume that the reader is familiar with the usual Tarskian semantics for first-order formulae, in which formulae are evaluated with respect to single assignments of a structure. In this paper, we also consider so-called *team semantics* for first-order formulae, in which formulae are evaluated with respect to teams. A *team* is a *set* of assignments of a structure, that is, a set of functions $s\colon \{x_1, \dots, x_k\} \to \mathrm{dom}(\mathcal{A})$, where we call $\{x_1, \dots, x_k\}$ the domain of the team. Note that the empty set $\emptyset$ is a team, called *empty team*, and the singleton $\{\emptyset\}$ containing only the empty assignment is also a team. We denote by $\mathrm{team}(\mathcal{A}, (x_1, \dots, x_k))$ the set of all teams over a structure $\mathcal{A}$ with the domain $\{x_1, \dots, x_k\}$. Due to certain connections between team logics and second-order logic it is often helpful to view teams as relations. Let $\mathcal{A}$ be a structure and X a team of $\mathcal{A}$ with domain $\{x_1, \dots, x_k\}$. $\mathcal{A}$ and X induce the k-ary relation $\mathrm{rel}(X)$ on $\mathrm{dom}(\mathcal{A})$ defined as

$$\mathrm{rel}(X) := \{(s(x_1), \dots, s(x_n)) \mid s \in X\}.$$

Furthermore, for any subset $V \subseteq \{x_1, \dots, x_k\}$ of variables we define

$$X\big|_V := \{s\big|_V \mid s \in X\},$$

the restriction of team X to domain V.

We define inductively the notion of a team X with domain $\{x_1, \dots, x_k\}$ of a structure $\mathcal{A}$ with $A := \mathrm{dom}(\mathcal{A})$ satisfying an FO-formula $\varphi(x_1, \dots, x_k)$, denoted by $\mathcal{A} \models_X \varphi$, as follows:

- $\mathcal{A} \models_X \alpha$ for α an atomic formula if and only if for all $s \in X$, $\mathcal{A} \models_s \alpha$ in the usual Tarskian semantics sense.
- $\mathcal{A} \models_X \varphi \vee \psi$ if and only if there are teams $Y, Z \subseteq X$ such that $Y \cup Z = X$, $\mathcal{A} \models_Y \varphi$ and $\mathcal{A} \models_Z \psi$.
- $\mathcal{A} \models_X \varphi \wedge \psi$ if and only if $\mathcal{A} \models_X \varphi$ and $\mathcal{A} \models_X \psi$.
- $\mathcal{A} \models_X \exists x\varphi$ if and only if there exists a function $F\colon X \to \mathcal{P}(A) \setminus \{\emptyset\}$, called *supplementing function*, such that $\mathcal{A} \models_{X[F/x]} \varphi$, where

$$X[F/x] = \{s[a/x] \mid s \in X \text{ and } a \in F(s)\} \qquad \text{and} \quad s[a/x](y) = \begin{cases} a, & \text{if } x = y, \\ s(y), & \text{else.} \end{cases}$$

- $\mathcal{A} \models_X \forall x\varphi$ if and only if $\mathcal{A} \models_{X[A/x]} \varphi$, where $X[A/x] = \{s[a/x] \mid s \in X \text{ and } a \in A\}$.

A sentence φ is said to be *true* in $\mathcal{A}$, written $\mathcal{A} \models \varphi$, if $\mathcal{A} \models_{\{\emptyset\}} \varphi$.

FO-formulae φ are flat over team semantics, i.e., $\mathcal{A} \models_X \varphi$, if and only if $\mathcal{A} \models_s \varphi$ for all $s \in X$. In this sense, team semantics is conservative over FO-formulae. We now extend first-order logic by sets of atomic formulae which are not flat. For any sequence $\overline{x}$ of variables and variable y, the string $=\!(\overline{x}, y)$ is called a *dependence atom.* For any sequences $\overline{x}, \overline{y}, \overline{z}$ of variables, the string $\overline{y} \perp_{\overline{x}} \overline{z}$ is called an *independence atom.* For any two sequences $\overline{x}$ and $\overline{y}$ of variables of the same length, the string $\overline{x} \subseteq \overline{y}$ is called an *inclusion atom.* The team semantics of these atoms is defined as follows:

- $\mathcal{A} \models_X =\!(\overline{x}, y)$, if and only if for all $s, s' \in X$, if $s(\overline{x}) = s'(\overline{x})$, then $s(y) = s'(y)$.
- $\mathcal{A} \models_X \overline{y} \perp_{\overline{x}} \overline{z}$ if and only if for all $s, s' \in X$ such that $s(\overline{x}) = s'(\overline{x})$, there exists $s'' \in X$ such that $s''(\overline{x}) = s(\overline{x})$, $s''(\overline{y}) = s(\overline{y})$ and $s''(\overline{z}) = s'(\overline{z})$.
- $\mathcal{A} \models_X \overline{x} \subseteq \overline{y}$ if and only if for all $s \in X$ there is $s' \in X$ such that $s(\overline{x}) = s'(\overline{y})$.

For any subset $A \subseteq \{=(\dots), \perp, \subseteq\}$, we define $\mathrm{FO}(A)$ as first-order logic extended by the respective atoms, and refer to such a logic as *team-based logic*. More precisely we extend the grammar $(\star)$ by adding a rule for each atom in A. For example for $\mathrm{FO}(\{\subseteq\})$ we add the rule

$$\varphi ::= \overline{x} \subseteq \overline{y},$$

where $\overline{x}, \overline{y}$ are tuples of variables. For convenience, we often omit the curly brackets and write for example $\mathrm{FO}(\subseteq)$ instead of $\mathrm{FO}(\{\subseteq\})$.

The team-based logic $\mathrm{FO}(=(\dots))$ is known in the literature as *dependence logic* [31], $\mathrm{FO}(\perp)$ as *independence logic* [14] and $\mathrm{FO}(\subseteq)$ as *inclusion logic* [12]. We recall some basic properties of these logics from [31, 14, 12]: Formulae of $\mathrm{FO}(=(\dots))$ are *closed downwards*, i.e., $\mathcal{A} \models_X \varphi$ and $Y \subseteq X$ implies $\mathcal{A} \models_Y \varphi$, formulae of $\mathrm{FO}(\subseteq)$ are *closed under unions*, i.e., $\mathcal{A} \models_X \varphi$ and $\mathcal{A} \models_Y \varphi$ implies $\mathcal{A} \models_{X \cup Y} \varphi$, and formulae of any of these logics have the *empty team property*, i.e., $\mathcal{A} \models_\emptyset \varphi$ always holds.

The above atoms expressing team properties can be generalized, as we will do next. Let us first recall below the definition of generalized quantifiers, where we follow the notations from [22, 26].

▶ **Definition 1.** *Let $i_1, \dots, i_n$ $(n > 0)$ be a sequence of positive integers, and σ a vocabulary consisting of an i_j-ary relation symbol for each $1 \leq j \leq n$. A* generalized quantifier *of type $(i_1, \dots, i_n)$ is a class $\mathcal{C}$ of σ-structures $(A, B_1, \dots, B_n)$ such that the following conditions hold:*

1. *$A \neq \emptyset$ and for each $1 \leq j \leq n$, we have $B_j \subseteq A^{i_j}$.*
2. *$\mathcal{C}$ is closed under isomorphisms, that is, if $(A', B'_1, \dots, B'_n) \in \mathcal{C}$ is isomorphic to $(A, B_1, \dots, B_n)$, then $(A', B'_1, \dots, B'_n) \in \mathcal{C}$.*

Let Q be a generalized quantifier of type $(i_1, \dots, i_n)$. Let us extend the syntax of first-order logic with an expression $A_Q(\overline{x_1}, \dots, \overline{x_n})$, where each $\overline{x_j}$ is a tuple of variables of length i_j and $Vars(\overline{x_i})$ is the set of variables in x_i. We call A_Q a generalized (dependency) atom *(of type $(i_1, \dots, i_n)$), and its team semantics is defined as:*

$$\mathcal{A} \models_X A_Q(\overline{x_1}, \dots, \overline{x_n}) \textit{ if and only if} (\mathrm{rel}(X|_{Vars(\overline{x_1})}), \dots, \mathrm{rel}(X|_{Vars(\overline{x_n})})) \in Q^{\mathcal{A}},$$

where $Q^{\mathcal{A}} = \{(B_1, \dots, B_n) \mid (\mathrm{dom}(\mathcal{A}), B_1, \dots, B_n) \in Q\}$.

We say that a generalized dependency atom A_Q is NP-definable if there is an NP-algorithm that decides for a given structure $\mathcal{A}$ and a given team X whether $\mathcal{A} \models_X A_Q(\overline{x_1}, \dots, \overline{x_n})$ holds or not. A set A of generalized atoms is NP-definable if every $a \in A$ is NP-definable. For example, the set $A = \{=(\dots), \perp, \subseteq\}$ is NP-definable.

Many results in this paper are based on the expressive power of the logics defined above, that we shall now recall. We first recall some notions and notations. Existential second-order logic (Σ^1_1) consists of formulae of the form $\exists R_1 \dots \exists R_k \varphi$, where φ is an FO-formula. Let σ be a vocabulary. We write $\sigma(R)$ for the vocabulary that arises by adding a fresh relation symbol R to σ, and we sometimes write $\varphi(R)$ to emphasize that the relation symbol R occurs in the $\sigma(R)$-formula φ. If $\mathcal{A}$ is a σ-structure, we write $(\mathcal{A}, Q)$ for $\mathcal{A}$ expanded into a $\sigma(R)$-structure where the new k-relation symbol R is interpreted as $Q \subseteq \mathrm{dom}(\mathcal{A})^k$. A $\sigma(R)$-sentence $\varphi(R)$ of Σ^1_1 is said to be *downward monotone* with respect to R if $(\mathcal{A}, Q) \models \varphi(R)$ and $Q' \subseteq Q$ imply $(\mathcal{A}, Q') \models \varphi(R)$. It is known that $\varphi(R)$ is downward monotone with respect to R if and only if $\varphi(R)$ is equivalent to a sentence where R occurs only negatively (see e.g., [24]).

▶ **Theorem 2** (see [12, 24, 13]).

1. *For every σ-formula φ of* $\mathrm{FO}(\perp)$*, there is an $\sigma(R)$-sentence $\psi(R)$ of Σ^1_1 such that for all σ-structures $\mathcal{A}$ and teams X,*

$$\mathcal{A} \models_X \varphi \iff (\mathcal{A}, \mathrm{rel}(X)) \models \psi(R). \tag{1}$$

Conversely, for every $\sigma(R)$-sentence $\psi(R)$ of Σ^1_1, there is a σ-formula φ of $\mathrm{FO}(\perp)$ *such that (1) holds for all σ-structures $\mathcal{A}$ and* non-empty *teams X.*

2. *The same as the above holds for formulae of* $\mathrm{FO}(=\!(\dots))$ *as well, except that in both directions for* $\mathrm{FO}(=\!(\dots))$ *the relation symbol R is assumed to occur only negatively in the sentence $\psi(R)$.*
3. *In particular, over sentences both* $\mathrm{FO}(\perp)$ *and* $\mathrm{FO}(=\!(\dots))$ *are expressively equivalent to Σ^1_1, in the sense that every σ-sentence of* $\mathrm{FO}(\perp)$ *(or* $\mathrm{FO}(=\!(\dots))$*) is equivalent to a σ-sentence ψ of Σ^1_1, i.e., for any σ-structure $\mathcal{A}$,*

$$\mathcal{A} \models \varphi \iff \mathcal{A} \models \psi,$$

and vice versa. As a consequence of Fagin's Theorem (see [11]), over finite structures both $\mathrm{FO}(\perp)$ *and* $\mathrm{FO}(=\!(\dots))$ *capture* NP*.*

4. *For any σ-formula $\varphi(x_1, \dots, x_k)$ of* $\mathrm{FO}(\subseteq)$*, there exists a $\sigma(R)$-formula $\psi(R)$ of positive greatest fixed point logic (*posGFP*) such that for all σ-structures $\mathcal{A}$ and teams X,*

$$\mathcal{A} \models_X \varphi \iff \mathcal{A}, \mathrm{rel}(X) \models_s \psi(R) \text{ for all } s \in X;$$

and vice versa. In particular, over sentences $\mathrm{FO}(\subseteq)$ *is expressively equivalent to* posGFP*. As a consequence of Immerman's Theorem (see [19]), over finite structures,* $\mathrm{FO}(\subseteq)$ *is expressively equivalent to least fixed point logic (*LFP*). Thus, by [19, 34], over ordered finite structures,* $\mathrm{FO}(\subseteq)$ *captures* P*.*

5. *Let $\varphi(R)$ be a* myopic *σ-formula, that is, $\varphi(R) = \forall \overline{x}(R(\overline{x}) \to \psi(R, \overline{x}))$, where ψ is a first order σ-formula with only positive occurrences of R. Then there exists a σ-formula $\chi \in$* $\mathrm{FO}(\subseteq)$ *such that for all σ-structures $\mathcal{A}$ and all teams X:*

$$\mathcal{A} \models_X \chi(\overline{x}) \Leftrightarrow \mathcal{A}, \mathrm{rel}(X) \models \varphi(R).$$

Propositional and Quantified Boolean formulae

In this paper, we will also consider certain classes of propositional and quantified Boolean formulae. As usual, we use CNF to denote the class of propositional formulae in conjunctive normal form and k-CNF to denote the class of propositional formulae in conjunctive normal form where each clause contains at most k literals. A formula in CNF is in the class DualHorn if each of its clauses contains at most one negative literal.

For a class $\mathcal{C}$ of Boolean formulae, we denote by $\Sigma_1\mathcal{C}$ the class of quantified Boolean formulae in prenex normal form with only existential quantifiers where the quantifier-free part is an element of $\mathcal{C}$.

For a class $\mathcal{C}$ of quantified Boolean formulae we denote with $\mathcal{C}^+$(resp. $\mathcal{C}^-$) the class of formulae in $\mathcal{C}$ whose free variables occur only positively (resp. negatively). For example, $\Sigma_1 3\mathrm{CNF}^-$ consists of all quantified Boolean formulae in prenex normal form with only existential quantifiers, where the quantifier-free part is in 3CNF and the free variables occur only negatively. Note that in Boolean formulae all variables are free.

Counting Problems and Counting Classes

This paper aims to identify model-theoretic characterizations of counting classes in terms of team-based logics. Let us now recall relevant previous results on the descriptive complexity of counting problems. We begin by defining the most important complexity classes for counting problems.

▶ **Definition 3.** *A function $f\colon \{0,1\}^* \to \mathbb{N}$ is in* $\#\mathrm{P}$ *if there is a non-deterministic polynomial time Turing machine M such that for all inputs $x \in \{0,1\}^*$,*

$$f(x) \text{ is the number of the accepting computation paths of } M \text{ on input } x.$$

This definition can be generalized as follows.

▶ **Definition 4.** *Let $\mathcal{C}$ be a complexity class. A function $f\colon \{0,1\}^* \to \mathbb{N}$ is said to be in $\# \cdot \mathcal{C}$ if there are a language $L \in \mathcal{C}$ and a polynomial p such that for all $x \in \{0,1\}^*$:*

$$f(x) = |\{y \mid |y| \leq p(|x|) \text{ and } (x,y) \in L\}|.$$

Obviously $\#\mathrm{P} = \# \cdot \mathrm{P}$, and it is well known that $\#\mathrm{P} \subseteq \# \cdot \mathrm{NP} \subseteq \# \cdot \mathrm{coNP} = \#\mathrm{P}^{\mathrm{NP}}$, where, under reasonable complexity-theoretic assumptions, all these inclusions are strict; see [17] for a survey of these issues.

▶ **Definition 5.** *A function $f\colon \{0,1\}^* \to \mathbb{N}$ is in* TotP *if there is a non-deterministic polynomial time Turing machine M such that for all inputs $x \in \{0,1\}^*$,*

$$f(x) \text{ is the number of the computation paths of } M \text{ on input } x \text{ minus } 1.$$

Subtracting 1 from the number of computation paths is neccessary since otherwise TotP-functions could never map to 0. In [29] it was shown that TotP is the closure with respect to parsimonious reductions of self-reducible counting problems from $\#\mathrm{P}$ whose decision version is in P. It follows that $\mathrm{TotP} \subsetneq \#\mathrm{P}$ unless $\mathrm{P} = \mathrm{NP}$.

Next, we define the relevant logical counting classes.

▶ **Definition 6.** *A function $f\colon \{0,1\}^* \to \mathbb{N}$ is said to be in $\#\mathrm{FO}^{\mathrm{rel}}$ if there is a vocabulary σ with a built-in linear order $\leq$, and an* FO*-formula $\varphi(R_1, \ldots, R_k, x_1, \ldots, x_\ell)$ over σ with free relation variables $R_1, \ldots, R_k$ and free individual variables $x_1, \ldots, x_\ell$ such that for all σ-structures $\mathcal{A}$,*

$$f(\mathrm{enc}_\sigma(\mathcal{A})) = |\{(S_1, \ldots, S_k, c_1, \ldots, c_\ell) : \mathcal{A} \models \varphi(S_1, \ldots, S_k, c_1, \ldots, c_\ell\}|.$$

If the input of f is not of the appropriate form, we assume the output to be 0*.*

In the same fashion, subclasses of $\#\mathrm{FO}^{\mathrm{rel}}$, such as $\#\Sigma_k^{\mathrm{rel}}$ and $\#\Pi_k^{\mathrm{rel}}$ for arbitrary k, are defined by assuming that the formula φ in the above definition is in the corresponding fragments Σ_k and Π_k.

Recall the relationship between the above defined logical counting classes and $\#\mathrm{P}$:

▶ **Theorem 7** ([30])**.** $\#\Sigma_0^{\mathrm{rel}} = \#\Pi_0^{\mathrm{rel}} \subset \#\Sigma_1^{\mathrm{rel}} \subset \#\Pi_1^{\mathrm{rel}} \subset \#\Sigma_2^{\mathrm{rel}} \subset \#\Pi_2^{\mathrm{rel}} = \#\mathrm{FO}^{\mathrm{rel}} = \#\mathrm{P}$*. Furthermore, $\#\Sigma_0^{\mathrm{rel}} \subseteq \mathrm{FP}$.*

Complete problems (i.e., hardest problems) for counting classes have also been studied extensively. Let us now recall three reductions that are relevant in this study. Let f and h be counting problems. We say that f is *parsimoniously* reducible to h if there is a polynomial-time computable function g such that $f(x) = h(g(x))$ for all inputs x, f is *Turing* reducible to h if $f \in \mathrm{FP}^h$, and f is *metrically* reducible to h if there are polynomial-time computable functions g_1, g_2 such that $f(x) = g_2(h(g_1(x)), x)$ for all inputs x. Clearly, metric reductions are Turing reductions with one oracle query. Besides these three familiar reductions we now define another type of reductions, called *first-order reductions.* First recall that for any two vocabularies σ_1, σ_2, an *FO-interpretation* (or a *first-order query*) is a function $I : \mathrm{STRUC}[\sigma_1] \to \mathrm{STRUC}[\sigma_2]$, represented as a tuple $I = (\varphi_0, \varphi_{R_1}, \ldots, \varphi_{R_\ell})$ of FO-formulae over σ_1 with k free variables, that maps any structure $\mathcal{A} \in \mathrm{STRUC}[\sigma_1]$ to another structure $I(\mathcal{A}) \in \mathrm{STRUC}[\sigma_2]$, whose domain is a subset of $\mathrm{dom}(\mathcal{A})^k$ (i.e., a set of k-ary tuples of elements in $\mathcal{A}$) defined by φ_0 and relations R_i are defined by φ_{R_i} (see [20] for detailed discussion). In the team semantics case, we also need to define how teams are transformed by the interpretation I. The value $I(X)$ is defined in a straightforward way: individual elements from $\mathcal{A}^{k \cdot m}$ in X are mapped to elements from $I(\mathcal{A})^m$ in $I(X)$, where k is the arity of tuples in the domain of the structure $I(\mathcal{A})$, and m is the arity of the team $I(X)$. Now, we define first-order reductions via FO-interpretations as follows:

▶ **Definition 8.** *Let f_1, f_2 be two functions. We say f_1 is first-order reducible or FO-reducible to f_2 (denoted $f_1 \leq_{fo} f_2$) if there are vocabularies σ_1, σ_2 with $\sigma_2 = (R_1^{a_1}, \ldots, R_l^{a_l})$ and an FO-interpretation $I = (\varphi_0, \varphi_{R_1}, \ldots, \varphi_{R_\ell})$, where $\varphi_0, \varphi_{R_1}, \ldots, \varphi_{R_\ell}$ are* FO*-formulae over σ_1, such that for all $\mathcal{A}_1 \in \mathrm{STRUC}[\sigma_1]$, there are $k \in \mathbb{N}$ and $\mathcal{A}_2 \in \mathrm{STRUC}[\sigma_2]$ with*

$$dom(\mathcal{A}_2) = \{(x^1, \ldots, x^k) \mid \mathcal{A}_1 \models \varphi_0(x^1, \ldots, x^k)\}$$

and for all $i \leq k$

$$R_i((x_1^1, \ldots, x_1^k), \ldots, (x_{a_i}^1, \ldots, x_{a_i}^k)) \Leftrightarrow \mathcal{A}_1 \models \varphi_{R_i}(x_1^1, \ldots, x_1^k, \ldots, x_{a_i}^1, \ldots, x_{a_i}^k)$$

and $f_1(\mathrm{enc}_{\sigma_1}(\mathcal{A}_1)) = f_2(\mathrm{enc}_{\sigma_2}(\mathcal{A}_2))$.

It is often possible to find complete problems in counting classes by counting satisfying assignments for certain (quantified) Boolean formulae. Let $\mathcal{F}$ be a class of quantified Boolean formulae. Define the problem $\#\mathcal{F}$ as follows:

Problem:	$\#\mathcal{F}$
Input:	Formula $\varphi \in \mathcal{F}$
Output:	Number of satisfying assignments of φ

For example, #SAT, the function counting the number of satisfying assignments for propositional formulae, as well as its restriction #3CNF, are complete for #P under parsimonious reductions, while $\#2\mathrm{CNF}^+$ and $\#2\mathrm{CNF}^-$ are complete for #P under Turing reductions. Observe that for all Σ_1CNF-formulae φ it holds that the number of satisfying assignments is equal to that of the formula $\widetilde{\varphi}$ obtained by negating all literals in all clauses in φ. Thus, $\#\Sigma_1\mathrm{CNF}^+$ and $\#\Sigma_1\mathrm{CNF}^-$ are in a sense the same problem. In fact all our results for $\#\mathcal{C}^+$ (for a class of formulae $\mathcal{C}$) also hold for $\#\mathcal{C}^-$ and vice versa. This also holds for #Horn and #DualHorn. Note that Aziz et al [2] studied the problem $\#\Sigma_1\mathrm{SAT}$ under the name *projected model counting* and observed that it is contained in $\# \cdot \mathrm{NP}$.

Next we introduce the central class for this paper, a class of counting problems in the context of team-based logics. For any set A of generalized dependency atoms, we define $\#\mathrm{FO}(A)^{\mathrm{team}}$ to consist of those functions counting non-empty satisfying teams for $\mathrm{FO}(A)$-formulae. Note that by the empty team property of dependence, independence, and inclusion logic formulae any function that counts all satisfying teams (including the empty team) could not attain the value 0.

▶ **Definition 9.** *For any set A of generalized atoms, $\#\mathrm{FO}(A)^{\mathrm{team}}$ is the class of all functions $f\colon \{0,1\}^* \to \mathbb{N}$ for which there is a vocabulary σ with a built-in linear order $\leq$ and an $\mathrm{FO}(A)$-formula $\varphi(\overline{x})$ over σ with a tuple $\overline{x}$ of free first-order variables such that for all σ-structures $\mathcal{A}$,*

$$f(\mathrm{enc}_\sigma(\mathcal{A})) = |\{X \in \mathit{team}(\mathcal{A}, (\overline{x})) : X \neq \emptyset \text{ and } \mathcal{A} \models_X \varphi(\overline{x})\}|.$$

We denote by f_φ the function defined by φ.

▶ **Example 10.** As an example for how to work with team semantics in a counting context, we show that the #P-complete problem $\#2\mathrm{CNF}^+$ is contained in both $\#\mathrm{FO}(=\!(\dots))^{\mathrm{team}}$ and $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$. Let $\varphi(x_1,\dots,x_n) = \bigwedge C_i \in 2\mathrm{CNF}^+$, where each $C_i = \ell_{i,1} \vee \ell_{i,2}$ and $\ell_{i,j} \in \{x_1,\dots,x_n\}$. Consider the vocabulary $\tau_{2\mathrm{CNF}^+} = \{C^2\}$. We encode the formula $\varphi(x_1,\dots,x_n)$ by the structure $\mathcal{A} = (\{x_1,\dots,x_n\}, C^{\mathcal{A}})$, where $(x,y) \in C^{\mathcal{A}}$ if and only if the clause $x \vee y$ occurs in φ.

We show that $\#2\mathrm{CNF}^+$ can be defined by counting non-empty teams (which correspond to assignments mapping at least one variable to true) satisfying suitable formulae from $\mathrm{FO}(\subseteq)$ as well as $\mathrm{FO}(=\!(\dots))$. For this purpose, we encode Boolean assignments to the variables $x_1,\dots,x_n$ by teams over one variable t. Since the universe of our structures is exactly the set of variables of the formula $\varphi(x_1,\dots,x_k)$ in question, assignments to the variables can be encoded by inclusion of the values of the variables x_i in t in the team.

Now, the following $\mathrm{FO}(\subseteq)$-formula defines $\#2\mathrm{CNF}^+$:

$$\varphi_{\subseteq}(t) = \forall x \forall y (\neg C(x,y) \vee x \subseteq t \vee y \subseteq t).$$

Intuitively, this formula states that if a pair (x,y) of variables in φ occur in the same clause (i.e., $C(x,y)$ holds), then the value of one of the two variables x,y is contained in the team, or it is set to true.

To define the same problem in $\mathrm{FO}(=\!(\dots))$ where inclusion atoms are not any more available in the language, we need to encode assignments differently. We now encode variables x_i being set to 1 by not including them in the team over the variable t. The $\mathrm{FO}(=\!(\dots))$-formula that defines $\#2\mathrm{CNF}^+$ is the following:

$$\begin{aligned}\varphi_{=(\dots)}(t) =& \exists \min \forall z \min \leq z \wedge \\ & \quad \forall x \forall y \exists x' \exists y' \Big(=\!(x,x') \wedge =\!(y,y') \wedge (\neg x = y \vee x' = y') \wedge \\ & \qquad (x \neq t \vee x' = \min) \wedge (\neg C(x,y) \vee x' \neq \min \vee y' \neq \min) \Big).\end{aligned}$$

Intuitively, in the above formula we use existential quantifications together with dependence atoms to express that x' is a function of x, and this function f is guaranteed in the formula to be consistent with the assignment encoded by the team. We shall interpret a function mapping to the minimal element of the universe (encoded by the variable min in the formula) as an assignment to 0, and a function mapping to any other element as an assignment to 1. Now, the second last conjunct in our formula states that all variables that occur as values of t in the team (which correspond to those x_i set to 0) are mapped by our function f to the minimal element. Finally, the last conjunct in our formula checks whether the 2CNF-formula is satisfied by the assignment encoded by f. Note that in order to talk about the assignment to two variables simultaneously, in the above formula we actually use two (equivalent) functions to encode the same assignment.

3 A Characterization of the Class $\# \cdot \mathrm{NP}$

In this section, we characterize the class $\# \cdot \mathrm{NP}$ in terms of team-based logics. Our first result shows that $\# \cdot \mathrm{NP}$ is the largest class attainable by counting teams in team-based logics $\mathrm{FO}(A)$, as long as all generalized atoms in A are NP-definable.

▶ **Theorem 11.** *For any set A of* NP*-definable generalized atoms,* $\#\mathrm{FO}(A)^{\mathrm{team}} \subseteq \# \cdot \mathrm{NP}$.

Proof Sketch. Let $\varphi(\overline{x}) \in \#\mathrm{FO}(A)^{\mathrm{team}}$. To show that $f_\varphi \in \# \cdot \mathrm{NP}$ we note that f_φ can be computed by counting on input $\mathrm{enc}_\sigma(\mathcal{A})$ the number of teams X such that $\mathcal{A} \models_X \varphi(\overline{x})$. It is thus sufficient to show that the letter can be checked in NP on input $(\mathrm{enc}_\sigma(\mathcal{A}), X)$. In this proof, the only places that involve nondeterminism are disjunctions (guess the split), existential quantifiers (guess the supplementing function) and NP-definable generalized atoms (checkable in NP by definition). ◀

Next, we prove the converse inclusion of the above theorem by proving a stronger result: The whole class $\#{\cdot}\mathrm{NP}$ can actually be captured by a single generalized atom, the independence atom.

▶ **Theorem 12.** $\# \cdot \mathrm{NP} \subseteq \#\mathrm{FO}(\perp)^{\mathrm{team}}$

Proof Sketch. It is sufficient to show $\# \cdot \mathrm{NP} \subseteq \#\Sigma_1^1$ since by Theorem 2.1 we have $\#\Sigma_1^1 = \#\mathrm{FO}(\perp)^{\mathrm{team}}$.

Let $f \in \# \cdot \mathrm{NP}$. Then there are a polynomial p and $L \in \mathrm{NP}$ such that

$$f(x) = |\{y \mid |y| = p(|x|), (x, y) \in L\}|.$$

We encode tuples (x, y) of strings with $|y| = p(|x|)$ as structures $\mathcal{A}_{(x,y)}$ by encoding the string x as a structure $\mathcal{A}_x$ in the standard way, and y as a unique relation R_y over $\mathrm{dom}(\mathcal{A}_x)^k$ for some $k \in \mathbb{N}$. Finally, Fagin's theorem gives a Σ_1^1-sentence φ such that for all x,

$$|\{y \mid |y| = p(|x|), (x, y) \in L\}| = |\{y \mid \mathcal{A}_{(x,y)} \models \varphi\}| = |\{R \mid \mathcal{A}_x \models \varphi(R)\}|. \quad ◀$$

▶ Remark 13. The class $\#\mathrm{P}$ can also be characterized by counting teams. A variant $\mathcal{L}$ of dependence logic that defines exactly the first-order definable team properties in the sense of Theorem 2 was introduced in [25]. Since $\#\mathrm{P} = \#\mathrm{FO}$ (see [30]), this logic $\mathcal{L}$ captures $\#\mathrm{P}$. We do not present the details of $\mathcal{L}$ in this paper, but only note that $\mathcal{L}$ has weaker versions of quantifiers and disjunction instead of the standard ones as defined in Section 2.

4 Counting Teams in Dependence and Inclusion Logic

In this section, we study the smaller classes $\#\mathrm{FO}(=(\dots))^{\mathrm{team}}$ and $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$. We begin by showing that the $\# \cdot \mathrm{NP}$-complete problem $\#\Sigma_1\mathrm{CNF}^-_*$, defined below, is contained in $\#\mathrm{FO}(=(\dots))^{\mathrm{team}}$. We will show $\# \cdot \mathrm{NP}$-completeness for $\#\Sigma_1\mathrm{CNF}^-_*$ in Theorem 26.

Problem:	$\#\Sigma_1\mathrm{CNF}^-_*$
Input:	Formula $\varphi(x_1, \dots, x_k) \in \mathrm{CNF}^-$
Output:	Number of satisfying assignments of φ, disregarding the all-0-assignment

Note that the *all-0-assignment* is the assignment mapping each variable to 0.

▶ **Theorem 14.** $\#\Sigma_1\mathrm{CNF}^-_* \in \#\mathrm{FO}(=(\dots))^{\mathrm{team}}$.

We will show that the above problem is actually complete for $\#\mathrm{FO}(=(\dots))^{\mathrm{team}}$ with respect to first-order reductions. First-order reductions turn out to be particularly natural in our context, as all our classes are closed under these reductions.

▶ **Theorem 15.** *$\#\mathrm{FO}(A)^{\mathrm{team}}$ is closed under first-order reductions for $A \subseteq \{=(\dots), \perp, \subseteq\}$.*

Next we show that the problem $\#\Sigma_1\mathrm{CNF}^-_*$ is hard and thus complete for $\#\mathrm{FO}(=(\dots))^{\mathrm{team}}$ under first-order reductions. Our proof technique is similar to that of [9], where the data complexity of inclusion logic is shown to be polynomial.

▶ **Theorem 16.** *$\#\Sigma_1\mathrm{CNF}^-_*$ is complete for $\#\mathrm{FO}(=(\dots))^{\mathrm{team}}$ with respect to first-order reductions.*

Having proven our results for dependence logic $\mathrm{FO}(=(\dots))$, we now turn to inclusion logic $\mathrm{FO}(\subseteq)$. We first prove that $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$ is a subclass of $\#\mathrm{P}$.

▶ **Theorem 17.** *$\#\mathrm{FO}(\subseteq)^{\mathrm{team}} \subseteq \#\mathrm{P}$.*

The above theorem naturally gives rise to the question whether $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$ actually coincides with $\#\mathrm{P}$. However, we identify in the next lemma a particular property of $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$ functions, making this equivalence unlikely to hold.

▶ **Lemma 18.** *Let $\varphi(\overline{x}) \in \mathrm{FO}(\subseteq)$ be a formula over a vocabulary σ. Then the language $L := \{w \mid f_\varphi(w) > 0\}$ is in P.*

▶ **Corollary 19.** *If $\mathrm{P} \neq \mathrm{NP}$, then $\#\mathrm{FO}(\subseteq)^{\mathrm{team}} \neq \#\mathrm{P}$.*

Theorem 17 and Corollary 19 indicate that $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$ is most likely a strict subclass of $\#\mathrm{P}$. Nevertheless, we show in the next theorem that $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$ contains the problem #DualHorn which is complete for $\#\mathrm{P}$ with respect to Turing reductions. It is unknown whether $\#\mathrm{DualHorn} \in \#\mathrm{FO}(=(\dots))$.

▶ **Theorem 20.** *$\#\mathrm{DualHorn} \in \#\mathrm{FO}(\subseteq)$.*

We continue by exhibiting a hard problem for the class $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$. It is an open question whether the problem is definable by an inclusion logic formula.

▶ **Theorem 21.** *$\#\Sigma_1\mathrm{DualHorn}$ is hard for $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$ with respect to first-order reductions.*

It seems to us that $\#\mathrm{FO}(\subseteq)$ is a strict subclass of $\#\mathrm{P}$ and the decision versions of problems in $\#\mathrm{FO}(\subseteq)$ are in P. For this reason, we now investigate relationship between $\#\mathrm{FO}(\subseteq)$ and the subclass TotP. We show that $\#\mathrm{FO}(\subseteq)$ is a subclass of TotP and that TotP contains $\#\Sigma_1\mathrm{DualHorn}$. We conjecture that these classes do not coincide, but this question remains open.

▶ **Theorem 22.** *$\#\mathrm{FO}(\subseteq) \subseteq \mathrm{TotP}$*

▶ **Theorem 23.** *$\#\Sigma_1\mathrm{DualHorn} \in \mathrm{TotP}$*

5 Complete Problems for $\# \cdot \mathrm{NP}$

In this section we show that $\#\Sigma_1\mathrm{CNF}^-_*$ is $\# \cdot \mathrm{NP}$-complete. To this end, we first observe that $\#\Sigma_1\mathrm{CNF}$ is $\# \cdot \mathrm{NP}$-complete. Afterwards we show that the smaller class $\#\Sigma_1\mathrm{CNF}^-$ remains $\# \cdot \mathrm{NP}$-complete by adapting the proof for $\#\mathrm{P}$-completeness of $\#2\mathrm{CNF}^+$ given by Valiant [33]. We conclude this section with a reduction from $\#\Sigma_1\mathrm{CNF}^-$ to $\#\Sigma_1\mathrm{CNF}^-_*$ showing the $\# \cdot \mathrm{NP}$-completeness of the latter.

▶ **Lemma 24.** $\#\Sigma_1\mathrm{SAT}$ *and* $\#\Sigma_1\mathrm{CNF}$ *are* $\# \cdot \mathrm{NP}$*-complete under parsimonious reductions.*

▶ **Theorem 25.** $\#\Sigma_1\mathrm{CNF}^-$ *is* $\# \cdot \mathrm{NP}$*-complete under Turing reductions.*

Proof Sketch. Membership follows from 24, since $\#\Sigma_1\mathrm{CNF}^-$ is a special case of $\#\Sigma_1\mathrm{CNF}$. For the hardness proof, we show a chain of reductions adapted from the one used by Valiant [33] to show the #P-completeness of $\#2\mathrm{CNF}^+$. Recall that the main steps of Valiant's chain of reductions are:

$$\begin{aligned} \#3\mathrm{CNF} \leq \mathrm{PERMANENT} & \leq \#\mathrm{PERFECT\text{-}MATCHING} \\ \leq \#\mathrm{IMPERFECT\text{-}MATCHING} & \leq \#2\mathrm{CNF}^+. \end{aligned}$$

Our idea is to add a $\Sigma_1 3\mathrm{CNF}$-formula to the input of each problem in the above chain of reductions, and to express certain properties of the respective inputs in the added formula. We then count only the solutions to the input that also satisfy the added formula.

As part of the chain of reductions we will make use of the following problem:

Problem:	$\#(3\mathrm{CNF}, \Sigma_1 3\mathrm{CNF}^-)$
Input:	Formula $\varphi(x_1, \dots, x_k) \in 3\mathrm{CNF}$ and formula $\psi(x_1, \dots, x_k) \in \Sigma_1 3\mathrm{CNF}^-$
Output:	Number of satisfying assignments of $\varphi \wedge \psi$

We will reduce $\#\Sigma_1 3\mathrm{CNF}$ to $\#(3\mathrm{CNF}, \Sigma_1 3\mathrm{CNF}^-)$, and then apply the above chain of reductions (with added formulae, as described above). This results in a reduction to $\#(2\mathrm{CNF}^-, \Sigma_1 3\mathrm{CNF}^-)$, defined analogously to the above problem. Finally it is straightforward to show $\#(2\mathrm{CNF}^-, \Sigma_1 3\mathrm{CNF}^-) \leq \#\Sigma_1\mathrm{CNF}^-$ using the fact that for $\varphi \in 2\mathrm{CNF}^-$ and $\psi \in \Sigma_1 3\mathrm{CNF}^-$, the prenex normal form of $\varphi \wedge \psi$ is a $\Sigma_1 3\mathrm{CNF}^-$-formula. We conclude by sketching the first reduction.

$\underline{\#\Sigma_1 3\mathrm{CNF} \leq \#(3\mathrm{CNF}, \Sigma_1 3\mathrm{CNF}^-)}$: We construct for any $\varphi \in \Sigma_1 3\mathrm{CNF}$ two formulae $\varphi' \in 3\mathrm{CNF}$ and $\psi \in \Sigma_1 3\mathrm{CNF}^-$ such that $\#\Sigma_1 3\mathrm{CNF}(\varphi) = \#(3\mathrm{CNF}, \Sigma_1 3\mathrm{CNF}^-)(\varphi', \psi)$.

Assume $\varphi = \exists y_1 \dots \exists y_\ell \bigwedge C_i \wedge \bigwedge D_i \wedge \bigwedge E_i$, where clauses C_i only contain free variables of φ, clauses D_i contain only bound variables of φ, and clauses E_i contain at least one free and at least one bound variable of φ. We can now simply add all clauses C_i to φ' and all clauses D_i to ψ.

To handle the remaining clauses, we add for each E_i a new free variable e_i. We then express in φ' that e_i is true if and only if clause E_i is not satisfied by the assignment to the free variables, and express in ψ that E_i has to be satisfied by the assignment to the bound variables if e_i is true. The former does not involve any bound variables and for the latter, the only needed free variable is e_i, which only occurs negatively. ◀

Because of the special role of the empty team in the team logics we consider, we will also show the completeness for another version of $\#\Sigma_1\mathrm{CNF}^-$, denoted as $\#\Sigma_1\mathrm{CNF}^-_*$, for which the all-0-assignment is not counted.

▶ **Theorem 26.** *The problem* $\#\Sigma_1\mathrm{CNF}^-_*$ *is* $\# \cdot \mathrm{NP}$*-complete under Turing reductions.*

6 Conclusion

In this paper we have studied the following hierarchy of classes defined by counting problems for team-based logics:

$$\begin{array}{ccccc}
\mathrm{TotP} & \subseteq \#\mathcal{L}^{\mathrm{team}} = \#\mathrm{P} & \subseteq & \#\mathrm{FO}(\perp)^{\mathrm{team}} = \# \cdot \mathrm{NP} \\
\rotatebox[origin=c]{90}{$\subseteq$} & & & \rotatebox[origin=c]{90}{$\subseteq$} \\
\#\mathrm{FO}(\subseteq)^{\mathrm{team}} & & & \#\mathrm{FO}(=(\dots))^{\mathrm{team}}
\end{array}$$

We also showed that our classes are closed under first-order reductions and that $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$ and $\#\mathrm{FO}(=(\dots))^{\mathrm{team}}$ contain complete problems from $\#\mathrm{P}$ and $\# \cdot \mathrm{NP}$, respectively. The latter problem is even complete for $\#\mathrm{FO}(=(\dots))^{\mathrm{team}}$ under first-order reductions.

The connection between $\#\mathrm{FO}(=(\dots))^{\mathrm{team}}$ and the classes $\#\mathrm{P}$ and $\# \cdot \mathrm{NP}$ is not yet clear. While we know that a complete problem from $\# \cdot \mathrm{NP}$ is contained in it, it is open whether the class coincides with $\# \cdot \mathrm{NP}$, and (if not) whether it contains the class $\#\mathrm{P}$. We conjecture that the answer to both questions is negative, since the defining logic has closure properties that make it unlikely to contain counting versions of non-monotone problems from $\#\mathrm{P}$.

Regarding $\#\mathrm{FO}(\subseteq)^{\mathrm{team}}$, the search for a complete problem could be interesting. We have only showed that the problem #DualHorn is contained in this class and the problem $\#\Sigma_1$DualHorn is hard for this class, but neither of the problems is known to be complete.

The lower end of our hierarchy deserves further study as well. The class $\#\mathrm{FO}^{\mathrm{team}}$ (i.e., the class with no dependency atoms in the formulae) can be shown to be a subclass of FTC^0, the class of functions computable by families of polynomial size constant depth majority circuits (see [35]). The circuit-based counting class $\#\mathrm{AC}^0$, counting proof trees in polynomial size constant depth unbounded fan-in circuits [35], was characterized in a model-theoretic manner by counting assignments to free function symbols in certain quantifier-restricted FO-formulae [7]. A similar quantifier restriction for $\#\mathrm{FO}(A)^{\mathrm{team}}$, where A consists of the dependency atom plus a totality atom (that we did not study in the present paper), also leads to a characterization of $\#\mathrm{AC}^0$. This suggests that low level counting classes and circuit classes in the context of counting problems for team-based logics might be worth studying. Another natural question is to search for generalized dependency atoms that lead to interesting relations to complexity classes. Besides the aforementioned totality atom, the constancy or the exclusion [12] atom are worth examining. In particular, it is an open question to find an atom A such that $\#\mathrm{FO}(A)^{\mathrm{team}} = \#\mathrm{P}$. The logic $\mathcal{L}$ of [25], though it satisfies the equality, is not of this form.

In the context of counting complexity theory, an interesting question to study is the approximability of problems in different classes. In our case, it is unlikely that any of our classes is efficiently approximable (in the sense of FPRAS): In [10] it was shown that it is unlikely that the number of satisfying assignments of CSPs can be approximated by an FPRAS, unless all relations in the constraint language are affine. Since our classes contain counting problems for classes of formulae which do not admit this property, this result applies. Consequently, it would be interesting to study restrictions of our full classes to obtain, possibly, efficiently approximable fragments.

Our proof of the completeness of $\#\Sigma_1\mathrm{CNF}^-$ for $\# \cdot \mathrm{NP}$ introduces problems that arise from "pairing decision problems" and gives simultaneous reductions between such pairs. This idea might be helpful in other contexts as well; in particular it should lead to more interesting complete problems for $\# \cdot \mathrm{NP}$ or higher levels $\# \cdot \Sigma_k$ of the counting polynomial-time hierarchy.

References

1 Marcelo Arenas, Martin Muñoz, and Cristian Riveros. Descriptive Complexity for counting complexity classes. In *LICS*, pages 1–12. IEEE Computer Society, 2017.

2 Rehan Abdul Aziz, Geoffrey Chu, Christian J. Muise, and Peter J. Stuckey. #∃SAT: Projected model counting. In *SAT*, volume 9340 of *Lecture Notes in Computer Science*, pages 121–137. Springer, 2015.

3 Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Solving #SAT and Bayesian Inference with Backtracking Search. *CoRR*, abs/1401.3458, 2014.

4 Ivano Ciardelli. Dependency as Question Entailment. In Samsom Abramsky, Juha Kontinen, Jouko Väänänen, and Heribert Vollmer, editors, *Dependence Logic: Theory and Application*, Progress in Computer Science and Applied Logic, pages 129–182. Birkhauser, 2016.

5 Jukka Corander, Antti Hyttinen, Juha Kontinen, Johan Pensar, and Jouko Väänänen. A Logical Approach to Context-Specific Independence. *Ann. Pure Appl. Logic*, 2019.

6 Arnaud Durand, Johannes Ebbing, Juha Kontinen, and Heribert Vollmer. Dependence Logic with a Majority Quantifier. *Journal of Logic, Language and Information*, 24(3):289–305, 2015. doi:10.1007/s10849-015-9218-3.

7 Arnaud Durand, Anselm Haak, Juha Kontinen, and Heribert Vollmer. Descriptive Complexity of #AC^0 Functions. In *CSL*, volume 62 of *LIPIcs*, pages 20:1–20:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

8 Arnaud Durand, Anselm Haak, and Heribert Vollmer. Model-Theoretic Characterization of Boolean and Arithmetic Circuit Classes of Small Depth. In *LICS*, pages 354–363. ACM, 2018.

9 Arnaud Durand, Juha Kontinen, Nicolas de Rugy-Altherre, and Jouko Väänänen. Tractability Frontier of Data Complexity in Team Semantics. In *GandALF*, volume 193 of *EPTCS*, pages 73–85, 2015.

10 Martin E. Dyer, Leslie Ann Goldberg, and Mark Jerrum. An approximation trichotomy for Boolean #CSP. *J. Comput. Syst. Sci.*, 76(3-4):267–277, 2010. doi:10.1016/j.jcss.2009.08.003.

11 Ronald Fagin. Generalized first-order spectra, and polynomial time recognizable sets. *SIAM-AMS Proceedings*, 7:43–73, 1974.

12 Pietro Galliani. Inclusion and exclusion dependencies in team semantics - On some logics of imperfect information. *Ann. Pure Appl. Logic*, 163(1):68–84, 2012.

13 Pietro Galliani and Lauri Hella. Inclusion Logic and Fixed Point Logic. In *CSL*, volume 23 of *LIPIcs*, pages 281–295. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

14 Erich Grädel and Jouko A. Väänänen. Dependence and Independence. *Studia Logica*, 101(2):399–410, 2013.

15 Miika Hannula, Åsa Hirvonen, Juha Kontinen, Vadim Kulikov, and Jonni Virtema. Facets of Distribution Identities in Probabilistic Team Semantics. *CoRR*, abs/1812.05873, 2018.

16 Miika Hannula and Juha Kontinen. A finite axiomatization of conditional independence and inclusion dependencies. *Inf. Comput.*, 249:121–137, 2016.

17 Lane A. Hemaspaandra and Heribert Vollmer. The satanic notations: counting classes beyond #P and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.

18 Tapani Hyttinen, Gianluca Paolini, and Jouko Väänänen. Quantum Team Logic and Bell's inequalities. *Rew. Symb. Logic*, 8(4):722–742, 2015. doi:10.1017/S1755020315000192.

19 Neil Immerman. Relational Queries Computable in Polynomial Time. *Information and Control*, 68(1-3):86–104, 1986.

20 Neil Immerman. *Descriptive Complexity.* Graduate texts in computer science. Springer, 1999.

21 Juha Kontinen. A logical characterization of the counting hierarchy. *ACM Trans. Comput. Log.*, 10(1):7:1–7:21, 2009.

22 Juha Kontinen, Antti Kuusisto, and Jonni Virtema. Decidability of Predicate Logics with Team Semantics. In *MFCS*, volume 58 of *LIPIcs*, pages 60:1–60:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

23 Juha Kontinen and Hannu Niemistö. Extensions of MSO and the monadic counting hierarchy. *Inf. Comput.*, 209(1):1–19, 2011.

24 Juha Kontinen and Jouko A. Väänänen. On Definability in Dependence Logic. *Journal of Logic, Language and Information*, 18(3):317–332, 2009.

25 Juha Kontinen and Fan Yang. Logics for first-order team properties. In *WoLLIC*, Lecture Notes in Computer Science. Springer, 2019.

26 Antti Kuusisto. A Double Team Semantics for Generalized Quantifiers. *Journal of Logic, Language and Information*, 24(2):149–191, 2015.

27 Antti Kuusisto and Carsten Lutz. Weighted model counting beyond two-variable logic. In *LICS*, pages 619–628. ACM, 2018.

28 Eric Pacuit and Fan Yang. Dependence and Independence in Social Choice: Arrow's Theorem. In H. Vollmer S. Abramsky, J. Kontinen and J. Väänänen, editors, *Dependence Logic: Theory and Application*, Progress in Computer Science and Applied Logic, pages 235–260. Birkhauser, 2016.

29 Aris Pagourtzis and Stathis Zachos. The Complexity of Counting Functions with Easy Decision Version. In *MFCS*, volume 4162 of *Lecture Notes in Computer Science*, pages 741–752. Springer, 2006.

30 Sanjeev Saluja, K. V. Subrahmanyam, and Madhukar N. Thakur. Descriptive Complexity of #P Functions. *J. Comput. Syst. Sci.*, 50(3):493–505, 1995.

31 Jouko A. Väänänen. *Dependence Logic - A New Approach to Independence Friendly Logic*, volume 70 of *London Mathematical Society student texts*. Cambridge University Press, 2007.

32 Leslie G. Valiant. The Complexity of Computing the Permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

33 Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

34 Moshe Y. Vardi. The Complexity of Relational Query Languages (Extended Abstract). In *STOC*, pages 137–146. ACM, 1982.

35 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999.

Approximating Activation Edge-Cover and Facility Location Problems

Zeev Nutov
The Open University of Israel, Ra'anana, Israel
nutov@openu.ac.il

Guy Kortsarz
Rutgers University, Camden, USA
guyk@camden.rutgers.edu

Eli Shalom[1]
The Open University of Israel, Ra'anana, Israel
eli.shalom@gmail.com

Abstract

What approximation ratio can we achieve for the Facility Location problem if whenever a client u connects to a facility v, the opening cost of v is at most θ times the service cost of u? We show that this and many other problems are a particular case of the Activation Edge-Cover problem. Here we are given a multigraph $G = (V, E)$, a set $R \subseteq V$ of terminals, and thresholds $\{t^e_u, t^e_v\}$ for each uv-edge $e \in E$. The goal is to find an assignment $\mathbf{a} = \{a_v : v \in V\}$ to the nodes minimizing $\sum_{v \in V} a_v$, such that the edge set $E_{\mathbf{a}} = \{e = uv : a_u \geq t^e_u, a_v \geq t^e_v\}$ activated by $\mathbf{a}$ covers R. We obtain ratio $1 + \max\limits_{x \geq 1} \dfrac{\ln x}{1 + x/\theta} \approx \ln\theta - \ln\ln\theta$ for the problem, where θ is a problem parameter. This result is based on a simple generic algorithm for the problem of minimizing a sum of a decreasing and a sub-additive set functions, which is of independent interest. As an application, we get the same ratio for the above variant of Facility Location. If for each facility all service costs are identical then we show a better ratio $1 + \max\limits_{k \in \mathbb{N}} \dfrac{H_k - 1}{1 + k/\theta}$, where $H_k = \sum_{i=1}^k 1/i$. For the Min-Power Edge-Cover problem we improve the ratio 1.406 of [4] (achieved by iterative randomized rounding) to 1.2785. For unit thresholds we improve the ratio $73/60 \approx 1.217$ of [4] to $\frac{1555}{1347} \approx 1.155$.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms

Keywords and phrases generalized min-covering problem, activation edge-cover, facility location, minimum power, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.20

Funding *Guy Kortsarz*: Partially supported by NSF grant number 1540547.

Acknowledgements We thank anonymous referees for many useful comments.

1 Introduction

Let $G = (V, E)$ be an undirected multigraph where each edge $e \in E$ has an **activating function** f^e from some range $L^e \subseteq \mathbb{R}^2_+$ to $\{0, 1\}$. Given a non-negative **assignment** $\mathbf{a} = \{a_v : v \in V\}$ to the nodes, we say that a uv-edge $e \in E$ is **activated** by $\mathbf{a}$ if $f^e(a_u, a_v) = 1$. Let $E_{\mathbf{a}} = \{e \in E : f^e(a_u, a_v) = 1\}$ denote the set of edges activated by $\mathbf{a}$. The **value** of an assignment $\mathbf{a}$ is $\mathbf{a}(V) = \sum_{v \in V} a_v$. In Activation Network Design problems the goal is to find an assignment $\mathbf{a}$ of minimum value, such that the edge set $E_{\mathbf{a}}$ activated by $\mathbf{a}$ satisfies a prescribed property. We make the following two assumptions, which are standard in the literature; see the paper of Panigrahi [20] that introduced the problem, and a recent survey [19] on various activation problems.

[1] Part of this work was done as a part of author's M.Sc. thesis done at the Open University of Israel.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 20; pp. 20:1–20:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Monotonicity Assumption. *For every $e \in E$, f^e is monotone non-decreasing, namely, $f^e(x_u, x_v) = 1$ implies $f^e(y_u, y_v) = 1$ if $y_u \geq x_u$ and $y_v \geq x_v$.*

Polynomial Domain Assumption. *Every $v \in V$ has a polynomial size in $n = |V|$ set L_v of "levels" and $L^e = L_u \times L_v$ for every uv-edge $e \in E$.*

Given a set $R \subseteq V$ of **terminals** we say that an edge set J is an **R-cover** or that J **covers** R if every $v \in R$ has some edge in J incident to it. In Edge-Cover problems we seek an R-cover J that minimizes a given value function, e.g., the edge cost of J. The min-cost Edge-Cover problem can be solved in polynomial time [9], and it is one of the most fundamental problems in Combinatorial Optimization, cf. [23].

We consider the Activation Edge-Cover problem. Since we consider multigraphs, $e = uv$ means that e is a uv-edge, namely, that u, v are the endnodes of e; $e = uv \in E$ means that e is a uv-edge that belongs to E. Under the two assumptions above, the problem can be formulated without activating functions. For this, replace each edge $e = uv$ by a set of at most $|L_u| \cdot |L_v|$ uv-edges $\{e(t_u, t_v) : (t_u, t_v) \in L_u \times L_v, f^e(t_u, t_v) = 1\}$. Then for any $J \subseteq E$ the optimal assignment $\mathbf{a}$ activating J is given by $a_u = \max\{t^e_u : e \in J \text{ is incident to } u\}$; here and everywhere a maximum or a minimum taken over an empty set is assumed to be zero. Consequently, the problem can be restated as follows.

Activation Edge-Cover
Input: A graph $G = (V, E)$, a set of terminals $R \subseteq V$, and thresholds $\{t^e_u, t^e_v\}$ for each uv-edge $e \in E$.
Output: An assignment $\mathbf{a}$ of minimum value $\mathbf{a}(V) = \sum_{v \in V} a_v$, such that the edge set $E_{\mathbf{a}} = \{e = uv \in E : a_u \geq t^e_u, a_v \geq t^e_v\}$ activated by $\mathbf{a}$ covers R.

As we will explain later, Activation Edge-Cover problems are among the most fundamental problems in network design, that include NP-hard problems such as Set-Cover, Facility Location, covering problems that arise in wireless networks (node weighted/min-power/installation problems), and many other problems. The Activation Edge-Cover problem admit ratio $O(\ln |R|)$ by a factor 2 reduction to the Facility Location problem.

To state our main result we define assignments $\mathbf{q}$ and $\mathbf{c}$, where $c_v = q_v = 0$ if $v \in V \setminus R$ and for $u \in R$:

- $q_u = \min\limits_{e=uv \in E} t^e_u$ is the minimum threshold at u of an edge in E incident to u.
- $c_u = \min\limits_{e=uv \in E} (t^e_u + t^e_v) - q_u$, so $c_u + q_u$ is the minimum value of an edge in E incident to u.

Following [19], the quantity $\max_{u \in R} c_u/q_u$ is called the **slope** of the instance. We say that an Activation Edge-Cover instance is **θ-bounded** if the instance slope is at most θ, namely if $c_u \leq \theta q_u$ for all $u \in R$; moreover, we assume by default that $\theta = \max_{u \in R} c_u/q_u$ is the instance slope. Let opt denote the optimal solution value of a problem instance at hand. For each $u \in R$ let e_u be some minimum value edge covering u. Then $\{e_u : u \in R\}$ is an R-cover of value at most $\sum_{u \in R}(c_u + q_u) = (\mathbf{c} + \mathbf{q})(R)$. From this and the definition of θ we get

$$0 \leq \mathsf{opt} - \mathbf{q}(R) \leq \mathbf{c}(R) \leq \theta \mathbf{q}(R) \leq \theta \mathsf{opt}$$

In particular, $(\mathbf{c} + \mathbf{q})(R) \leq (\theta + 1)\mathsf{opt}$. Using this, it is possible to design a greedy algorithm with ratio $1 + \ln(\theta + 1)$. We will show how to obtain a better ratio (the difference is quite significant when $\theta \leq 10^4$ – see Table 1). In what follows, let H_k denote the k-th harmonic number. Our approximation ratios are expressed in the terms of the following two functions $\omega(\theta)$ and $\bar{\omega}(\theta)$ defined for $\theta > 0$, where $\bar{\omega}$ can be viewed as a "discrete version" of ω:

$$\omega(\theta) = \max_{x \geq 1} \frac{\ln x}{1 + x/\theta} \qquad \bar{\omega}(\theta) = \max_{k \in \mathbb{N}} \frac{H_k - 1}{1 + k/\theta}$$

Table 1 Some numerical bounds on $1+\omega(\theta)$, $1+\bar{\omega}(\theta)$, $\ln\theta - \ln\ln\theta$, and $1+\ln(\theta+1)$.

θ	1	2	3	5	10	100	1000	10000	1000000
$1+\omega(\theta)$	1.2785	1.4631	1.6036	1.8146	2.1569	3.6360	5.4214	7.3603	11.4673
$1+\bar{\omega}(\theta)$	1.2167	1.3667	1.4834	1.6637	1.9645	3.3428	5.0808	6.9967	11.0820
$\ln\theta - \ln\ln\theta$	-	1.0597	1.0046	1.1336	1.4686	3.0780	4.9752	6.9901	11.1898
$1+\ln(\theta+1)$	1.6932	2.0987	2.3863	2.7918	3.3979	5.6152	7.9088	10.2105	14.8156

In Lemma 8 we show that $\omega(\theta) = W(\theta/e)$, where $W(z)$ is the **Lambert W Function** (a.k.a. ProductLog Function), which is the inverse function of $z(W) = We^W$, c.f. [7, 15]. We are not aware of any known formula for $\bar{\omega}(\theta)$, but since $H_k - 1 \leq \ln k$, $\bar{\omega}(\theta) < \omega(\theta)$ for $\theta > 0$. We also observe in Section 4 that in the definition of $\bar{\omega}(\theta)$ the maximum is attained for the smallest integer k such that $H_k \geq 2 + \frac{\theta-1}{k+1}$. It follows from [15] that $\lim_{\theta\to\infty}[1 + W(\theta/e) - (\ln\theta - \ln\ln\theta)] = 0$, so $1+\omega(\theta)$ (and it seems that also $1+\bar{\omega}(\theta)$) is close to $\ln\theta - \ln\ln\theta$ for large values of θ, although the convergence is very slow; see Table 1.

These functions were implicitly used before to bound approximation ratios. E.g., Robins & Zelikovsky [22] proved that their algorithm for the Steiner Tree problem in quasi-bipartite graph achieves ratio $1+\omega(1)+\epsilon < 1.2785$; this was improved to $1+\bar{\omega}(1)+\epsilon = \frac{73}{60}+\epsilon$ in [3]. See also [12] and a survey on the Steiner Tree problem in [13]. In [4] is established ratio $\frac{73}{60}$ for the Min-Power Edge-Cover problem in bipartite graphs and for unit thresholds.

Our main result is:

▶ **Theorem 1.** Activation Edge-Cover *admits ratio* $1+\omega(\theta)$ *for* θ*-bounded instances. The problem also admits ratio* $1+\ln(\Delta+1)$*, and ratio* $1+\ln\Delta$ *if* R *is an independent set in* G*, where* Δ *is the maximum number of terminal neighbors of a node in* G*.*

This result is based on a generic simple approximation algorithm for the problem of minimizing a sum of a decreasing and a sub-additive set functions, which is of independent interest; it is described in the next section. This result is inspired by the algorithm of Robins & Zelikovsky [22] for the Steiner Tree problem, and the analysis in [13] of this algorithm.

We note that Slavik [25] proved that the greedy algorithm for Set-Cover achieves ratio $\ln n - \ln\ln n + \Theta(1)$, while our ratio for Activation Edge-Cover is asymptotically $\ln\theta - \ln\ln\theta + \Theta(1)$; but we do not see an immediate relation between the two results.

Let us say that $v \in V$ is a **steady node** if the thresholds t_v^e of the edges e incident to v are all equal to the same number w_v, which we call the **weight of** v. Note that we may assume that all non-terminals are steady, by replacing each $v \in V \setminus R$ by L_v new nodes; see the so called "Levels Reduction" in [19]. This implies that no two parallel edges are incident to the same non-terminal. Clearly, we may assume that $V \setminus R$ is an independent set in G. Let Bipartite Activation Edge-Cover be the restriction of Activation Edge-Cover to instances when also R is an independent set, namely, when G is bipartite with sides $R, V \setminus R$. Note that in this case G is a simple graph and all non-terminals are steady.

We now mention some threshold types in Activation Edge-Cover problems, known problems arising from these types, and some implications of Theorem 1 for these problems.

Weighted Set-Cover.

This is a particular case of Bipartite Activation Edge-Cover when all nodes are steady and nodes in R have weight 0. Note that in this case θ is infinite, and we can only deduce from Theorem 1 the known ratio $1 + \ln \Delta$. Consider a modification of the problem, which we call θ-Bounded Weighted Set-Cover: when we pick a set $v \in V \setminus R$, we pay w_v for v, and also pay w_v/θ for each element in R covered by v that was not yet covered. Then the corresponding Activation Edge-Cover instance is θ-bounded.

Facility Location.

Here we have a bipartite graph with sides R (clients) and $V \setminus R$ (facilities), weights (opening costs) $\mathbf{w} = \{w_v : v \in V \setminus R\}$, and distances (service costs) $\mathbf{d} = \{d_{uv} : u \in R, v \in V \setminus R\}$. We need to choose $S \subseteq V \setminus R$ with $\mathbf{w}(S) + \sum_{u \in R} d(u, S)$ minimal, where $d(u, S) = \min_{v \in S} d_{uv}$ is the minimal distance from u to S. This is equivalent to Bipartite Activation Edge-Cover. Note however that if for some constant θ we have $w_v \leq \theta d_{uv}$ for all $uv \in E$ with $u \in R$ and $v \in V \setminus R$, then the corresponding Bipartite Activation Edge-Cover instance is θ-bounded, and achieves a low constant ratio even for large values of θ, by Theorem 1.

Installation Edge-Cover.

Suppose that the installation cost of a wireless network is proportional to the total height of the towers for mounting antennas. An edge uv is activated if the towers at u and v are tall enough to overcome obstructions and establish line of sight between the antennas. This is modeled as each pair $u, v \in V$ has a height demand h^{uv} and constants γ_{uv}, γ_{vu}, such that a uv-edge is activated by $\mathbf{a}$ if the scaled heights $\gamma_{uv} a_u, \gamma_{vu} a_v$ sum to at least h^{uv}. In the Installation Edge-Cover problem, we need to assign heights to the antennas such that each terminal can communicate with some other node, while minimizing the total sum of the heights. The problem is Set-Cover hard even for $0, 1$ thresholds and bipartite G [20]. But in a practical scenario, the quotient of the maximum tower height over the minimum tower height is a small constant; say, if possible tower heights are $5, 15, 20$, then $\theta = 4$.

Min-Power Edge-Cover.

This problem is a particular case of Activation Edge-Cover when $t_u^e = t_v^e$ for every edge $e = uv \in E$; note that $\theta = 1$ in this case (in fact, the case $\theta = 1$ is much more general). The motivation is to assign energy levels to the nodes of a wireless network while minimizing the total energy consumption, and enabling communication for every terminal. The Min-Power Edge-Cover problem is NP-hard even if $R = V$, or if R is an independent set in the input graph G and unit thresholds [14]. The problem admits ratio 2 by a trivial reduction to the min-cost case. This was improved to 1.5 in [17], and then to 1.406 in [4], which also establishes the ratio 73/60 for the bipartite case and for unit thresholds.

From Theorem 1 and the discussion above we get:

▶ **Corollary 2.** Min-Power Edge-Cover *admits ratio* $1 + \omega(1) < 1.2785$*, and the* θ*-bounded versions of each of the problems* Weighted Set-Cover, Facility Location*, and* Installation Edge-Cover*, admits ratio* $1 + \omega(\theta)$*.*

Let us illustrate this result on the Facility Location problem. One might expect a constant ratio for any $\theta > 0$, but our ratio $1+\omega(\theta)$ is surprisingly low. Even if $\theta = 100$ (service costs are at least 1% of opening costs) then we get a small ratio $1 + \omega(100) < 3.636$. Even for $\theta = 10^4$ we still get a reasonable ratio $1 + \omega(10^4) < 7.3603$. All previous results for the problem are usually summarized by just two observations: the problem is Set-Cover hard (so has a logarithmic approximation threshold by [21, 10]), and that it admits a matching logarithmic ratio $1 + \ln |R|$ [6]; see surveys on Facility Location problems by Vygen [26] and Shmoys [24]. Due to this, almost all work focused on the more tractable Metric Facility Location problem. Our Theorem 1 implies that many practical non-metric Facility Location instances admit a reasonable small constant ratio.

Note that our Theorem 1 ratio 1.2785 for $\theta = 1$ significantly improves the previous best ratio 1.406 of [4] for Min-Power Edge-Cover on general graphs achieved by iterative randomized rounding; we do not match the ratio 73/60 of [4] for the bipartite case, but note that the case $\theta = 1$ is much more general than the min-power case considered in [4].

For the case of "locally uniform" thresholds – when for each non-terminal (facility) all thresholds (service costs) are identical, we show a better ratio, see also Table 1.

▶ **Theorem 3.** Bipartite Activation Edge-Cover *with locally uniform thresholds admits ratio* $1 + \bar{\omega}(\theta)$.

In addition, we consider unit thresholds, and using some new ideas in addition to [4], improve the "natural" previous best ratio $73/60 \approx 1.217$ of [4] as follows.

▶ **Theorem 4.** Activation Edge-Cover *with unit thresholds admits ratio* $\frac{1555}{1347} < 1.155$.

We note that the proofs of some of our results are non-trivial, although we invested an effort to simplify matters. In any case, the focus in this paper is not technical, but rather conceptual. Our main contribution is giving a unified algorithm for a large class of problems that we identify – θ-Bounded Activation Edge-Cover problems, either substantially improving known ratios, or showing that many seemingly Set-Cover hard problems may be tractable in practice. Let us also point out that our main result is more general than the applications listed in Corollary 2. The generalization to θ-bounded Activation Edge-Cover problems is different from earlier results; besides finding a unifying algorithmic idea that generalizes and improves previous results, we are also able to find tractable special cases in a new direction.

The rest of this paper is organized as follows. In Section 2 we define the Generalized Min-Covering problem and analyze a greedy algorithm for it, see Theorem 5. In Section 3 we use Theorem 5 to prove Theorem 1. Theorems 3 and 4 are proved using a modified method in Sections 4 and 5, respectively.

2 The Generalized Min-Covering problem

A set function f is **increasing** if $f(A) \leq f(B)$ whenever $A \subseteq B$; f is **decreasing** if $-f$ is increasing, and f is **sub-additive** if $f(A \cup B) \leq f(A) + f(B)$ for any subsets A, B of the ground-set. Let us consider the following algorithmic problem:

Generalized Min-Covering
Input: Non-negative set functions ν, τ on subsets of a ground-set U such that ν is decreasing, τ is sub-additive, and $\tau(\emptyset) = 0$.
Output: $A \subseteq U$ such that $\nu(A) + \tau(A)$ is minimal.

The "ordinary" Min-Covering problem is $\min\{\tau(A) : \nu(A) = 0\}$; it is a particular case of the Generalized Min-Covering problem when we seek to minimize $M\nu(A) + \tau(A)$ for a large enough constant M. Under certain assumptions, the Min-Covering problem admits ratio $1 + \ln \nu(\emptyset)$ [16]. Various generic covering problems are considered in the literature, among them the Submodular Covering problem [27], and several other types, cf. [5].

The variant we consider is inspired by the algorithms of Robins & Zelikovsky [22] for the Steiner Tree problem, and the analysis in [13] of this algorithm; but, to the best of our knowledge, the explicit formulation of the Generalized Min-Covering problem given here is new. Interestingly, our ratio for Activation Edge-Cover with $\theta = 1$ is the same as that of [22] for Steiner Tree in quasi-bipartite graphs.

We call ν the **potential** and τ the **payment**. The idea behind this interpretation and the subsequent greedy algorithm is as follows. Given an optimization problem, the potential $\nu(A)$ is the value of some "simple" augmenting feasible solution for A. We start with an empty set solution, and iteratively try to decrease the potential by adding a set $B \subseteq U \setminus A$ of minimum "density" – the price paid for a unit of the potential. The algorithm terminates when the price ≥ 1, since then we gain nothing from adding B to A. The ratio of such an algorithm is bounded by $1 + \ln \frac{\nu(\emptyset)}{\mathsf{opt}}$ (assuming that during each iteration a minimum density set can be found in polynomial time). So essentially the greedy algorithm converts ratio $\alpha = \frac{\nu(\emptyset)}{\mathsf{opt}}$ into ratio $1 + \ln \alpha$. However, sometimes a tricky definition of the potential and the payment functions may lead to a smaller ratio.

Let opt be the optimal solution value of a problem instance at hand. Fix an optimal solution A^*. Let $\nu^* = \nu(A^*)$, $\tau^* = \tau(A^*)$, so $\mathsf{opt} = \tau^* + \nu^*$. The quantity $\frac{\tau(B)}{\nu(A)-\nu(A\cup B)}$ is called the **density** of B (w.r.t. A); this is the price paid by B for a unit of potential. The Greedy Algorithm (a.k.a. Relative Greedy Heuristic) for the problem starts with $A = \emptyset$ and while $\nu(A) > \nu^*$ repeatedly adds to A a non-empty augmenting set $B \subseteq U$ that satisfies the following condition, while such B exists:

Density Condition: $$\frac{\tau(B)}{\nu(A) - \nu(A \cup B)} \leq \min\left\{1, \frac{\tau^*}{\nu(A) - \nu^*}\right\}.$$

Note that since ν is decreasing $\nu(A) - \nu(A \cup A^*) \geq \nu(A) - \nu(A^*) = \nu(A) - \nu^*$; hence if $\nu(A) > \nu^*$, then $\frac{\tau(A^*)}{\nu(A)-\nu(A\cup A^*)} \leq \frac{\tau^*}{\nu(A)-\nu^*}$ and there exists an augmenting set B that satisfies the condition $\frac{\tau(B)}{\nu(A)-\nu(A\cup B)} \leq \frac{\tau^*}{\nu(A)-\nu^*}$, e.g., $B = A^*$. Thus if B^* is a minimum density set and $\frac{\tau(B^*)}{\nu(A)-\nu(A\cup B^*)} \leq 1$, then B^* satisfies the Density Condition; otherwise, the density of B^* is larger than 1 so no set can satisfy the Density Condition.

▶ **Theorem 5.** *The* Greedy Algorithm *achieves approximation ratio*

$$1 + \frac{\tau^*}{\mathsf{opt}} \ln \frac{\nu_0 - \nu^*}{\tau^*} = 1 + \frac{\tau^*}{\mathsf{opt}} \cdot \ln\left(1 + \frac{\nu_0 - \mathsf{opt}}{\tau^*}\right) .$$

Proof. Let ℓ be the number of iterations. Let $A_0 = \emptyset$ and for $i = 1, \ldots, \ell$ let A_i be the intermediate solution at the end of iteration i and $B_i = A_i \setminus A_{i-1}$. Let $\nu_i = \nu(A_i)$, $i = 0, \ldots, \ell$. Then:

$$\frac{\tau(B_i)}{\nu_{i-1} - \nu_i} \leq \min\left\{1, \frac{\tau^*}{\nu_{i-1} - \nu^*}\right\} \qquad i = 1, \ldots, \ell$$

Since ν is decreasing

$$\sum_{i=1}^{\ell} \tau(B_i) \leq \sum_{i=1}^{\ell} \min\left\{1, \frac{\tau^*}{\nu_{i-1} - \nu^*}\right\} (\nu_{i-1} - \nu_i)$$

This is the lower Darboux sum of the function $f(\nu) = \begin{cases} 1 & \text{if } \nu \leq \tau^* + \nu^* \\ \frac{\tau^*}{\nu - \nu^*} & \text{if } \nu > \tau^* + \nu^* \end{cases}$ in the interval $[\nu_\ell, \nu_0]$ w.r.t. the partition $\nu_\ell < \nu_{\ell-1} < \cdots < \nu_0$. We claim that $\tau^* + \nu^* \geq \nu_\ell$. Since the algorithm stopped with A_ℓ, at least one of the following holds: (i) $A_\ell = U$; (ii) $\nu_\ell \leq \nu^*$; (iii) A^* has density > 1. If (i) holds then (ii) holds (by the monotonicity of ν), but case (ii) is trivial. So assume that only (iii) holds. Then $\frac{\tau(A^*)}{\nu(A_\ell)-\nu(A_\ell \cup A^*)} > 1$, thus since ν is decreasing $\tau(A^*) \geq \nu_\ell - \nu(A \cup A^*) \geq \nu_\ell - \nu(A^*)$. Consequently, $\sum_{i=1}^{\ell} \tau(B_i)$ is bounded by

$$\int_{\nu_\ell}^{\nu_0} f(\nu)d\nu = \int_{\nu_\ell}^{\tau^*+\nu^*} 1d\nu + \int_{\tau^*+\nu^*}^{\nu_0} \frac{\tau^*}{\nu - \nu^*}d\nu = \tau^* + \nu^* - \nu_\ell + \tau^* \ln \frac{\nu_0 - \nu^*}{\tau^*}$$

Let $A = A_\ell = \bigcup_{i=1}^{\ell} B_i$ be the set computed by the algorithm. Since τ is sub-additive

$$\tau(A) \leq \sum_{i=1}^{\ell} \tau(B_i) \leq \tau^* + \nu^* - \nu(A) + \tau^* \ln \frac{\nu_0 - \nu^*}{\tau^*}$$

Thus the approximation ratio is bounded by $\frac{\tau(A)+\nu(A)}{\mathsf{opt}} \leq 1 + \frac{\tau^*}{\mathsf{opt}} \ln \frac{\nu_0 - \nu^*}{\tau^*}$. ◀

3 Algorithm for general thresholds (Theorem 1)

Given an instance $G = (V, E), \mathbf{t}, R$ of Activation Edge-Cover define the corresponding Generalized Min-Covering instance U, τ, ν as follows. We put at each node $u \in V$ a large set of "assignment units", and let U be the union of these sets of "assignment units". Note that to every $A \subseteq U$ naturally corresponds the assignment $\mathbf{a}$ where a_u is the number of units in A put at u. It would be more convenient to define ν and τ in terms of assignments, by considering instead of a set $A \subseteq U$ the corresponding assignment $\mathbf{a}$.

To define ν and τ, let us recall the assignments $\mathbf{q}$ and $\mathbf{c}$ from the Introduction. We have $c_v = q_v = 0$ if $v \in V \setminus R$ and for $u \in R$:

- $q_u = \min_{e=uv \in E} t_u^e$ is the minimum threshold at u of an edge in E incident to u.
- $c_u = \min_{e=uv \in E} (t_u^e + t_v^e) - q_u$, so $c_u + q_u$ is the minimum value of an edge in E incident to u.

We let $Q = \mathbf{q}(V) = \mathbf{q}(R)$ and $C = \mathbf{c}(R)$. Note that $\mathbf{c}(R') \leq \theta\mathbf{q}(R')$ for any $R' \subseteq R$; in particular, $C \leq \theta Q$. For an assignment $\mathbf{a}$ that "augments" $\mathbf{q}$ let $R_{\mathbf{q+a}}$ denote the set of terminals covered by the edge set $E_{\mathbf{q+a}}$ activated by the assignment $\mathbf{q}+\mathbf{a}$. A natural definition of the potential and the payment functions would be $\tau(\mathbf{a}) = \mathbf{a}(V)$ and $\nu(\mathbf{a}) = (\mathbf{c}+\mathbf{q})(R \setminus R_{\mathbf{q+a}})$ but this enable us to prove only ratio $1 + \ln(\theta + 1)$. We show a better ratio by adding to the potential in advance the "fixed" part Q. We define

$$\tau(\mathbf{a}) = \mathbf{a}(V) \qquad \nu(\mathbf{a}) = Q + \mathbf{c}(R \setminus R_{\mathbf{q+a}})$$

It is easy to see that ν is decreasing, τ is sub-additive, and $\tau(\mathbf{0}) = 0$.

The next lemma shows that the obtained Generalized Min-Covering instance is equivalent to the original Activation Edge-Cover instance.

▶ **Lemma 6.** *If $\mathbf{q}+\boldsymbol{a}$ is a feasible solution for* Activation Edge-Cover *then $\tau(\boldsymbol{a})+\nu(\boldsymbol{a}) = Q+\boldsymbol{a}(V)$. If $\mathbf{a}$ is a feasible solution for* Generalized Min-Covering *then one can construct in polynomial time a feasible solution for* Activation Edge-Cover *of value at most $\tau(\boldsymbol{a}) + \nu(\boldsymbol{a})$. In particular, both problems have the same optimal value, and* Generalized Min-Covering *has an optimal solution $\boldsymbol{a}^*$ such that $\nu(\boldsymbol{a}^*) = Q$ and thus $\mathsf{opt} = \tau(\boldsymbol{a}^*) + Q$.*

Proof. If $\mathbf{q} + \mathbf{a}$ is a feasible Activation Edge-Cover solution then $R_{\mathbf{q+a}} = R$ and thus $\nu(\mathbf{a}) = Q$. Consequently, $\tau(\mathbf{a}) + \nu(\mathbf{a}) = \mathbf{a}(V) + Q$.

Let now $\mathbf{a}$ be a Generalized Min-Covering solution. The assignment $\mathbf{q} + \mathbf{a}$ has value $Q + \mathbf{a}(V)$ and activates the edge set $E_{\mathbf{q+a}}$ that covers $R_{\mathbf{q+a}}$. To cover $R \setminus R_{\mathbf{q+a}}$, pick for every $u \in R \setminus R_{\mathbf{q}+\mathbf{a}}$ an edge uv with $t_u^{uv} + t_v^{uv}$ minimum. Let $\mathbf{b}$ be an assignment defined by $b_u = c_u$ if $u \in R \setminus R_{\mathbf{q}+\mathbf{a}}$ and $b_u = 0$ otherwise. The set of picked edges can be activated by an assignment $\mathbf{q}+\mathbf{b}$ that has value $Q + \mathbf{c}(R \setminus R_{\mathbf{q}+\mathbf{a}})$. The assignment $\mathbf{q}+\mathbf{a}+\mathbf{b}$ activates both edge sets and has value $Q + \mathbf{a}(V) + \mathbf{c}(R \setminus R_{\mathbf{q}+\mathbf{a}}) = \tau(\mathbf{a}) + \nu(\mathbf{a})$, as required. ◀

For the obtained Generalized Min-Covering instance, let us fix an optimal solution $\mathbf{a}^*$ as in Lemma 6, so $\nu^* = Q$ and $\mathsf{opt} = \tau^* + Q$. Denote $\nu_0 = \nu(\mathbf{0}) = Q + \mathbf{c}(R)$, and note that $\mathbf{c}(R) \le \theta Q$. To apply Theorem 5 we need several bounds given in the next lemma.

▶ **Lemma 7.** $\dfrac{\mathsf{opt}}{\tau^*} \ge 1 + \dfrac{1}{\theta}$, $\dfrac{\nu_0}{\tau^*} \le (\theta+1)\left(\dfrac{\mathsf{opt}}{\tau^*} - 1\right)$, *and* $\dfrac{\nu_0 - \nu^*}{\tau^*} \le \Delta + 1$.

Proof. Note that

$$\tau^* + Q = \mathsf{opt} \le \nu_0 \le (\theta+1)Q \ .$$

In particular, $\tau^* \le \theta Q$, and this implies the first bound of the lemma

$$\frac{\mathsf{opt}}{\tau^*} = 1 + \frac{Q}{\tau^*} \ge 1 + \frac{1}{\theta} \ .$$

The second bound of the lemma holds since $\nu_0 \le (\theta+1)Q = (\theta+1)(\mathsf{opt} - \tau^*)$.

The last bound of the lemma is equivalent to the bound $\mathbf{c}(R) \le \tau^*(\Delta+1)$. Let J be an inclusion minimal edge cover of R activated by $\mathbf{q}+\mathbf{a}^*$. Then J is a collection $\mathcal{S}$ of node disjoint rooted stars with leaves in R. Let $S \in \mathcal{S}$. By the definition of $\mathbf{c}$, $\mathbf{a}^*(S) \ge \max\limits_{u \in R \cap S} c_u$, thus $\mathbf{c}(R \cap S) \le |R \cap S|\mathbf{a}^*(S) \le (\Delta+1)\mathbf{a}^*(S)$. Consequently, $\mathbf{c}(R) = \sum\limits_{S \in \mathcal{S}} \mathbf{c}(R \cap S) \le (\Delta+1)\sum\limits_{S \in \mathcal{S}} \mathbf{a}^*(S) \le (\Delta+1)\mathbf{a}^*(V)$. ◀

We will show later that the Greedy Algorithm can be implemented to run in polynomial time; now we focus on showing that it achieves the approximation ratios stated in Theorem 1. Denote $x = \theta\left(\frac{\mathsf{opt}}{\tau^*} - 1\right)$ and $f(x) = \frac{\ln x}{1 + x/\theta}$. By Lemma 7 first bound, $x \ge 1$. Substituting Lemma 7 second bound in Theorem 5 second bound we get that the ratio is bounded by

$$1 + \frac{\tau^*}{\mathsf{opt}} \cdot \ln\left(1 + \frac{\nu_0}{\tau^*} - \frac{\mathsf{opt}}{\tau^*}\right) \le 1 + \frac{\ln x}{1 + x/\theta} = 1 + f(x)$$

Consequently, the ratio is bounded by $1 + \max\{f(x) : x \ge 1\} = 1 + \omega(\theta)$.

Substituting Lemma 7 third bound in Theorem 5 first bound and observing that $\tau^* \le \mathsf{opt}$ we get that the ratio is bounded by $1 + \ln(\Delta+1)$. In the case when R is an independent set in G, it is easy to see that the third bound in Lemma 7 improves to $\dfrac{\nu_0 - \nu^*}{\tau^*} \le \Delta$, and we get ratio $1 + \ln \Delta$ in this case.

In the next lemma we show that $\omega(\theta) = W(\theta/e)$, where $W(z)$ is the Lambert W Function, which is the inverse function of $z(W) = We^W$.

▶ **Lemma 8.** *For any $\theta > 0$, the equation $y + 1 = \ln(\theta/y)$ has a unique (real) root $y(\theta)$, and $0 < y(\theta) \le \theta$. Furthermore, $\omega(\theta) = y(\theta) = W(\theta/e)$ for any $\theta > 0$.*

Proof. Since the function $y+1$ is strictly increasing and the function $\ln(\theta/y)$ is strictly decreasing, the equation has at most one root; we claim that this root exists and is in the interval $(0, \theta]$. To see this consider the function $h(y) = y + 1 - \ln(\theta/y)$, and note that h is continuous and that $h(\theta) = \theta + 1 > 0$ while $h(\epsilon) = \epsilon + 1 - \ln(\theta/\epsilon) < 0$ for $\epsilon > 0$ small enough.

Let $f(x)=\frac{\ln x}{1+x/\theta}$. Since $f(1)=0$ and $\lim\limits_{x\to\infty} f(x)=0$ (by L'Hospital's Rule), $f(x)$ attains a maximum when

$$f'(x)=\frac{(1+x/\theta)/x-(\ln x)/\theta}{(1+x/\theta)^2}=\frac{1+x/\theta-(x/\theta)\ln x}{x(1+x/\theta)^2}=0\ .$$

So $f'(x)=0$ if and only if $1+x/\theta=(x/\theta)\ln x$, and then $f(x)=\theta/x$. Substituting $y\leftarrow\theta/x$ we get the equation $y+1=\ln(\theta/y)$, where $0<y\leq\theta$. Thus $\omega(\theta)=y(\theta)$.

To prove that $W(\theta/e)=y(\theta)$ we show that $W(\theta/e)+1=\ln(\theta/W(\theta/e))$, namely, that $W(\theta/e)$ is the root of the equation $y+1=\ln(\theta/y)$ that defines $y(\theta)$. We have $W(z)e^{W(z)}=z$ for any $z>0$ (c.f. Eq. 1.5 in [7]), which is equivalent to $W(z)+\ln W(z)=\ln z$. Thus $W(\theta/e)+\ln W(\theta/e)=\ln(\theta/e)$, which gives $W(\theta/e)+1=\ln(\theta/W(\theta/e))$, as claimed. ◀

Finally, we show that the Greedy Algorithm algorithm can be implemented in polynomial time. As was mentioned in Section 2 before Theorem 5, we just need to perform in polynomial time the following two operations for any assignment $\mathbf{a}$: to check the condition $\nu(\mathbf{a})>\nu^*$, and to find an augmenting assignment $\mathbf{b}$ of minimum density.

The assignments $\mathbf{q}$ and $\mathbf{c}$ can be computed in polynomial time, and thus the potential $\nu(\mathbf{a})=Q+\mathbf{c}(R\setminus R_{\mathbf{q}+\mathbf{a}})$ can be computed in polynomial time, for any $\mathbf{a}$. Let $\mathbf{a}^*$ be an optimal solution as in Lemma 6, and denote $\tau^*=\tau(\mathbf{a}^*)$ and $\nu^*=\nu(\mathbf{a}^*)=Q$. Then the condition $\nu(A)>\nu^*$ is equivalent to $\nu(\mathbf{a})\geq Q$ and thus can be checked in polynomial time.

Now we show how to find an augmenting assignment $\mathbf{b}$ of minimum density. Note that the density of an assignment $\mathbf{b}$ w.r.t. $\mathbf{a}$ is

$$\frac{\tau(\mathbf{b})}{\nu(\mathbf{a})-\nu(\mathbf{a}+\mathbf{b})}=\frac{\mathbf{b}(V)}{\mathbf{c}(R\setminus R_{\mathbf{q}+\mathbf{a}})-\mathbf{c}(R\setminus R_{\mathbf{q}+\mathbf{a}+\mathbf{b}})}=\frac{\mathbf{b}(V)}{\mathbf{c}(R_{\mathbf{q}+\mathbf{a}+\mathbf{b}}\setminus R_{\mathbf{q}+\mathbf{a}})}\ .$$

▶ **Lemma 9.** *There exists a polynomial time algorithm that given an instance of* Activation Edge-Cover *and an assignment* $\mathbf{a}$ *finds an assignment* $\mathbf{b}$ *of minimum density.*

Proof. A **star** is a rooted tree $S=(V_S,E_S)$ with at least one edge such that only its root s may have degree ≥ 2. We say that a star S is a **proper star** if all the leaves of S are terminals. We denote the terminals in S by $R_S=R\cap V_S$.

Since $\mathbf{q},\mathbf{a}$ are given assignments, we simplify the notation by assuming that $R\leftarrow R\setminus R_{\mathbf{q}+\mathbf{a}}$ is our set of terminals, and that $\mathbf{a}\leftarrow\mathbf{q}+\mathbf{a}$ is our given assignment. Then the density of $\mathbf{b}$ is just $\frac{\mathbf{b}(V)}{\mathbf{c}(R_{\mathbf{a}+\mathbf{b}})}$. Let $\mathbf{b}^*$ be an assignment of minimum density, and let $J^*\subseteq E_{\mathbf{a}+\mathbf{b}^*}$ be an inclusion minimal $R_{\mathbf{a}+\mathbf{b}^*}$-cover. Then J^* decomposes into a collection $\mathcal{S}$ of node disjoint proper stars that collectively cover $R_{\mathbf{a}+\mathbf{b}^*}$. For $S\in\mathcal{S}$ let $\mathbf{b}^S$ be the optimal assignment such that $\mathbf{a}+\mathbf{b}^S$ activates S. Since the stars in $\mathcal{S}$ are node disjoint

$$\sum_{S\in\mathcal{S}}\mathbf{b}^S(V)\leq\mathbf{b}^*(V)\quad\text{and}\quad\sum_{S\in\mathcal{S}}\mathbf{c}(R_S)=\mathbf{c}(R_{\mathbf{a}+\mathbf{b}^*})\ .$$

By an averaging argument, $\frac{\mathbf{b}^S(V)}{\mathbf{c}(R_S)}\leq\frac{\mathbf{b}^*(V)}{\mathbf{c}(R_{\mathbf{a}+\mathbf{b}^*})}$ holds for some $S\in\mathcal{S}$, and since $\mathbf{b}^*$ is a minimum density assignment, so is $\mathbf{b}^S$, and $\frac{\mathbf{b}^S(V)}{\mathbf{c}(R_S)}=\frac{\mathbf{b}^*(V)}{\mathbf{c}(R_{\mathbf{a}+\mathbf{b}^*})}$ holds. Consequently, it is sufficient to show how to find in polynomial time an assignment $\mathbf{b}$ such that $\mathbf{a}+\mathbf{b}$ activates a proper star S and $\frac{\mathbf{b}(V)}{\mathbf{c}(R_S)}$ is minimal.

We may assume that we know the root v and the value $w=b_v$ of an optimal density pair $S,\mathbf{b}$; there are at most $|V||E|$ choices and we can try all and return the best outcome. Let $R_w=\{u\in R:\text{there is a }uv\text{-edge }e\text{ with }t_v^e\leq a_v+w\}$. For $u\in R_w$ let b_u be the minimal non-negative number for which there is a uv-edge e with $a_v+w\geq t_v^e$ and $a_u+b_u\geq t_u^e$.

Then our problem is equivalent to finding $R_S \subseteq R_w$ with $\sigma(R_S) = \frac{w+\mathbf{b}(R_S)}{\mathbf{c}(R_S)}$ minimum. This problem can be solved in polynomial time, by starting with $R_S = \emptyset$ and while there is $u \in R_w \setminus R_S$ with $\sigma(R_S + u) < \sigma(R_S)$, adding $u \in R_w \setminus R_S$ to R_S with b_u/c_u minimum. ◀

The proof of Theorem 1 is complete.

4 Locally uniform thresholds (Theorem 3)

Here we consider the Bipartite Activation Edge-Cover problem with locally uniform thresholds. This means that each non-terminal $v \in V \setminus R$ has weight w_v and all edges incident to v have the same threshold t^v; in the θ-bounded version $w_v \leq \theta t^v$. We consider a natural greedy algorithm that repeatedly picks a star S that minimizes the average price paid for each terminal (the quotient of the optimal activation value of S over $|R_S|$), and then removes R_S. Each time we choose a star S we distribute its activation value uniformly among its terminals, paying in the computed solution the average price for each terminal of S.

We now apply a standard "set-cover" analysis, cf. [28]. In some optimal solution fix an inclusion maximal star S^* with center v and terminals R_{S^*} covered by the algorithm in the order $r_k, r_{k-1}, ..., r_1$, where r_k is covered first and r_1 last; we bound the algorithm payment for covering R_{S^*}. Note that $1 \leq k \leq \Delta$. Denote $w = w_v$ and let t be the threshold of the terminals in S^*. Let S^*_i be the substar of S^* with leaves $r_i, \ldots, r_1$. At the start of the iteration in which the algorithm covers r_i, the terminals of S^*_i are uncovered. Thus the algorithm pays for covering r_i at most the average price paid by S^*_i, namely $(w+it)/i = w/i + t$. Over all iterations, the algorithm pays for covering R_{S^*} at most $wH_k + kt$, while the optimum pays $w + kt$. Thus the quotient between them is bounded by

$$\frac{wH_k + kt}{w + kt} = \frac{w/tH_k + k}{w/t + k} \leq \frac{\theta H_k + k}{\theta + k} = 1 + \frac{\theta(H_k - 1)}{\theta + k} \leq 1 + \max_{1 \leq k \leq \Delta} \frac{H_k - 1}{1 + k/\theta} \leq 1 + \bar{\omega}(\theta) .$$

Since any optimal solution decomposes into node disjoint stars, the last term bounds the approximation ratio, concluding the proof of Theorem 3.

Let $g(k) = \frac{H_k - 1}{1 + k/\theta}$ where $k \in \mathbb{N}$, so $\bar{\omega}(\theta) = \max_{k \in \mathbb{N}} g(k)$. We have

$$g(k+1) - g(k) = \frac{\theta}{(k+\theta)(k+1+\theta)} \left(2 - H_k + \frac{\theta - 1}{k+1} \right) .$$

Thus $g(k+1) \geq g(k)$ if and only if $2 - H_k + \frac{\theta-1}{k+1} > 0$. Consequently, $\bar{\omega}(\theta) = g(k(\theta))$ where $k(\theta)$ be the smallest integer k such that $H_k \geq 2 + \frac{\theta-1}{k+1}$.

For $\theta = 1$ we have $k(\theta) = 4$, so $1 + \bar{\omega}(1) = 73/60$. The greedy algorithm cannot do better even for unit thresholds, as shows the example in Fig. 1. The instance has 48 terminals (in black), and two sets of covering nodes: the upper 12 nodes that form an optimal cover, and the bottom 13 nodes. The bottom nodes have 3 nodes of degree 4, 4 of degree 3, and 6 of degree 2. The algorithm may start taking all bottom nodes, and only then add the upper ones, thus creating a solution of value 73, instead of the optimum 60.

5 Unit thresholds (Theorem 4)

Here we consider the case of unit thresholds when $t^e_u = t^e_v = 1$ for every uv-edge e. By a reduction from [4], we may assume that the instance is bipartite. Specifically, for any optimal assignment $\mathbf{a}$ we have $a_u = 1$ for all $u \in R$, hence we can consider the residual instance obtained by removing the terminals covered by edges with both ends in R; in the new obtained instance R is an independent set, and recall that we may assume that $V \setminus R$ is an independent set.

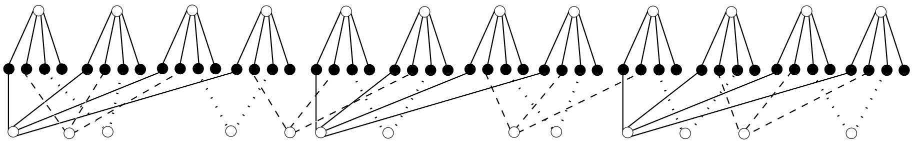


Figure 1 Tight example of ratio $\frac{73}{60}$ for unit thresholds.

One can observe that in the obtained bipartite instance, **a** is an optimal solution if and only if $a_v \in \{0,1\}$ for all $v \in V$, $a_v = 1$ for all $v \in R$, and the set $C = \{v \in V \setminus R : a_v = 1\}$ covers R, meaning that R is the set of neighbors of C. Namely, our problem is equivalent to $\min\{|C| + |R| : C \subseteq V \setminus R, C \text{ covers } R\}$. On the other hand the problem $\min\{|C| : C \subseteq V \setminus R, C \text{ covers } R\}$ is essentially the (unweighted) Set-Cover problem, and C is a feasible solution to this Set-Cover instance if and only if $C \cup R$ is the characteristic set of a feasible assignment for the Activation Edge-Cover instance. Note that both problems are equivalent w.r.t. their optimal solutions but may differ w.r.t. approximation ratios, since if C^* is an optimal Set-Cover solution then $\frac{|C|+|R|}{|C^*|+|R|}$ may be much smaller than $\frac{|C|}{|C^*|}$.

Recall that a standard greedy algorithm for Set-Cover repeatedly picks the center of a largest star and removes the star from the graph. This algorithm has ratio H_k for k-Set-Cover, where $k = \Delta$ is the maximum degree of a non-terminal (the maximum size of a set). However, the same algorithm achieves a much smaller ratio $\frac{73}{60}$ for Activation Edge-Cover with unit thresholds; the ratio $\frac{73}{60}$ was established in [4], and it also follows from the case $\theta = 1$ in Theorem 3. In what follows we denote by α_k the best known ratio for k-Set-Cover. We have $\alpha_1 = \alpha_2 = 1$ ($k = 2$ is the Edge-Cover problem) and $\alpha_3 = 4/3$ [8]. The current best ratios for $k \geq 4$ are due to [11] (see also [18, 1]). We summarize the current values of α_k for $k \leq 7$ in the following table.

Table 2 Current values of α_k for $k \leq 7$.

α_1	α_2	α_3	α_4	α_5	α_6	α_7
1	1	$\frac{4}{3}$	$\frac{73}{48}$	$\frac{26}{15}$	$\frac{28}{15}$	$\frac{212}{105}$

We now show how these ratios for k-Set-Cover can be used to approximate the Activation Edge-Cover problem with unit costs. We start by describing a simple algorithm with ratio $1\frac{67}{360} < \frac{73}{60}$, that uses only the $k = 2$ case.

Algorithm 1 ratio $1\frac{67}{360}$

```
$A \leftarrow \emptyset$
while there exists a star with at least 3 terminals do
    add to $A$ and remove from $G$ the node-set of a maximum size star
add to $A$ an optimal solution of the residual instance
```

We claim that the above algorithm achieves approximation ratio $1\frac{67}{360}$ for Activation Edge-Cover (a similar analysis implies ratio $H_k - \frac{1}{6}$ for Set-Cover). In some optimal solution fix a star S^* with terminals covered in the order $r_k, r_{k-1}, ..., r_1$, where r_k is covered first and r_1 last; we bound the algorithm payment to cover these terminals. Let S^*_i be the substar of S^* with leaves $r_i, \ldots, r_1$. At the start of the iteration when r_i is covered, the terminals of S^*_i are uncovered. Thus the algorithm pays for covering r_i at most the density of S^*_i, namely, $(i+1)i = 1 + 1/i$. Over all iterations, the algorithm pays for covering R_S at most $k + H_k$, while the optimum pays $k + 1$. If $k = 1$ then the algorithm pays at most the amount of the optimum. We claim that if $k \geq 2$ then in fact the payment is at most $k + H_k - 1/6$. If $k = 2$ then the payment is at most $3 < 3 + H_3 - 1/6$ (we pay 3 if the star "survives" all the iterations before the last). For $k \geq 3$, the pay for the last 3 terminals is either: $4/3$ for each of for r_3, r_2 and 2 for r_1 (a total of $14/3$), or $4/3$ for r_3 and 3 for r_2, r_1 (a total of $13/3$). The maximum is $14/3 = 3 + H_3 - 1/6$. Consequently, the ratio is bounded by

$$\max_{k \geq 2} \frac{k + H_k - 1/6}{k+1} = 1 + \max_{k \geq 2} \frac{H_k - 7/6}{k+1} = 1 + \max_{k \geq 2} g(k)$$

By fundamental computations we have $g(k+1) - g(k) = \frac{13/6 - H_k}{(k+1)(k+2)}$. Thus $g(k)$ is increasing if and only if $H_k < \frac{13}{6}$. Since $H_4 = \frac{23}{12} < \frac{13}{6}$ and $H_5 = \frac{137}{60} > \frac{13}{6}$, we get that $\max_{k \geq 2} g(k) = g(5) = \frac{67}{360}$, so we have ratio $1\frac{67}{360}$.

We now show ratio $\frac{8}{7} < \frac{1555}{1347} < \frac{7}{6}$. As in the greedy algorithm for Set-Cover, we repeatedly remove an inclusion maximal set of disjoint stars with maximum number of leaves and pick the set of roots of these stars. The difference is that each time stars with more than k leaves are exhausted, we compute an α_k-approximate solution A_k for the remaining k-Set-Cover instance; we let $A_0 = \emptyset$. This gives many Set-Cover solutions, each is a union of the centers of stars picked and A_k; we choose the smallest one, and together with R this gives a feasible Activation Edge-Cover solution. Formally, the algorithm is:

Algorithm 2 ratio $\rho = \frac{1555}{1347} < 1.1545$

```
1 for $k \leftarrow \Delta$ downto 0 do
2    remove from $G$ a maximal collection of node disjoint $(k+1)$-stars
     let $C_{k+1}$ be the set of the roots of the stars removed so far
3    compute an $\alpha_k$-approximate $k$-Set-Cover solution $A_k$ in $G$
4 return the smallest set $C_{k+1} \cup A_k$, $k \in \{\Delta, \dots, 0\}$
```

Since we claim ratio $\frac{1555}{1347} > 1.15 > \frac{8}{7}$, at iterations when $k \geq 7$ step 3 can be skipped, since then we can apply a standard "local ratio" analysis [2]. Indeed, when a star with $k \geq 7$ terminals is removed, the partial solution value increases by $k+1$ while the optimum decreases by at least k. Hence for $k \geq 7$ it is a $\frac{k+1}{k} \leq \frac{8}{7}$ local ratio step. Consequently, we may assume that $\Delta \leq 6$, provided that we do not claim ratio better than $8/7$.

Let $r = |R|$. Let τ be the optimal value to the initial Set-Cover instance. At iteration k the algorithm computes a solution of value at most $\alpha_k \tau + r + |C_{k+1}|$. Thus we get ratio ρ if $\rho(r + \tau) \geq \alpha_k \tau + r + |C_{k+1}|$ holds for some $k \leq 6$. Otherwise,

$$\begin{aligned}
\rho(r+\tau) &< \alpha_6 \tau + r \\
\rho(r+\tau) &< \alpha_5 \tau + r + |C_6| \\
\rho(r+\tau) &< \alpha_4 \tau + r + |C_5| \\
\rho(r+\tau) &< \alpha_3 \tau + r + |C_4| \\
\rho(r+\tau) &< \alpha_2 \tau + r + |C_3| \\
\rho(r+\tau) &< \alpha_1 \tau + r + |C_2| \\
\rho(r+\tau) &< \qquad\quad r + |C_1|
\end{aligned}$$

Denote $\sigma = \alpha_1 + \cdots + \alpha_5 = \frac{1581}{240}$. Note that $|C_1| + \cdots + |C_6| = r$, since in this sum the number of stars with k leaves is summed exactly k times, $k = 1, \dots, 6$. The first inequality, and the inequality obtained as the sum of the other six inequalities gives the following two inequalities:

$$\begin{aligned}\rho(r+\tau) &< \alpha_6\tau + r\\ 6\rho(r+\tau) &< \sigma\tau + 7r\end{aligned}$$

Dividing both inequalities by τ and denoting $x = r/\tau$ gives:

$$\begin{aligned}\rho(x+1) &< \alpha_6 + x\\ 6\rho(x+1) &< \sigma + 7x\end{aligned}$$

Since $\rho > 1$ and $7 > 6\rho$ this is equivalent to:

$$\frac{6\rho-\sigma}{7-6\rho} < x < \frac{\alpha_6-\rho}{\rho-1}$$

We obtain a contradiction if ρ is the solution of the equation $\frac{6\rho-\sigma}{7-6\rho} = \frac{\alpha_6-\rho}{\rho-1}$, namely

$$\rho = \frac{7\alpha_6-\sigma}{6\alpha_6-\sigma+1} = 1 + \frac{\alpha_6-1}{6\alpha_6-\sigma+1} = 1 + \frac{208}{1347} = \frac{1555}{1347} \ .$$

This concludes the proof of Theorem 4.

References

1 S. Athanassopoulos, I. Caragiannis, and C. Kaklamanis. Analysis of Approximation Algorithms for k-Set Cover Using Factor-Revealing Linear Programs. *Theory Comput. Syst.*, 45(3):555–576, 2009.

2 Reuven Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. *Algorithmica*, 27(2):131–144, 2000.

3 J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. Steiner Tree Approximation via Iterative Randomized Rounding. *J. ACM*, 60(1):6:1–6:33, 2013.

4 G. Calinescu, G. Kortsarz, and Z. Nutov. Improved approximation algorithms for minimum power covering problems. In *WAOA*, pages 134–148, 2018.

5 K. Chandrasekaran, R. M. Karp, E. Moreno-Centeno, and S. Vempala. Algorithms for Implicit Hitting Set Problems. In *SODA*, pages 614–629, 2011.

6 V. Chvatal. Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

7 R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the Lambert W function. *Advances in Computational Mathematics*, 5:329–359, 1996.

8 Rong-chii Duh and Martin Fürer. Approximation of k-set cover by semi-local optimization. In *STOC*, pages 256–264, 1997.

9 J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

10 U. Feige. A threshold of $\ln n$ for approximating set cover. *J. of the ACM*, 45(4):634–652, 1998.

11 M. Fürer and H. Yu. Packing-Based Approximation Algorithm for the k-Set Cover Problem. In *ISAAC*, pages 484–493, 2011.

12 M. X. Goemans, N. Olver, T. Rothvoß, and R. Zenklusen. Matroids and integrality gaps for hypergraphic steiner tree relaxations. In *STOC*, pages 1161–1176, 2012.

13 C. Gröpl, S. Hougardy, T. Nierhoff, and H. J. Prömel. Approximation Algorithms for the Steiner Tree Problem in Graphs. In X. Z. Cheng and D-. Z. Du, editors, *Steiner Trees in Industry*, pages 235–279. Kluwer Academic Publishers, 2001.

14 M. Hajiaghayi, G. Kortsarz, V. Mirrokni, and Z. Nutov. Power optimization for connectivity problems. *Math. Programming*, 110(1):195–208, 2007.

15 A. Hoorfar and M. Hassani. Inequalities on the Lambert W Function and Hyperpower Function. *Journal of Inequalities in Pure and Applied Mathematics (IPAM)*, 9(2), 2008. Article 51, 5 pp.

16 D. S. Johnson. Approximation Algorithms for Combinatorial Problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

17 G. Kortsarz and Z. Nutov. Approximating minimum-power edge-covers and $2,3$-connectivity. *Discrete Applied Mathematics*, 157:1840–1847, 2009.

18 A. Levin. Approximating the unweighted k-set cover problem: greedy meets local search. In *WAOA*, pages 290–301, 2006.

19 Z. Nutov. Activation Network Design Problems. In T. F. Gonzalez, editor, *Handbook on Approximation Algorithms and Metaheuristics, Second Edition*, volume 2, chapter 15. Chapman & Hall/CRC, 2018.

20 D. Panigrahi. Survivable Network Design Problems in Wireless Networks. In *SODA*, pages 1014–1027, 2011.

21 R. Raz and S. Safra. A sub constant error probability low degree test, and a sub constant error probability PCP characterization of NP. In *STOC*, pages 475–484, 1997.

22 G. Robins and A. Zelikovsky. Tighter Bounds for Graph Steiner Tree Approximation. *SIAM J. Discrete Math.*, 19(1):122–134, 2005.

23 A. Schrijver. *Combinatorial Optimization, Polyhedra and Efficiency.* Springer-Verlag Berlin, Heidelberg New York, 2004.

24 D B. Shmoys. Approximation algorithms for facility location problems. In *APPROX*, pages 265–274, 2000.

25 P. Slavik. A Tight Analysis of the Greedy Algorithm for Set Cover. *J. Algorithms*, 25(2):237–254, 1997.

26 J. Vygen. Approximation algorithms for facility location problems (Lecture Notes). Technical Report 05950, Research Institute for Discrete Mathematics, University of Bonn, 2005.

27 L. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

28 N. E. Young. Greedy Set-Cover Algorithms. In *Encyclopedia of Algorithms*, pages 886–889. Springer, 2016.

Distributed Minimum Vertex Coloring and Maximum Independent Set in Chordal Graphs

Christian Konrad
Department of Computer Science, University of Bristol, UK
christian.konrad@bristol.ac.uk

Viktor Zamaraev
Department of Computer Science, Durham University, UK
viktor.zamaraev@durham.ac.uk

Abstract

We give deterministic distributed $(1+\varepsilon)$-approximation algorithms for Minimum Vertex Coloring and Maximum Independent Set on chordal graphs in the LOCAL model. Our coloring algorithm runs in $O(\frac{1}{\varepsilon}\log n)$ rounds, and our independent set algorithm has a runtime of $O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon})\log^* n)$ rounds. For coloring, existing lower bounds imply that the dependencies on $\frac{1}{\varepsilon}$ and $\log n$ are best possible. For independent set, we prove that $\Omega(\frac{1}{\varepsilon})$ rounds are necessary.

Both our algorithms make use of the tree decomposition of the input chordal graph. They iteratively peel off interval subgraphs, which are identified via the tree decomposition of the input graph, thereby partitioning the vertex set into $O(\log n)$ layers. For coloring, each interval graph is colored independently, which results in various coloring conflicts between the layers. These conflicts are then resolved in a separate phase, using the particular structure of our partitioning. For independent set, only the first $O(\log \frac{1}{\varepsilon})$ layers are required as they already contain a large enough independent set. We develop a $(1+\varepsilon)$-approximation maximum independent set algorithm for interval graphs, which we then apply to those layers.

This work raises the question as to how useful tree decompositions are for distributed computing.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases local model, approximation algorithms, minimum vertex coloring, maximum independent set, chordal graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.21

Related Version This work has previously been presented as a Brief Announcement at PODC 2018 [28]. A full version of the paper is available at [29], `https://arxiv.org/abs/1805.04544`.

Funding *Christian Konrad*: C.K. carried out most work on this paper while being at the University of Warwick. He was supported by the Centre for Discrete Mathematics and its Applications (DIMAP) at Warwick University and by EPSRC award EP/N011163/1.
Viktor Zamaraev: V.Z. is supported by EPSRC award EP/P020372/1.

1 Introduction

The LOCAL Model. In the LOCAL model of distributed computation [31], the input graph $G = (V, E)$ with $n = |V|$ represents a communication network, where every network node hosts a computational entity. Nodes have unique IDs. A distributed algorithm is executed on all network nodes simultaneously and proceeds in discrete rounds. Initially, besides their own IDs, nodes only know their neighbors. Each round consists of a computation and a communication phase. In the computation phase, nodes are allowed to perform unlimited computations. In the communication phase, nodes can send individual messages of unbounded sizes to all their neighbors (and receive messages from them as well). The runtime of the algorithm is the total number of communication rounds, and the objective is to design algorithms that run in as few rounds as possible. The output is typically distributed: For

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 21; pp. 21:1–21:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

vertex colorings, it is required that upon termination of the algorithm, every node knows its own color, and for independent sets, every node knows whether it participates in the independent set.

Network Decompositions. Network decompositions (see for example [3, 35, 32]) are a widely employed tool in distributed computing. They allow us to partition the vertex set of a graph into connected clusters of bounded diameter such that the cluster graph, i.e., the graph obtained by contracting the clusters into vertices, has small chromatic number. The Linial-Saks network decomposition method [32] guarantees that the cluster graph can be colored with $\mathrm{O}(\log n)$ colors (and provides such a coloring), and no better method is known with regards to the number of colors. When employing a network decomposition algorithm for the Minimum Vertex Coloring (MVC) problem, the chromatic number of the cluster graph directly translates to the achieved approximation factor: Iterate through the color classes of the cluster graph and make each cluster color itself *optimally* using exponential time computations (recall that MVC is NP-hard) and using a color palette that is disjoint to the colors employed by already colored neighboring clusters. Indeed, this techniques yields the currently best known distributed algorithm for MVC (with approx. factor $\mathrm{O}(\log n)$) [4].

Tree Decompositions. Our objective is to obtain distributed coloring algorithms with improved approximation guarantees, i.e., sub-logarithmic in n. To this end, we follow a route that at a first glance is very similar to the approach outlined above. However, instead of employing network decompositions, we make use of the *tree decomposition* of the input graph. A tree decomposition is a decomposition of the vertex set into (not necessarily disjoint) bags that are arranged in a tree such that every edge of the input graph is contained in the induced subgraph of at least one bag, and all bags that contain a specific vertex form a subtree in the tree decomposition. The key advantage of employing tree decompositions rather than network decompositions is that trees have small chromatic number and can efficiently be colored distributively. There are however multiple obstacles: First, there are graphs whose tree decompositions necessarily contain at least one bag whose diameter (maximum distance in the original graph between any two nodes in the bag) is $\Omega(n)$ (e.g. on a ring [16]). In these graphs, network nodes thus cannot learn the set of bags they are contained in in a small number of rounds and it appears impossible that nodes can obtain a local view of a coherent global tree decomposition. Second, the fact that every node may appear in multiple bags renders the coloring process more difficult since we cannot simply color independent parts of the tree simultaneously as some nodes may appear in multiple parts.

Our Results. Despite these obstacles, in this paper we show that the tree decomposition route can successfully be taken for the class of *chordal graphs*. A graph is chordal, if every cycle on at least four nodes contains a *chord*, i.e., an edge different from the edges of the cycle connecting two nodes of the cycle. Chordal graphs admit a tree decomposition where every bag has diameter 1, i.e., every bag is a clique. We first show that the tree decomposition of a chordal graph can efficiently be computed in the distributed setting. We then employ a peeling process on the tree decomposition that partitions the chordal graph into *interval subgraphs*, which can be colored efficiently distributively using a line of work by Halldórsson and Konrad [24, 25]. Particular care needs to be taken where these subgraphs meet, and we employ a color rotation technique to correct the colors at their boundaries. Perhaps surprisingly, this approach allows us to obtain a $(1+\varepsilon)$-approximation algorithm on this graph class, which constitutes our main result (**Theorem 14**). Our algorithm runs in $\mathrm{O}(\frac{1}{\varepsilon}\log n)$ rounds and prior work shows that the dependencies on both $\frac{1}{\varepsilon}$ and $\log n$ are optimal.

We further adapt this approach to the Maximum Independent Set (MIS) problem. In general graphs, a $(1+\varepsilon)$-approximation to MIS can be obtained in $\mathrm{O}(\frac{1}{\varepsilon}\log n)$ rounds [11, 21]. We show that using the tree decomposition approach outlined above, a $(1+\varepsilon)$-approximation on chordal graphs can be obtained in $\mathrm{O}(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon})\log^* n)$ rounds (**Theorem 16**), and we prove that $\Omega(\frac{1}{\varepsilon})$ rounds are necessary (**Theorem 17**).

Related Work: Distributed Vertex Coloring. Distributed vertex coloring has been studied since more than 30 years (e.g. [14, 23]). Given a graph $G = (V, E)$, a *(legal) c-coloring* of G is an assignment $\gamma : V \rightarrow \{1, 2, \dots, c\}$ of at most c colors to the nodes of G such that every pair of adjacent nodes receives different colors. The algorithmic challenge lies in computing colorings with few colors. The *chromatic number* $\chi(G)$ is the smallest c such that there is a c-coloring. The MVC problem asks to find a $\chi(G)$-coloring. This is a difficult task, even in the centralized setting: In general graphs, MVC is NP-complete [27] and hard to approximate within a factor of $n^{1-\varepsilon}$, for any $\varepsilon > 0$ [39].

Most research papers on distributed vertex coloring address the problem of computing a $(\Delta + 1)$-coloring (e.g. [14, 38, 37, 5, 8, 26, 18, 12]) (other objectives, such as Δ-colorings [36, 20] and other degree-based objectives [1] have been studied as well). Only few research papers address the MVC problem in a distributed model itself. On general graphs, the best distributed algorithm computes a $\mathrm{O}(\log n)$-approximation in $\mathrm{O}(\log^2 n)$ rounds [4] and is based on the network decomposition of Linial and Saks [32]. This algorithm uses exponential time computations, which due to the computational hardness of MVC is necessary unless $P = NP$. Barenboim et al. [7] gave a $\mathrm{O}(n^{\varepsilon})$-approximation algorithm that runs in $\exp(\mathrm{O}(1/\varepsilon))$ rounds. Both the exponential time computations and the relatively large best known approximation factor of $\mathrm{O}(\log n)$ on general graphs motivate the study of special graph classes. Besides results on graph classes with bounded chromatic number (planar graphs [23] and graphs of bounded arboricity [6, 22]), the only natural graph class with unbounded chromatic number that has been addressed in the literature are *interval graphs*, which are the intersection graphs of intervals on the line. Halldórsson and Konrad gave a $(1 + \varepsilon)$-approximation algorithm for MVC on interval graphs that runs in $\mathrm{O}(\frac{1}{\varepsilon}\log^* n)$ rounds [25] (see also [24]). This work is the most relevant related work to our results.

Related Work: Distributed Independent Sets. An *independent set* in a graph $G = (V, E)$ is a subset of non-adjacent nodes $I \subseteq V$. Algorithms for independent sets are usually designed with one of the following two objectives in mind: (1) Compute a *maximal independent set*, i.e., an independent set I that cannot be enlarged by adding a node outside I to it, or (2) Compute a Maximum Independent Set (MIS) (or an approximation thereof), i.e., an independent set of maximum size, which is the variant studied in this paper. Similar to MVC, the MIS problem is NP-complete [27] and hard to approximate within a factor of $n^{1-\varepsilon}$, for every $\varepsilon > 0$ [39]. In the distributed setting, Luby [33] and independently Alon et al. [2] gave distributed $\mathrm{O}(\log n)$ rounds maximal independent set algorithms more than 30 years ago. Improved results are possible for graphs with bounded maximum degree ([8, 19]) or on specific graph classes (e.g. [14, 38]). Using exponential time computations, a $(1 + \varepsilon)$-approximation to MIS can be computed in general graphs in $\mathrm{O}(\frac{1}{\varepsilon}\log n)$ rounds [11] (see also [21]). Deterministic distributed MIS algorithms may be inferior to randomized ones: It is known that every deterministic MIS $\mathrm{O}(1)$-approximation algorithm on a path requires $\Omega(\log^* n)$ rounds [30, 15], while a simple randomized $\mathrm{O}(1)$-round $\mathrm{O}(1)$-approximation algorithm exists [15].

Outline. In Section 2, we give notation and definitions. We then discuss in Section 3 how network nodes can obtain coherent local views of the tree decomposition. A centralized $(1+\varepsilon)$-approximation MVC algorithm is then presented in Section 4, and distributed implementation of this algorithm is given in Section 5. Due to space restrictions, we only briefly sketch our results on MIS in Section 6 and defer a complete exposition to the full version of this paper [29]. Finally, we conclude in Section 7.

2 Preliminaries

Basic Notation and Definitions. Let $G = (V, E)$ be a graph. For a node $v \in V$, we denote by $\Gamma_G(v)$ the *neighborhood* of v in G. The *degree* of v in G is defined as $\deg_G(v) := |\Gamma_G(v)|$. By $\Gamma_G[v]$ we denote the set $\Gamma_G(v) \cup \{v\}$. Similarly, for a set of nodes $W \subseteq V$ we write $\Gamma_G(W) := (\bigcup_{v\in W} \Gamma_G(v)) \setminus W$, and $\Gamma_G[W] := \Gamma_G(W) \cup W$. The *distance-$k$ neighborhood* of v in G, i.e., the set of nodes at distance at most k from v in G, is denoted $\Gamma_G^k(v)$. The subgraph of G induced by a set of nodes U is denoted by $G[U]$. A set of pairwise adjacent (resp., non-adjacent) nodes in G is called a *clique* (resp., an *independent set*). A clique (resp., an independent set) S is *maximal* if $S \cup \{v\}$ is not a clique (resp., an independent set), for every $v \in V \setminus S$. A *maximum* independent set in G is an independent set of maximum size. The cardinality of a maximum independent set is called the *independence number* of G. A graph is *chordal* if every cycle of length at least four contains a chord, i.e., an edge that connects two non-consecutive nodes of the cycle. It is a well-known fact that an n-node chordal graph has at most n maximal cliques.

Tree Decomposition. A *tree decomposition* of an n-node graph $G = (V, E)$ is a forest $\mathcal{T} = (\mathcal{S}, \mathcal{E})$ whose vertex set $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ is a family of subsets of V, and:

1. every node[1] $v \in V$ belongs to at least one subset in $\mathcal{S}$;
2. for every edge $uv \in E$, there is a subset $S_i \in \mathcal{S}$ containing both nodes u and v;
3. for every node $v \in V$ the family $\phi(\mathcal{T}, v) \subseteq \mathcal{S}$ of subsets containing v induces a tree in $\mathcal{T}$, which we denote $\mathcal{T}(v)$, i.e., $\mathcal{T}(v) := \mathcal{T}[\phi(\mathcal{T}, v)]$.

When the tree decomposition is clear from the context we will write $\phi(v)$ instead of $\phi(\mathcal{T}, v)$. It follows from the definition that G is a subgraph of the intersection graph of the trees $\mathcal{T}(v)$.

Tree Decomposition of Chordal Graphs. It is well known (see, e.g., [10]) that a graph G is chordal if and only if it has a tree decomposition $\mathcal{T} = (\mathcal{C}, \mathcal{E})$ whose vertex set $\mathcal{C}$ is the family of maximal cliques of G. We call such a tree decomposition a *clique forest* of chordal graph G. Since every vertex of the clique forest is a clique, G coincides with the intersection graph of the subtrees $\mathcal{T}(v)$ of clique forest $\mathcal{T}$. In other words, a clique forest of a chordal graph G is a forest $\mathcal{T} = (\mathcal{C}, \mathcal{E})$ whose vertex set $\mathcal{C}$ is the family of maximal cliques of G, such that $\mathcal{T}[\phi(v)]$ is a tree for every v. If a clique forest of a chordal graph is linear, i.e., a forest with every component being a path, then the graph is interval.

▶ **Theorem 1** ([17]). *A chordal graph G is interval iff its clique forest is a linear forest.*

[1] For convenience, throughout the paper, we say *node* when referring to a vertex of an underlying graph, and we say *vertex* when referring to a vertex of its tree decomposition.

Binary Paths. We say that a path $v_1, \ldots, v_k$ in G is *binary*, if $\deg_G(v_i) \leq 2$, for every $i \in [k]$ (note that this implies that $\deg_G(v_i) = 2$, for every $i \in \{2, 3, \ldots, k-1\}$). We say that a binary path $v_1, \ldots, v_k$ is a *pendant path*, if either $\deg_G(v_1) = 1$ or $\deg_G(v_k) = 1$ (or both). For convenience, we consider an isolated vertex as a pendant path. A binary path $v_1, \ldots, v_k$ is an *internal path*, if $\deg_G(v_i) = 2$, for every $i \in [k]$. A binary/pendant/internal path is *maximal* if it cannot be enlarged by adding a vertex outside the path to it.

Let $G = (V, E)$ be a chordal graph with clique forest $\mathcal{T} = (\mathcal{C}, \mathcal{E})$. Let $\mathcal{P} = C_1, \ldots, C_k$ be a binary path in $\mathcal{T}$. We define the *diameter* of $\mathcal{P}$ to be the maximum distance in G between nodes in $C_1 \cup \ldots \cup C_k$, that is, $diam(\mathcal{P}) = \max\limits_{u \in C_i, v \in C_j, i,j \in [k]} dist_G(u, v)$. Similarly, we define the *independence number* of $\mathcal{P}$ to be the independence number of $G[C_1 \cup \ldots \cup C_k]$.

Distributed Algorithms for Interval Graph. Halldórsson and Konrad [25] gave a deterministic distributed algorithm for coloring interval graphs. For every $\varepsilon \geq \frac{2}{\chi(G)}$, their algorithm computes a $(1+\varepsilon)$-approximation to MVC in $O(\frac{1}{\varepsilon} \log^* n)$ rounds. We will reuse this algorithm and denote it by $\textsc{ColIntGraph}(\varepsilon)$.

3 Computing Local Views of the Clique Forest

Our algorithms make use of the clique forest of the input chordal graph. For network nodes to obtain a coherent view of the clique forest, we make use of the following *maximum weight spanning forest* characterization: With a chordal graph G we associate the *weighted clique intersection graph* $\mathcal{W}_G$ whose vertex set is the family $\mathcal{C}$ of maximal cliques of G, and any two cliques $C_1, C_2 \in \mathcal{C}$ with a nonempty intersection are connected by an edge with weight $|C_1 \cap C_2|$. Then:

▶ **Theorem 2** ([9]). *A forest $\mathcal{T} = (\mathcal{C}, \mathcal{E})$ is a clique forest of a chordal graph G if and only if it is a maximum weight spanning forest of $\mathcal{W}_G$.*

Observe that while the vertex set of a clique forest is unique, i.e., the family of maximal cliques of G, the edge set is not necessarily unique as there may be multiple different maximum weight spanning forests in $\mathcal{W}_G$. To obtain coherent local views of a clique forest, it is thus necessary that nodes agree on a unique maximum weight spanning forest in $\mathcal{W}_G$. We achieve this by defining a linear order $<$ on the edges of $\mathcal{W}_G$ that respects the partial order given by the edge weights, and preferring edges that are larger with respect to $<$. To this end, we first assign to every maximal clique $C \in \mathcal{C}$ a word $\sigma(C)$ over the alphabet of the identifiers of nodes, where $\sigma(C)$ consists of the identifiers of the nodes in C listed in increasing order. Further, we associate with every edge $e = C_iC_j$ a triple (w_e, l_e, h_e), where w_e is the weight of e, i.e., $w_e = |C_i \cap C_j|$, $l_e = \text{lexmin}\{\sigma(C_i), \sigma(C_j)\}$, and $h_e = \text{lexmax}\{\sigma(C_i), \sigma(C_j)\}$. Now for two edges e and f we define $e < f$ if and only if either $w_e < w_f$, or $w_e = w_f$ and $l_e \prec l_f$, or $w_e = w_f$, $l_e = l_f$ and $h_e \prec h_f$, where $\prec$ is the lexicographical order. Clearly, $<$ orders the edges of $\mathcal{W}_G$ linearly while preserving the weight order.

In what follows, when we say that $\mathcal{T}$ is *the* clique forest of a chordal graph G, we implicitly assume that it is the clique forest uniquely specified by the above mechanism. Figure 2 demonstrates the weighted clique intersection graph and the clique forest of the chordal graph presented in Figure 1. A maximum weight spanning forest has the following easily verifiable local optimality property:

▶ **Lemma 3.** *Let $G = (V, E)$ be a weighted graph with a unique maximum weight spanning forest F, and let $U \subseteq V$ be a set of nodes inducing a tree T in F. Then $G[U]$ has a unique maximum weight spanning tree, which coincides with T.*

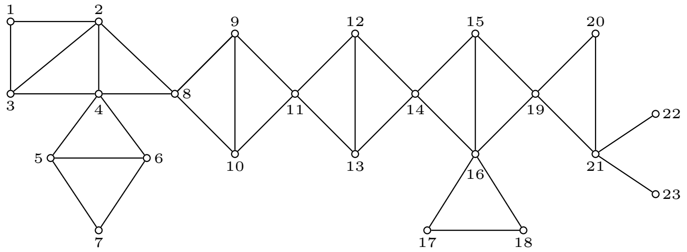


Figure 1 Chordal graph G.

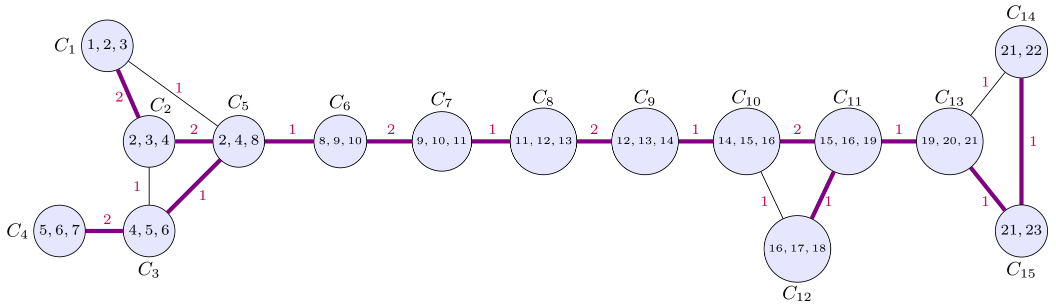


Figure 2 The weighted clique intersection graph $\mathcal{W}_G$ of chordal graph G presented in Fig. 1. The vertices of $\mathcal{W}_G$ are the maximal cliques of G, and two vertices C_i, C_j of $\mathcal{W}_G$ with a nonempty intersection are connected by an edge with weight $|C_1 \cap C_2|$. The bold edges are the edges of the clique forest $\mathcal{T}$ of G., i.e., the edges of the unique maximum weight spanning forest of $\mathcal{W}_G$ corresponding to the linear order of edges $<$.

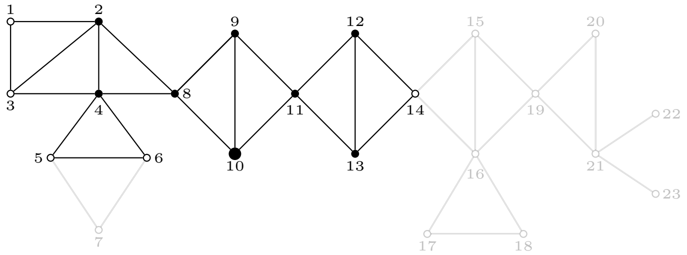


Figure 3 Local view of graph G from node 10. The non-gray nodes are the nodes in $\Gamma^3_G[10]$, and the black nodes are the nodes in $\Gamma^2_G[10]$.

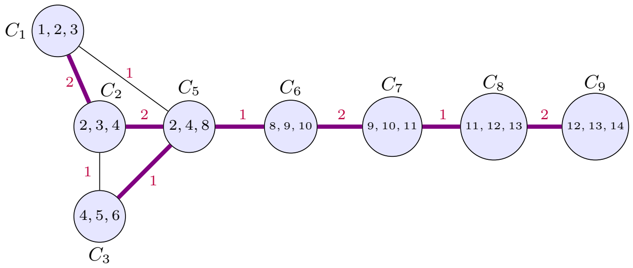


Figure 4 Local view of the graph $\mathcal{W}_G$ from node 10. The cliques in $\mathcal{C}' = \{C_1, C_2, C_3, C_5, C_6, C_7, C_8, C_9\}$ are exactly the maximal cliques of G that contain at least one node from $\Gamma^2_G[10]$. The bold edges are the edges of the unique maximum weight spanning forest of $\mathcal{W}_G[\mathcal{C}']$, which coincides with the subtree of $\mathcal{T}$ induced by $\mathcal{C}'$.

Applied to a chordal graph and its clique forest, we thus obtain:

▶ **Lemma 4.** *Let $G = (V, E)$ be a chordal graph and $\mathcal{T} = (\mathcal{C}, \mathcal{E})$ its clique forest. Then for any $v \in V$ the unique maximum weight spanning forest in $\mathcal{W}_G[\phi(v)]$ equals to tree $\mathcal{T}(v) = \mathcal{T}[\phi(v)]$.*

This suggests a method for a node $v \in V$ to compute a local view $\mathcal{T}'$ of clique forest $\mathcal{T}$: Suppose that v knows its distance d-neighborhood $\Gamma_G^d[v]$. For every $u \in \Gamma_G^{d-1}[v]$, v computes the family $\phi(u)$ of maximal cliques containing u (notice that a maximal clique that contains a node at distance $d-1$ from v may include nodes at distance d). Then, v computes the maximum weight spanning forest in every $\mathcal{W}_G[\phi(u)]$ and adds the edges of this forest to $\mathcal{T}'$. Figures 3 and 4 illustrate construction of a local view of the clique forest of the chordal graph presented in Figure 1.

4 Minimum Vertex Coloring: Centralized Algorithm

In this section, we give a centralized $(1+\varepsilon)$-approximation algorithm for MVC on chordal graphs. This algorithm will later be implemented in the LOCAL model in Section 5.

4.1 Algorithm

Algorithm 1 A centralized $(1+\varepsilon)$-approximation coloring algorithm for chordal graphs.

```
Input: $G = (V, E)$ is an $n$-node chordal graph with clique forest $\mathcal{T} = (\mathcal{C}, \mathcal{E})$; a parameter $\varepsilon > \frac{2}{\chi(G)}$.
   Set $k = 2/\varepsilon$.
(1) Pruning Phase.
   Let $\mathcal{T}_1 = \mathcal{T}$, $U_1 = V$.
   for $i = 1, 2, \ldots, \lceil \log n \rceil$ do:
      a. Let $\mathcal{L}_i$ be the set that contains all maximal pendant paths of $\mathcal{T}_i$, and all maximal internal paths of $\mathcal{T}_i$ of diameter at least $3k$.
      b. Let $V_i \subseteq U_i$ be such that for each $v \in V_i$, $\mathcal{T}(v)$ is a subpath of a path in $\mathcal{L}_i$.
      c. Let $U_{i+1} = U_i \setminus V_i$, and let $\mathcal{T}_{i+1}$ be the forest obtained from $\mathcal{T}_i$ by removing all paths in $\mathcal{L}_i$. As proved in Lemma 5, $\mathcal{T}_{i+1}$ is the clique forest of $G[U_{i+1}]$.
(2) Coloring Phase.
   for $i = 1, 2, \ldots, \lceil \log n \rceil$ do: Color $G[V_i]$ with at most $(1 + 1/k)\chi(G[V_i]) + 1$ colors.
(3) Color Correction Phase.
   for $i = \lceil \log n \rceil - 1, \lceil \log n \rceil - 2, \ldots, 1$ do:
      for each path $\mathcal{P} \in \mathcal{L}_i$ do:
         (a) Let $W \subseteq V_i$ be the set of nodes $w$ such that $\mathcal{T}(w)$ is a subpath of $\mathcal{P}$.
         (b) Let $W' \subseteq \bigcup_{l>i} V_l$ be the subset of nodes that have neighbors in $W$.
         (c) As we will show, $G[W \cup W']$ is an interval graph. Using Lemma 8, we recolor those nodes of $W$ that are at distance at most $k+3$ from a node in $W'$ using at most $(1 + 1/k)\chi(G[V_i]) + 1$ colors to resolve all coloring conflicts between $W$ and $W'$.
```

Our algorithm (Alg. 1) consists of the pruning, the coloring, and the color correction phases: In the pruning phase, the node set V is partitioned into at most $\lceil \log n \rceil$ *layers* $V_1, \ldots, V_{\lceil \log n \rceil}$ such that, for every $i \in [\lceil \log n \rceil]$, $G[V_i]$ constitutes an interval graph. In each step of the pruning phase, we remove every node $v \in U_i$ from the current graph $G[U_i]$ (we set $U_1 = V$ and hence $G[U_1] = G$) whose corresponding subtree $\mathcal{T}(v)$ in the clique forest $\mathcal{T}_i$ of $G[U_i]$ is a subpath of a pendant path or an internal path of diameter at least $3k$. The set of removed nodes is denoted V_i, and $G[V_i]$ forms an interval graph (which follows from Lemma 7, [29]). We prove in Lemma 5 that the clique forest $\mathcal{T}_{i+1}$ of the resulting graph $G[U_{i+1}]$, where

$U_{i+1} = U_i \setminus V_i$, can be obtained by removing all pendant paths and all internal paths of diameter at least $3k$ from $\mathcal{T}_i$. We also show in Lemma 6 that the pruning process ends after at most $\lceil \log n \rceil$ iterations and thus creates at most $\lceil \log n \rceil$ layers.

In the coloring phase, each interval graph $G[V_i]$ is colored with at most $(1+1/k)\chi(G[V_i])+1$ colors. In the centralized setting, it would be easy to color these interval graphs optimally. However, since we will implement the algorithm later in the distributed setting, and an optimal coloring on interval graphs cannot be computed distributively in few rounds, we impose a weaker quality guarantee that can be achieved distributively. The colorings of different layers are computed independently from each other and do not give a coherent coloring of the entire input graph.

In the color correction phase, these incoherences are corrected. To this end, the colors of $V_{\lceil \log n \rceil}$ remain unchanged and we correct the layers iteratively, starting with layer $\lceil \log n \rceil - 1$ and proceeding downwards to layer V_1. In a general step, for every path $\mathcal{P} \in \mathcal{L}_i$, we show that the subgraph induced by the nodes $W \subseteq V_i$ whose subtrees are subpaths of $\mathcal{P}$ forms an interval graph together with those nodes in $\bigcup_{j \geq i+1} V_j$ that have coloring conflicts towards W (Lemma 8, [29]). Notice that each path $\mathcal{P}$ connects to at most two maximal cliques in $\mathcal{T}_i$. The neighborhood of W thus consists of subsets of these (at most two) cliques, which further implies that all conflicting nodes in $\bigcup_{j \geq i+1} V_j$ are included in these cliques as well. We then reuse a recoloring result previously proved by Halldórsson and Konrad [25], which shows that we can resolve all conflicts by changing the colors of those nodes in W that are at distance at most $k+3$ from the (at most) two conflicting cliques.

4.2 Analysis

The analysis of our algorithm relies on various technical lemmas that are given in the full version of this paper [29]. We now give the most important lemmas that allow us to prove correctness of our algorithm. Concerning the pruning step, we show that $\mathcal{T}_i$ is indeed the clique forest of $G[U_i]$ in Lemma 5, and we prove that at most $\lceil \log n \rceil$ iterations are required to complete the pruning process[2] in Lemma 6.

▶ **Lemma 5.** *For every i, $\mathcal{T}_i$ is the clique forest of $G[U_i]$.*

▶ **Lemma 6.** *Phase 1 in Alg. 1 requires at most $\lceil \log n \rceil$ iterations, i.e., $\bigcup_{1 \leq i \leq \lceil \log n \rceil} V_i = V$.*

Next, we address the color correction phase. In each iteration of the phase we consider every path $\mathcal{P} \in \mathcal{L}_i$ independently. The subgraph induced by the set of nodes $W \subseteq V_i$ whose corresponding trees are subpaths of $\mathcal{P}$ is legally colored in the coloring phase. This coloring may be inconsistent with the coloring of subgraph $G[U_{i+1}]$. However, we prove in Lemma 8, [29], that the set $W' \subseteq \bigcup_{s>i} V_s = U_{i+1}$ of neighbors of W in $G[U_{i+1}]$ (i.e., the nodes in U_{i+1} that could potentially cause conflicts) is the union of at most two cliques, which are included in the end vertices of the clique forest of interval graph $G[W \cup W']$. In order to resolve these conflicts we carry out a recoloring process on interval graph $G[W \cup W']$ with fixed colorings of its "boundary" cliques. To this end, we reuse a result by Halldórsson and Konrad [25]:

▶ **Lemma 7** (Halldórsson and Konrad [25]). *Let $G = (V, E)$ be an interval graph with its clique forest $\mathcal{T} = (\mathcal{C}, \mathcal{E})$ being a path $\mathcal{P} = C_1, C_2, \ldots, C_k$ such that $\mathrm{dist}_G(u, v) \geq r$ for every pair of nodes $u \in C_1, v \in C_k$, for an integer $r \geq 5$. Suppose that cliques C_1 and C_k are legally colored using at most c colors. Then the coloring of $G[C_1 \cup C_k]$ can be extended to a legal coloring of G with at most $\max\{\lfloor (1 + \frac{1}{r-3})\chi(G) \rfloor + 1, c\}$ colors.*

[2] Recently, inspired by Miller and Reif's *parallel tree contraction* [34], a very similar procedure was developed by Chang and Pettie [13].

Equipped with Lemma 7, we now prove correctness of the color correction phase.

▶ **Lemma 8** (Recoloring Lemma). *Consider the color correction phase (Step 3) of Algorithm 1. Let $\mathcal{P} \in \mathcal{L}_i$ be a path and let $W \subseteq V_i$ be the subset of nodes whose corresponding subtrees are included in $\mathcal{P}$. Further, let $W' \subseteq \bigcup_{s>i} V_s = U_{i+1}$ be the nodes in U_{i+1} that have neighbors in W. Suppose that W' is colored using colors from the set $[\lfloor(1+1/k)\chi(G)+1\rfloor]$. Then, we can recolor those nodes of W that are at distance at most $k+4$ from W' in G with colors from the set $[\lfloor(1+1/k)\chi(G)+1\rfloor]$ so that $G[W \cup W']$ is legally colored.*

Proof. By Lemma 8, [29], $G[W \cup W']$ is an interval graph and its clique forest is a path. Let $\mathcal{P}' = C_1, C_2, \dots, C_r$ denote this path. The same lemma also states that $W' \subseteq C_1 \cup C_r$.

Let i be the minimum index such that $\mathrm{dist}(u,v) \geq k+3$, for every $u \in W' \cap C_1$ and $v \in C_i$. Then, by Lemma 7, the nodes of the cliques $C_2, \dots, C_{i-1}$ can be recolored using at most $\lfloor(1+1/k)\chi(G)\rfloor + 1$ colors to resolve the coloring conflicts between $W' \cap C_1$ and W. Similarly, let j be the maximum index such that $\mathrm{dist}(u,v) \geq k+3$, for every $u \in W' \cap C_r$ and $v \in C_j$. Then, by Lemma 7, the nodes of the cliques $C_{j+1}, \dots, C_{r-1}$ can be recolored using at most $\lfloor(1+1/k)\chi(G)\rfloor + 1$ colors to resolve the conflicts between $W' \cap C_r$ and W. ◀

▶ **Theorem 9.** *For any $\varepsilon > \frac{2}{\chi(G)}$, Algorithm 1 is a $(1+\varepsilon)$-approximation* MVC *algorithm on chordal graphs.*

Proof. First, we show by induction that the algorithm uses at most $(1+1/k)\chi(G)+1$ colors. This is clearly true for $G_{\lceil \log n \rceil}$. The induction step follows from Lemma 8. Now, using the assumption $\varepsilon > \frac{2}{\chi(G)}$, we obtain: $(1+1/k)\chi(G)+1 \leq (1+\varepsilon/2)\chi(G)+\varepsilon\chi(G)/2 = (1+\varepsilon)\chi(G)$, which proves the approximation factor of the algorithm. ◀

5 Minimum Vertex Coloring: Distributed Algorithm

We give now a LOCAL model implementation of Algorithm 1 that runs in $\mathrm{O}\left(\frac{1}{\varepsilon}\log n\right)$ rounds.

5.1 Algorithm

The global behavior of our distributed algorithm, Algorithm 2, is identical to that of our centralized Algorithm 1. The main challenge lies in the coordination of the network nodes. One particular difficulty stems from the fact that network nodes are not aware of n, the total number of nodes, and thus do not know when the $\lceil \log n \rceil$ iterations of the pruning phase have completed. For this reason, nodes execute the three phases of Algorithm 1 asynchronously.

We will first present the pseudocode of our distributed algorithm, which is executed independently on every node v. Then we will describe each of the three phases in detail.

Algorithm 2 A distributed $(1+\varepsilon)$-approximation algorithm, code for node v.

```
Input:  a parameter $\varepsilon$, let $k = \lceil 2/\varepsilon \rceil$
1. Pruning Phase. $(l_v, parent_v, children_v) \leftarrow$ PRUNETREE()
2. Coloring Phase. Run ColIntGraph($\frac{1}{k}$) on layer $l_v$ and store color in $c_v$
3. Color Correction Phase.
     if $parent_v \neq \bot$ then
         Wait until message SetColor($c$) received from $parent_v$;   Set $c_v \leftarrow c$;
     end if
     CorrectChildren($children_v, k$)
```

The Pruning Phase. In the pruning phase, the subroutine PruneTree is invoked and returns parameters l_v, $parent_v$, and $children_v$, where l_v is the layer of node v, and $parent_v$ and $children_v$ are variables necessary for the coordination of the color correction phase and are defined and explained later. The pseudocode of PruneTree is given in Algorithm 3.

Algorithm 3 PruneTree(), code for node v.

```
Initialization:
Let $i = 1$, $l_v = -1, children_v = \{\}$, and $parent_v = \perp$
while $l_v = -1$ do:
  1. Collect $\Gamma_G^{10k}(v)$ together with variables $l_u$ and $ID_u$, for every $u \in \Gamma_G^{10k}(v)$
  2. Compute local view of the clique forest $\mathcal{T}_i = (\mathcal{C}_i, \mathcal{E}_i)$ of the subgraph of $G$ induced by the
     nodes $u \in \Gamma_G^{10k}(v)$ with $l_u = -1$
  3. if $\mathcal{T}_i(v)$ is a subpath of a pendant path in $\mathcal{T}_i$, or $\mathcal{T}_i(v)$ is a subpath of a binary path in $\mathcal{T}_i$ of
     diameter at least $3k$ then
       $l_v = i$;    $parent_v$ = parent of $v$;
     else
       Add children in layer $i$ (if there are any) to $children_v[i]$
     end if
  4. $i = i + 1$
return $(l_v, parent_v, children_v)$
```

In each iteration of the while loop of PruneTree, one layer is removed from the clique forest of the input graph. To describe the global behavior of the algorithm, we will reuse the naming conventions already used in Algorithm 1. Let $U_1 = V$, and let $V_i \subseteq U_i$ be the set of nodes removed in iteration i, i.e., assigned layer index i. Let also $U_{i+1} = U_i \setminus V_i$, and let $\mathcal{L}_i$ be the set of maximal paths removed from the clique forest $\mathcal{T}_i$ of $G[U_i]$.

In each iteration i, first, each node v collects its distance-$10k$ neighborhood. Then, v computes its local view of the clique forest $\mathcal{T}_i$ of the graph induced by the nodes that have not yet been removed from the graph, i.e., of $G[U_i]$ (as in Section 3). Next, node v is removed from $G[U_i]$ and added to the current layer V_i if its corresponding subtrees $\mathcal{T}_i(v)$ is entirely contained in either a pendant path or a binary path of large enough diameter. This step is identical to Algorithm 1, and the exact same partitioning is computed. Node v that is removed in the current iteration stores its parent in $parent_v$, and nodes that remain in the graph potentially store some of the removed nodes as their children in $children_v$.

▶ **Definition 10** (Parent, Child). *Let $v \in V_i$ and let $\mathcal{P}$ be the maximal binary path in $\mathcal{T}_i$ that contains $\mathcal{T}_i(v)$. If $\mathcal{P}$ is a component of $\mathcal{T}_i$ then we define $parent_v := \perp$. Otherwise, let C be the vertex outside $\mathcal{P}$ in $\mathcal{T}_i$ such that C is adjacent to an end vertex of $\mathcal{P}$ and $dist_G(v, C)$ is minimal. Let c be the node with maximum ID in C. Then the* parent *of v is defined to be c, if $dist_G(v, C) \leq k + 3$, and $\perp$ otherwise. If c is the parent of v, then v is a* child *of c.*

The parent of node v is responsible for recoloring v in the color correction phase. Notice that a node v does not have a parent if the closest maximal clique outside v's path $\mathcal{P}$ is at least at distance $k + 4$ from v. In this case, the color that v will receive in the coloring phase is final and no color correction is needed for v. Recall that in the color correction phase of Algorithm 1, we only need to recolor nodes that are at distance at most $k + 3$ from the cliques that contain nodes with color conflicts. Finally, the subroutine returns the node's level l_v, its parent $parent_v$, and its children $children_v$.

The Coloring Phase. Notice that all nodes of layer i return from PruneTree in the same round. They can hence invoke the coloring phase simultaneously. They run the algorithm ColIntGraph of Halldórsson and Konrad [25] and compute a coloring on $G[V_i]$ that uses at most $\lfloor (1+\frac{1}{k})\chi(G[V_i]) + 1 \rfloor$ colors. This algorithm runs in $\mathrm{O}(k \log^* n)$ rounds.

While some nodes execute the coloring phase, others still execute PruneTree. These nodes repeatedly collect their distance-$10k$ neighborhood. This requires all other network nodes to continuously forward messages, which can be taken care of in the background.

The Color Correction Phase. In color correction phase, nodes with assigned parents (i.e., nodes v with $parent_v \neq \perp$) first wait until they received their final color from their parents. Only then they proceed and correct the colors of their children. To this end, each such node v runs subroutine CorrectChildren, which processes $children_v$ layer by layer, starting with layer $l_v - 1$ down to 1. If v has children in layer V_i, then it waits until all nodes adjacent to $children_v[i]$ which are contained in layers $> i$ have received their final colors. This can be done by repeatedly collecting its local distance-$(k+5)$ neighborhood and checking whether the colors of all nodes in $\Gamma_{G[U_i]}(children_v[i])$ are final. Then, v locally computes the color correction for $children_v[i]$ and notifies them about their new colors.

Algorithm 4 CorrectChildren($children_v, k$), code for node v.

```
for $l \leftarrow l_v - 1, l_v - 2, \ldots, 1$ do:
    if $children_v[l] \neq \{\}$ then
        a. Wait until all neighbors of $children_v[l]$ in $G[U_l]$ have received their final color
        b. Compute color correction as in Lemma 8
        c. For each $u \in children_v[l]$, send message SetColor($c$) to $u$, where $c$ is $u$'s new color
```

5.2 Analysis

To ensure correctness of our algorithm, we need to show that the parent of a node $v \in V_i$ is contained in a layer $j > i$. This is shown via the following lemma.

▶ **Lemma 11.** *Let $\mathcal{P} \in \mathcal{L}_i$, and let $W \subseteq V_i$ be the set of nodes whose corresponding subtrees are included in $\mathcal{P}$. Then every node $u \in \Gamma_{G[U_i]}(W)$ is contained in a layer V_j with $j > i$.*

It follows from the definition that the parent of a node $v \in V_i$ belongs to $\Gamma_{G[U_i]}(W)$. Hence:

▶ **Corollary 12.** *The parent of a node $v \in V_i$ is contained in a layer V_j with $j > i$.*

The following lemma demonstrates that Algorithm 2 mimics the behavior of our centralized algorithm and uses $\mathrm{O}(\frac{1}{\varepsilon}\log n)$ rounds. This establishes our main result, stated in Theorem 14.

▶ **Lemma 13.** *The global behavior of Algorithm 2 is identical to the behavior of Algorithm 1. Furthermore, Algorithm 2 runs in $\mathrm{O}(\frac{1}{\varepsilon}\log n)$ rounds.*

▶ **Theorem 14.** *For every $\varepsilon \geq \frac{2}{\chi(G)}$, there is a deterministic $(1+\varepsilon)$-approximation algorithm for* MVC *on chordal graphs that runs in $\mathrm{O}(\frac{1}{\varepsilon}\log n)$ rounds in the* LOCAL *model.*

6 Maximum Independent Set

Maximum Independent Set on Interval Graphs. Let $H = (V, E)$ be an interval graph, i.e., a chordal graph whose clique forest is a collection of paths. We first observe that the subset of nodes $V' \subseteq V$, with $u \in V'$ iff there exists a node $v \in V$ with $\Gamma_H[v] \subsetneq \Gamma_H[u]$, is not needed for the computation of a large maximum independent set: If a maximum independent set I^* in H contains a node $u \in V'$, then it can simply be replaced by a node v with $\Gamma_H[v] \subsetneq \Gamma_H[u]$. We thus only need to consider graph $H' := H[V \setminus V']$, which constitutes a unit interval graph. We first compute a distance-k maximal independent set I in $\mathrm{O}(k \log^* n)$ rounds, for some $k = \Theta(\frac{1}{\varepsilon})$, by simulating the maximal independent set $\mathrm{O}(\log^* n)$ rounds algorithm for bounded-independence graphs [38] on H'^k. Then, every $(v_1, v_2) \in P$, where P is the set of pairs of nodes of I that are of mutual distance at most $2k-1$, computes a maximum independent set I_{v_1,v_2} among the nodes *located between them* but outside of $\Gamma_{H'}(v_1) \cup \Gamma_{H'}(v_2)$ in $\mathrm{O}(k)$ rounds. Set $I \cup (\cup_{(v_1,v_2)\in P} I_{v_1,v_2})$ has the desired size.

▶ **Theorem 15.** *For every $\varepsilon > 0$, there is a deterministic $(1+\varepsilon)$-approximation algorithm for* MIS *on interval graphs that operates in $\mathrm{O}(\frac{1}{\varepsilon} \log^* n)$ rounds in the LOCAL model.*

Maximum Independent Set on Chordal Graphs. Our distributed MIS algorithm uses an adapted version of the peeling process used in our coloring algorithm. The key observation that allows us to obtain a runtime of $o(\log n)$ is the fact that the first $\mathrm{O}(\log \frac{1}{\varepsilon})$ layers computed by our peeling process already contain a large enough independent set. Our algorithm proceeds as follows: In each iteration $i = 1, \ldots, \mathrm{O}(\frac{1}{\varepsilon})$ of the peeling process, we remove set $\mathcal{L}_i$ of all pendant paths and all internal paths of large enough diameter from the clique forest $\mathcal{T}_i$ of the graph induced by the remaining nodes. Next, we compute large independent sets among the nodes whose trees are included in each path $\mathcal{P} \in \mathcal{L}_i$. If $\mathcal{P}$ has a large independence number then we run our $(1+\varepsilon)$-approximation algorithm for interval graphs in $\mathrm{O}(\frac{1}{\varepsilon} \log^* n)$ rounds. If $\mathcal{P}$ has small independence number we need to compute an optimal independent set in order to locally stay within a $(1+\varepsilon)$-approximation guarantee. This can be achieved using only $\mathrm{O}(\frac{1}{\varepsilon})$ rounds, since paths with small independence number necessarily have small diameter. The runtime is dominated by the product of the number of iterations $\mathrm{O}(\log \frac{1}{\varepsilon})$ and the $\mathrm{O}(\frac{1}{\varepsilon} \log^* n)$ runtime of our MIS algorithm for interval graphs.

▶ **Theorem 16.** *For any $\varepsilon \in (0, 1/2)$, there is a deterministic $(1+\varepsilon)$-approximation algorithm for* MIS *on chordal graphs that runs in $\mathrm{O}(\frac{1}{\varepsilon} \log(\frac{1}{\varepsilon}) \log^* n)$ rounds in the LOCAL model.*

When implementing this idea, care needs to be taken when combining the computed independent sets of different levels. Indeed, our algorithm bases the computation of the independent set in level i on the outcome of the computations of the independent sets of levels $< i$ and it seems difficult to avoid this. We therefore cannot execute the independent set computations of different levels simultaneously, which would reduce the runtime to $\mathrm{O}(\frac{1}{\varepsilon}(\log^* n + \log \frac{1}{\varepsilon}))$.

Lower Bound. Our lower bound is established on a path P_n of length n and uses an indistinguishability argument: Consider a k rounds MIS algorithm that operates on P_n, and let v be an arbitrary node that is not too close to an end point of the path. Suppose that v is selected into the independent set. Let u_1 be a node at distance $2k+1$ from v and let u_2 be the adjacent node to u_1 that is at distance $2k+2$ from v. Since the runtime of the algorithm is k, the outputs computed by u_1 and u_2 are independent from v. Since in average it is equally likely that u_1 or u_2 are selected, the expected size of an independent set in the subpath starting at v and ending at u_2 is thus strictly smaller than $k+2$, while a maximum independent set in this subpath is of size $k+2$. Based on this insight, we obtain

▶ **Theorem 17.** *For every $\varepsilon > 0$ and n large enough, every randomized algorithm in the LOCAL model with expected approximation factor at most $1+\varepsilon$ for* MIS *requires $\Omega(\frac{1}{\varepsilon})$ rounds.*

7 Conclusion

In this paper, we gave distributed $(1 + \varepsilon)$-approximation algorithms for MVC and MIS on chordal graphs. We showed that in chordal graphs network nodes can obtain coherent views of a global tree decomposition, which enabled us to exploit the tree structure of the input graph for the design of algorithms. How can we extend the class of graphs on which we can solve MVC and MIS within a small approximation factor even further? In particular, how can we handle graphs that contain longer induced cycles, such as k-chordal graphs (for some integer k)?

References

1 Pierre Aboulker, Marthe Bonamy, Nicolas Bousquet, and Louis Esperet. Distributed Coloring in Sparse Graphs with Fewer Colors. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 419–425, New York, NY, USA, 2018. ACM. doi:10.1145/3212734.3212740.

2 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.

3 Baruch Awerbuch, Michael Luby, Andrew V Goldberg, and Serge A Plotkin. Network Decomposition and Locality in Distributed Computation. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, SFCS '89, pages 364–369, Washington, DC, USA, 1989. IEEE Computer Society. doi:10.1109/SFCS.1989.63504.

4 Leonid Barenboim. On the Locality of Some NP-complete Problems. In *Proceedings of the 39th International Colloquium Conference on Automata, Languages, and Programming - Volume Part II*, ICALP'12, pages 403–415, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-31585-5_37.

5 Leonid Barenboim. Deterministic $(\Delta + 1)$-Coloring in Sublinear (in Δ) Time in Static, Dynamic and Faulty Networks. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 345–354, New York, NY, USA, 2015. ACM. doi:10.1145/2767386.2767410.

6 Leonid Barenboim and Michael Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. *Distributed Computing*, 22(5):363–379, 2010.

7 Leonid Barenboim, Michael Elkin, and Cyril Gavoille. A Fast Network-Decomposition Algorithm and Its Applications to Constant-Time Distributed Computation. In *Post-Proceedings of the 22nd International Colloquium on Structural Information and Communication Complexity - Volume 9439*, SIROCCO 2015, pages 209–223, New York, NY, USA, 2015. Springer-Verlag New York, Inc. doi:10.1007/978-3-319-25258-2_15.

8 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The Locality of Distributed Symmetry Breaking. *J. ACM*, 63(3):20:1–20:45, June 2016. doi:10.1145/2903137.

9 Philip A Bernstein and Nathan Goodman. Power of natural semijoins. *SIAM Journal on Computing*, 10(4):751–771, 1981.

10 Jean RS Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.

11 Marijke H.L. Bodlaender, Magnús M. Halldórsson, Christian Konrad, and Fabian Kuhn. Brief Announcement: Local Independent Set Approximation. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 93–95, New York, NY, USA, 2016. ACM. doi:10.1145/2933057.2933068.

12 Yi-Jun Chang, Wenzheng Li, and Seth Pettie. An optimal distributed $(\Delta + 1)$-coloring algorithm? In *Proceedings 50th ACM Symposium on Theory of Computing (STOC)*, pages 445–456, 2018.

13 Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the LOCAL model. *SIAM Journal on Computing*, 48(1):33–69, 2019.

14 Richard Cole and Uzi Vishkin. Deterministic Coin Tossing with Applications to Optimal Parallel List Ranking. *Inf. Control*, 70(1):32–53, July 1986. doi:10.1016/S0019-9958(86)80023-7.

15 Andrzej Czygrinow, Michal Hańćkowiak, and Wojciech Wawrzyniak. Fast Distributed Approximations in Planar Graphs. In Gadi Taubenfeld, editor, *Distributed Computing*, pages 78–92, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

16 Yon Dourisboure and Cyril Gavoille. Tree-decompositions with Bags of Small Diameter. *Discrete Math.*, 307(16):2008–2029, July 2007. doi:10.1016/j.disc.2005.12.060.

17 Peter C Fishburn. *Interval orders and interval graphs: A study of partially ordered sets*. John Wiley &amp; Sons, 1985.

18 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local Conflict Coloring. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 625–634, 2016.

19 Mohsen Ghaffari. An Improved Distributed Algorithm for Maximal Independent Set. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 270–277, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2884435.2884455.

20 Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, and Yannic Maus. Improved Distributed Delta-Coloring. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 427–436, New York, NY, USA, 2018. ACM. doi:10.1145/3212734.3212764.

21 Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the Complexity of Local Distributed Graph Problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 784–797, New York, NY, USA, 2017. ACM. doi:10.1145/3055399.3055471.

22 Mohsen Ghaffari and Christina Lymouri. Simple and Near-Optimal Distributed Coloring for Sparse Graphs. In *Distributed Computing: 31th International Symposium, DISC 2017, Vienna, Austria, October 16-20, 2017. Proceedings*, 2017.

23 Andrew V. Goldberg, Serge A. Plotkin, and Gregory E. Shannon. Parallel Symmetry-Breaking in Sparse Graphs. *SIAM J. Discrete Math.*, 1(4):434–446, 1988. doi:10.1137/0401044.

24 Magnús M. Halldórsson and Christian Konrad. *Distributed Algorithms for Coloring Interval Graphs*, pages 454–468. Springer, 2014. doi:10.1007/978-3-662-45174-8_31.

25 Magnús M. Halldórsson and Christian Konrad. Improved Distributed Algorithms for Coloring Interval Graphs with Application to Multicoloring Trees. In *Post-Proceedings of the 24th International Colloquium on Structural Information and Communication Complexity*, SIROCCO 2017, 2017.

26 David G. Harris, Johannes Schneider, and Hsin-Hao Su. Distributed $(\Delta + 1)$-coloring in Sublogarithmic Rounds. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 465–478, New York, NY, USA, 2016. ACM. doi:10.1145/2897518.2897533.

27 Richard M Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

28 Christian Konrad and Viktor Zamaraev. Brief Announcement: Distributed Minimum Vertex Coloring and Maximum Independent Set in Chordal Graphs. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 159–161, 2018. doi:10.1145/3212734.3212787.

29 Christian Konrad and Viktor Zamaraev. Distributed Coloring and Independent Set in Chordal Graphs. *arXiv preprint arXiv:1805.04544.*, 2018.

30 Christoph Lenzen and Roger Wattenhofer. Leveraging Linial's Locality Limit. In Gadi Taubenfeld, editor, *Distributed Computing*, pages 394–407, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

31 Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM J. Comput.*, 21(1):193–201, February 1992. doi:10.1137/0221015.

32 Nathan Linial and Michael Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, December 1993. doi:10.1007/BF01303516.

33 Michael Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986. doi:10.1137/0215074.

34 Gary L. Miller and John H. Reif. Parallel Tree Contraction–Part I: Fundamentals. In *Randomness and Computation*, volume 5, pages 47–72. JAI Press, 1989.

35 Alessandro Panconesi and Aravind Srinivasan. Improved Distributed Algorithms for Coloring and Network Decomposition Problems. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 581–592, New York, NY, USA, 1992. ACM. doi:10.1145/129712.129769.

36 Alessandro Panconesi and Aravind Srinivasan. The Local Natur of Delta-Coloring and its Algorithmic Applications. *Combinatorica*, 15(2):255–280, 1995. doi:10.1007/BF01200759.

37 Johannes Schneider and Roger Wattenhofer. A New Technique for Distributed Symmetry Breaking. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 257–266, New York, NY, USA, 2010. ACM. doi:10.1145/1835698.1835760.

38 Johannes Schneider and Roger Wattenhofer. An optimal maximal independent set algorithm for bounded-independence graphs. *Distributed Computing*, 22(5):349–361, August 2010. doi:10.1007/s00446-010-0097-1.

39 David Zuckerman. Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number. *Theory of Computing*, 3(1):103–128, 2007.

Multistage Knapsack

Evripidis Bampis
Sorbonne Université, CNRS, LIP6, France
evripidis.bampis@lip6.fr

Bruno Escoffier
Sorbonne Université, CNRS, LIP6, France
bruno.escoffier@lip6.fr

Alexandre Teiller
Sorbonne Université, CNRS, LIP6, France
alexandre.teiller@lip6.fr

Abstract

Many systems have to be maintained while the underlying constraints, costs and/or profits change over time. Although the state of a system may evolve during time, a non-negligible transition cost is incured for transitioning from one state to another. In order to model such situations, Gupta et al. (ICALP 2014) and Eisenstat et al. (ICALP 2014) introduced a *multistage* model where the input is a sequence of instances (one for each time step), and the goal is to find a sequence of solutions (one for each time step) that simultaneously (i) have good quality on the time steps and (ii) as stable as possible. We focus on the *multistage* version of the KNAPSACK problem where we are given a time horizon $t = 1, 2, \dots, T$, and a sequence of knapsack instances $I_1, I_2, \dots, I_T$, one for each time step, defined on a set of n objects. In every time step t we have to choose a feasible knapsack S_t of I_t, which gives a *knapsack profit*. To measure the stability/similarity of two consecutive solutions S_t and S_{t+1}, we identify the objects for which the decision, to be picked or not, remains the same in S_t and S_{t+1}, giving a *transition profit*. We are asked to produce a sequence of solutions $S_1, S_2, \dots, S_T$ so that the total knapsack profit plus the overall transition profit is maximized.

We propose a PTAS for the MULTISTAGE KNAPSACK problem. This is the first approximation scheme for a combinatorial optimization problem in the considered multistage setting, and its existence contrasts with the inapproximability results for other combinatorial optimization problems that are even polynomial-time solvable in the static case (e.g.MULTISTAGE SPANNING TREE, or MULTISTAGE BIPARTITE PERFECT MATCHING). Then, we prove that there is no FPTAS for the problem even in the case where $T = 2$, unless $P = NP$. Furthermore, we give a pseudopolynomial time algorithm for the case where the number of steps is bounded by a fixed constant and we show that otherwise the problem remains NP-hard even in the case where all the weights, profits and capacities are 0 or 1.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Knapsack, Approximation Algorithms, Multistage Optimization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.22

Acknowledgements This research benefited from the support of FMJH program PGMO and from the support of EDF-Thalès-Orange.

1 Introduction

In a classical combinatorial optimization problem, given an instance of the problem we seek a feasible solution optimizing the objective function. However, in many systems the input may change over the time and the solution has to be adapted to the input changes. It is then necessary to determine a tradeoff between the optimality of the solutions in each time step and the stability/similarity of consecutive solutions. This is important since in many applications there is a significant transition cost for changing (parts of) a solution. Recently,

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 22; pp. 22:1–22:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Gupta et al. [15] and Eisenstat et al. [11] introduced a *multistage* model in order to deal with such situations. They consider that the input is a sequence of instances (one for each time step), and the goal is to find a sequence of solutions (one for each time step) reaching such a tradeoff.

Our work follows the direction proposed by Gupta et al. [15] who suggested the study of more combinatorial optimization problems in their multistage framework. In this paper, we focus on the multistage version of the KNAPSACK problem. Consider a company owning a set $N = \{u_1, \dots, u_n\}$ of production units. Each unit can be used or not; if u_i is used, it spends an amount w_i of a given resource (energy, raw material,...), and generates a profit p_i. Given a bound W on the global amount of available resource, the static KNAPSACK problem aims at determining a feasible solution that specifies the chosen units in order to maximize the total profit under the constraint that the total amount of the resource does not exceed the bound of W. In a multistage setting, considering a time horizon $t = 1, 2, \dots, T$ of, let us say, T days, the company needs to decide a production plan for each day of the time horizon, given that data (such as prices, level of resources,...) usually change over time. This is a typical situation, for instance, in energy production planning (like electricity production, where units can be nuclear reactors, wind or water turbines,...), or in data centers (where units are machines and the resource corresponds to the available energy). Moreover, in these examples, there is an extra cost to turn ON or OFF a unit like in the case of turning ON/OFF a reactor in electricity production [25], or a machine in a data center [1]. Obviously, whenever a reactor is in the ON or OFF state, it is beneficial to maintain it at the same state for several consecutive time steps, in order to avoid the overhead costs of state changes. Therefore, the design of a production plan over a given time horizon has to take into account both the profits generated each day from the operation of the chosen units, as well as the potential transition profits from maintaining a unit at the same state for two consecutive days.

We formalize the problem as follows. We are given a time horizon $t = 1, 2, \dots, T$, and a sequence of knapsack instances $I_1, I_2, \dots, I_T$, one for each time step, defined on a set of n objects. In every time step t we have to choose a feasible knapsack S_t of I_t, which gives a *knapsack profit.* Taking into account transition costs, we measure the stability/similarity of two consecutive solutions S_t and S_{t+1} by identifying the objects for which the decision, to be picked or not, remains the same in S_t and S_{t+1}, giving a *transition profit.* We are asked to produce a sequence of solutions $S_1, S_2, \dots, S_T$ so that the total knapsack profit plus the overall transition profit is maximized.

Our main contribution is a polynomial time approximation scheme (PTAS) for the multistage version of the KNAPSACK problem. Up to the best of our knowledge, this is the first approximation scheme for a multistage combinatorial optimization problem and its existence contrasts with the inapproximability results for other combinatorial optimization problems that are even polynomial-time solvable in the static case (e.g. MULTISTAGE SPANNING TREE [15], or MULTISTAGE BIPARTITE PERFECT MATCHING [4]).

1.1 Problem definition

Formally, the MULTISTAGE KNAPSACK problem can be defined as follows.

▶ **Definition 1.** *In the* MULTISTAGE KNAPSACK *problem (MK) we are given:*

- *a time horizon $T \in \mathbb{N}^*$, a set $N = \{1, 2, \dots, n\}$ of objects;*
- *For any $t \in \{1, \dots, T\}$, any $i \in N$:*
  - *p_{ti} the profit of taking object i at time t*
  - *w_{ti} the weight of object i at time t*

- *For any $t \in \{1, \ldots, T-1\}$, any $i \in N$: $B_{ti} \in \mathbb{R}^+$ the bonus of the object i if we keep the same decision for i at time t and $t+1$.*
- *For any $t \in \{1, \ldots, T\}$: the capacity C_t of the knapsack at time t.*

We are asked to select a subset $S_t \subseteq N$ of objects at each time t so as to respect the capacity constraint: $\sum_{i \in S_t} w_{ti} \leq C_t$. To a solution $S = (S_1, \ldots, S_T)$ are associated:

- *A knapsack profit $\sum_{t=1}^{T} \sum_{i \in S_t} p_{ti}$ corresponding to the sum of the profits of the T knapsacks;*
- *A transition profit $\sum\limits_{t=1}^{T-1} \sum\limits_{i \in \Delta_t} B_{ti}$ where Δ_t is the set of objects either taken or not taken at both time steps t and $t+1$ in S (formally $\Delta_t = (S_t \cap S_{t+1}) \cup (\overline{S_t} \cap \overline{S_{t+1}})$).*

The value of the solution S is the sum of the knapsack profit and the transition profit, to be maximized.

1.2 Related works

Multistage combinatorial optimization. A lot of optimization problems have been considered in online or semi-online settings, where the input changes over time and the algorithm has to modify the solution by making as few changes as possible. Tradeoffs between modification costs and quality of solutions have been also studied in the reoptimization setting. We refer the reader to [3, 6, 10, 14, 22, 23, 26] and the references therein.

Multistage optimization has been studied for fractional problems by Buchbinder et al. [8] and Buchbinder, Chen and Naor [7]. The multistage model considered in this article is the one studied in Eisenstat et al. [11] and Gupta et al. [15]. Eisenstat et al. [11] studied the multistage version of facility location problems. They proposed a logarithmic approximation algorithm. An et al. [2] obtained constant factor approximation for some related problems. Gupta et al. [15] studied the Multistage Maintenance Matroid problem for both the offline and the online settings. They presented a logarithmic approximation algorithm for this problem, which includes as a special case a natural multistage version of Spanning Tree. The same paper also introduced the study of the Multistage Minimum Perfect Matching problem. They showed that the problem becomes hard to approximate even for a constant number of stages. Later, Bampis et al. [4] showed that the problem is hard to approximate even for bipartite graphs and for the case of two time steps. In the case where the edge costs are metric within every time step they first proved that the problem remains APX-hard even for two time steps. They also showed that the maximization version of the problem admits a constant factor approximation algorithm but is APX-hard. In another work [5], the Multistage Max-Min Fair Allocation problem has been studied in the offline and the online settings. This corresponds to a multistage variant of the Santa Klaus problem. For the off-line setting, the authors showed that the multistage version of the problem is much harder than the static one. They provided constant factor approximation algorithms for the off-line setting.

Knapsack variants. Our work builds upon the Knapsack literature [18]. It is well known that there is a simple 2-approximation algorithm as well as a fully polynomial time (FPTAS) for the static case [16, 20, 21, 17]. There are two variants that are of special interest for our work:

(i) The first variant is a generalization of the Knapsack problem known as the k-Dimensional Knapsack ($k - DKP$) problem:

▶ **Definition 2.** *In the k-*Dimensional Knapsack *problem ($k - DKP$), we have a set $N = \{1, 2, \dots, n\}$ of objects. Each object i has a profit p_i and k weights w_{ji}, $j = 1, \dots, k$. We are also given k capacities C_j. The goal is to select a subset $Y \subseteq N$ of objects such that:*

- *The capacity constraints are respected: for any j, $\sum_{i \in Y} w_{ji} \leq C_j$;*
- *The profit $\sum_{i \in Y} p_i$ is maximized.*

It is well known that for the usual Knapsack problem, in the continuous relaxation (variables in $[0, 1]$), at most one variable is fractional. Caprara et al. [9] showed that this can be generalized for $k - DKP$.

Let us consider the following ILP formulation ($ILP - DKP$) of the problem:

$$\begin{cases} \max \sum\limits_{i \in N} p_i y_i \\ s.t. \left| \begin{array}{lcll} \sum\limits_{i \in N} w_{ji} y_i & \leq & C_j & \forall j \in \{1, ..., k\} \\ y_i \in \{0, 1\} & & & \forall i \in N \end{array} \right. \end{cases}$$

▶ **Theorem 3.** *[9] In the continuous relaxation ($LP - DKP$) of ($ILP - DKP$) where variables are in $[0, 1]$, in any basic solution at most k variables are fractional.*

A basic solution is an extreme point (vertex) of the polytope of solutions. Note that with an easy affine transformation on variables, the same result holds when variable y_i is subject to $a_i \leq y_i \leq b_i$ instead of $0 \leq y_i \leq 1$: *in any basic solution at most k variables y_i are such that $a_i < y_i < b_i$.*

Caprara et al. [9] use the result of Theorem 3 to show that for any fixed constant k, $k - DKP$ admits a polynomial time approximation scheme (PTAS). Other PTASes have been presented in [24, 12]. Korte and Schrader [19] showed that there is no FPTAS for $k - DKP$ unless $P = NP$.

(ii) The second related variant is a simplified version of $k - DKP$ called Cardinality-2-KP, where the dimension is 2, all the profits are 1 and, given a K, we are asked if there is a solution of value at least K (decision problem). In other words, given two knapsack constraints, can we take K objects and verify the two constraints? The following result is shown in [18].

▶ **Theorem 4.** *[18]* Cardinality-2-KP *is NP-complete.*

1.3 Our contribution

As stated before, our main contribution is to propose a PTAS for the multistage Knapsack problem. Furthermore, we prove that there is no FPTAS for the problem even in the case where $T = 2$, unless $P = NP$. We also give a pseudopolynomial time algorithm for the case where the number of steps is bounded by a fixed constant and we show that otherwise the problem remains NP-hard even in the case where all the weights, profits and capacities are 0 or 1. The following table summarizes our main result pointing out the impact of the number T of time steps on the difficulty of the problem ("no FPTAS" means "no FPTAS unless P=NP").

$T = 1$	T fixed	any T
pseudopolynomial	pseudopolynomial	strongly NP-hard
FPTAS	PTAS	PTAS
-	no FPTAS	no FPTAS

We point out that the negative results (strongly NP-hardness and no FPTAS) hold even in the case of *uniform bonus*, i.e., when $B_{ti} = B$ for all $i \in N$ and all $t = 1, \dots, T - 1$.

2 ILP formulation

The Multistage Knapsack problem can be written as an ILP as follows. We define Tn binary variables x_{ti} equal to 1 if i is taken at time t ($i \in S_t$) and 0 otherwise. We also define $(T-1)n$ binary variables z_{ti} corresponding to the transition profit of object i between time t and $t+1$. The profit is 1 if i is taken at both time steps, or taken at none, and 0 otherwise. Hence, $z_{ti} = 1 - |x_{(t+1)i} - x_{ti}|$. Considering that we solve a maximization problem, this can be linearized by the two inequalities: $z_{ti} \leq -x_{(t+1)i} + x_{ti} + 1$ and $z_{ti} \leq x_{(t+1)i} - x_{ti} + 1$. We end up with the following ILP (called $ILP - MK$):

$$\begin{cases} \max \sum_{t=1}^{T} \sum_{i \in N} p_{ti} x_{ti} + \sum_{t=1}^{T-1} \sum_{i \in N} z_{ti} B_{ti} & \\ s.t. \left| \begin{array}{llll} \sum_{i \in N} w_{ti} x_{ti} & \leq & C_t & \forall t \in \{1, ..., T\} \\ z_{ti} & \leq & -x_{(t+1)i} + x_{ti} + 1 & \forall t \in \{1, ..., T-1\}, \forall i \in N \\ z_{ti} & \leq & x_{(t+1)i} - x_{ti} + 1 & \forall t \in \{1, ..., T-1\}, \forall i \in N \\ x_{ti} \in \{0,1\} & & & \forall t \in \{1, ..., T\}, \forall i \in N \\ z_{ti} \in \{0,1\} & & & \forall t \in \{1, ..., T-1\}, \forall i \in N \end{array} \right. \end{cases}$$

In devising the PTAS we will extensively use the linear relaxation $(LP - MK)$ of $(ILP - MK)$ where variables x_{ti} and z_{ti} are in $[0,1]$.

3 A polynomial time approximation scheme

In this section we show that Multistage Knapsack admits a PTAS. The central part of the proof is to derive a PTAS when the number of steps is a fixed constant (Sections 3.1 and 3.2). The generalization to an arbitrary number of steps is done in Section 3.3.

Building upon [9], our PTAS for a fixed number of time steps heavily relies on a property of the relaxed LP-formulation of Multistage Knapsack: we show that there are at most T^3 fractional variables in an optimal (basic) solution of the (relaxed) Multistage Knapsack problem. Based on this bound, the PTAS is built from a combination of (1) bruteforce search (to find the most profitable objects), (2) a preprocessing step and (3) a rounding of the fractional solution of the (relaxed) LP-formulation. The preprocessing step associated to the bound on the number of fractional variables allow to bound the global loss of the solution built by the algorithm.

We show how to bound the number of fractional variables in Section 3.1. We first illustrate the reasoning on the case of two time-steps, and then present the general result. In Section 3.2 we present the PTAS for a constant number of steps. For ease of notation, we will sometimes write a feasible solution as $S = (S_1, \ldots, S_T)$ (subsets of objects taken at each time step), or as $S = (x, z)$ (values of variables in $(ILP - MK)$ or $(LP - MK)$).

3.1 Bounding the number of fractional objects in $(LP - MK)$

3.1.1 Warm-up: the case of two time-steps

We consider in this section the case of two time-steps ($T = 2$), and focus on the linear relaxation $(LP - MK)$ of $(ILP - MK)$ with the variables x_{ti} and z_i in $[0,1]$ (we write z_i instead of z_{1i} for readability). We say that an object is *fractional* in a solution S if x_{1i}, x_{2i} or z_i is fractional.

Let us consider a (feasible) solution $\hat{S} = (\hat{x}, \hat{z})$ of $(LP - MK)$, where $\hat{z}_i = 1 - |\hat{x}_{2i} - \hat{x}_{1i}|$ (variables $\hat{z}_i$ are set to their optimal value w.r.t. $\hat{x}$). We show the following.

▶ **Proposition 5.** *If $\hat{S}$ is a basic solution of $(LP - MK)$, at most 4 objects are fractional.*

Proof. First note that since we assume $\hat{z}_i = 1 - |\hat{x}_{1i} - \hat{x}_{2i}|$, if $\hat{x}_{1i}$ and $\hat{x}_{2i}$ are both integers then $\hat{z}_i$ is an integer. So if an object i is fractional either $\hat{x}_{1i}$ or $\hat{x}_{2i}$ is fractional.

Let us denote:

- L the set of objects i such that $\hat{x}_{1i} = \hat{x}_{2i}$.
- $P = N \setminus L$ the set of objects i such that $\hat{x}_{1i} \neq \hat{x}_{2i}$.

We first show Fact 1.

Fact 1. In P there is at most one object i with $\hat{x}_{1i}$ fractional.

Suppose that there are two such objects i and j. Note that since $0 < |\hat{x}_{1i} - \hat{x}_{2i}| < 1$, $\hat{z}_i$ is fractional, and so is $\hat{z}_j$. Then, for a sufficiently small $\epsilon > 0$, consider the solution S_1 obtained from $\hat{S}$ by transfering at time 1 an amount ϵ of weight from i to j (and adjusting consequently z_i and z_j). Namely, in S_1:

- $x^1_{1i} = \hat{x}_{1i} - \frac{\epsilon}{w_{1i}}$, $z^1_i = \hat{z}_i - d_i \frac{\epsilon}{w_{1i}}$, where $d_i = 1$ if $\hat{x}_{2i} > \hat{x}_{1i}$ and $d_i = -1$ if $\hat{x}_{2i} < \hat{x}_{1i}$ (since i is in P $\hat{x}_{2i} \neq \hat{x}_{1i}$).
- $x^1_{1j} = \hat{x}_{1j} + \frac{\epsilon}{w_{1j}}$, $z^1_j = \hat{z}_i + d_j \frac{\epsilon}{w_{1j}}$, where $d_j = 1$ if $\hat{x}_{2j} > \hat{x}_{1j}$ and $d_j = -1$ otherwise.

Note that (for ϵ sufficiently small) S_1 is feasible. Indeed (1) $\hat{x}_{1i}, \hat{x}_{1j}, \hat{z}_i$ and $\hat{z}_j$ are fractional (2) the weight of the knapsack at time 1 is the same in S_1 and in $\hat{S}$ (3) if $\hat{x}_{1i}$ increases by a small δ, if $\hat{x}_{2i} > \hat{x}_{1i}$ then $|\hat{x}_{2i} - \hat{x}_{i1}|$ decreases by δ so $\hat{z}_i$ can increase by δ (so $d_i = 1$), and if $\hat{x}_{2i} < \hat{x}_{i1}$ then $\hat{z}_i$ has to decrease by δ (so $d_i = -1$), and similarly for $\hat{x}_{1j}$.

Similarly, let us define S_2 obtained from $\hat{S}$ with the reverse transfer (from j to i). In S_2:

- $x^2_{1i} = \hat{x}_{1i} + \frac{\epsilon}{w_{1i}}$, $z^2_i = \hat{z}_i + d_i \frac{\epsilon}{w_{1i}}$
- $x^2_{1j} = \hat{x}_{1j} - \frac{\epsilon}{w_{1j}}$, $z^2_j = \hat{z}_i - d_j \frac{\epsilon}{w_{1j}}$

As previously, S_2 is feasible. Then $\hat{S}$ is clearly a convex combination of S_1 and S_2 (with coefficient 1/2), so not a basic solution, and Fact 1 is proven.

In other words (and this interpretation will be important in the general case), for this case we can focus on variables at time one, and interpret *locally* the problem as a (classical, unidimensional) fractional knapsack problem. By locally, we mean that we consider as fixed the variables at time 2: for variables at time 1, if $\hat{x}_{1i} < \hat{x}_{2i}$ then x_{1i} must be in $[0, \hat{x}_{2i}]$ (in S^1, x^1_{1i} cannot be larger than $\hat{x}_{2i}$, otherwise the previous value of z^1_i would be erroneous); similarly if $\hat{x}_{1i} > \hat{x}_{2i}$ then x_{1i} must be in $[\hat{x}_{2i}, 1]$. The profit associated to object i is $p_{1i} + d_i B_{1i}$ (if x_{i1} increases/decreases by ϵ, then the knapsack profit increases/decreases by $p_{1i}\epsilon$, and the transition profit increases/decreases by $\epsilon d_i B_{1i}$, as explained above). Then we have at most one fractional variable, as in any fractional knapsack problem.

In P there is at most one object i with $\hat{x}_{1i}$ fractional. Similarly there is at most one object k with $\hat{x}_{2k}$ fractional. In P, for all but at most two objects, both $\hat{x}_{1i}$ and $\hat{x}_{2i}$, and thus $\hat{z}_i$, are integers.

Note that this argument would not hold for variables in L. Indeed if $\hat{x}_{1i} = \hat{x}_{2i}$, then $\hat{z}_i = 1$, and the transition profit decreases in *both* cases: when $\hat{x}_{1i}$ increases by $\delta > 0$ and when it decreases by δ. So, we cannot express $\hat{S}$ as a convex combination of S_1 and S_2 as previously.

However, let us consider the following linear program $2-DKP$ obtained by fixing variables in P to their values in $\hat{S}$, computing the remaining capacities $C'_t = C_t - \sum_{j \in P} w_{tj}\hat{x}_{tj}$, and "imposing" $x_{1i} = x_{2i}$:

$$\begin{cases} \max \sum\limits_{i\in L}(p_{1i}+p_{2i})y_i + \sum\limits_{i\in L} B_{1i} & & \\ \qquad\sum\limits_{i\in L} w_{1i}y_i & \leq & C'_1 \\ \qquad\sum\limits_{i\in L} w_{2i}y_i & \leq & C'_2 \\ \qquad y_i \in [0,1] & & \forall i \in L \end{cases}$$

Clearly, the restriction of $\hat{S}$ to variables in L is a solution of $2-DKP$. Formally, let $\hat{S}_L = (\hat{y}_j, j \in L)$ defined as $\hat{y}_j = \hat{x}_{1j}$. $\hat{S}_L$ is feasible for $2-DKP$. Let us show that it is basic: suppose a contrario that $\hat{S}_L = \frac{S^1_L+S^2_L}{2}$, with $S^1_L = (y^1_i, i \in L) \neq S^2_L$ two feasible solutions of $2-DKP$. Then consider the solution $S^1 = (x^1, y^1)$ of $(LP-MK)$ defined as:

- If $i \in L$ then $x^1_{1i} = x^1_{2i} = y^1_i$, and $z^1_{1i} = 1 = \hat{z}_{1i}$.
- Otherwise (for i in P) S^1 is the same as $\hat{S}$.

S^1 is clearly a feasible solution of MULTISTAGE KNAPSACK. If we do the same for S^2_L, we get a (different) feasible solution S^2, and $\hat{S} = \frac{S^1+S^2}{2}$, so $\hat{S}$ is not basic, a contradiction.

By the result of [9], $\hat{S}_L$ has at most 2 fractional variables. Then, in L, for all but at most 2 variables both $\hat{x}_{1i}$, $\hat{x}_{2i}$ and $\hat{z}_i$ are integers. ◀

3.1.2 General case

The case of 2 time steps suggests to bound the number of fractional objects by considering 3 cases:

- Objects with $\hat{x}_{1i}$ fractional and $\hat{x}_{1i} \neq \hat{x}_{2i}$. As explained in the proof of Proposition 5, this can be seen locally (as long as x_{1i} does not reach $\hat{x}_{2i}$) as a knapsack problem from which we can conclude that there is at most 1 such fractional object.
- Similarly, objects with $\hat{x}_{2i}$ fractional and $\hat{x}_{1i} \neq \hat{x}_{2i}$.
- Objects with $\hat{x}_{1i} = \hat{x}_{2i}$ fractional. As explained in the proof of Proposition 5, this can be seen as a $2-DKP$ from which we can conclude that there are at most 2 such fractional objects.

For larger T, we may have different situations. Suppose for instance that we have 5 time steps, and a solution (x, z) with an object i such that: $x_{1i} < x_{2i} = x_{3i} = x_{4i} < x_{5i}$. So we have x_{ti} fractional and constant for $t = 2, 3, 4$, and different from x_{1i} and x_{5i}. The idea is to say that we cannot have many objects like this (in a basic solution), by interpreting these objects on time steps $3, 4, 5$ as a basic optimal solution of a $3-DKP$ (locally, i.e. with a variable y_i such that $x_{1i} \leq y_i \leq x_{5i}$).

Then, roughly speaking, the idea is to show that for any pair of time steps $t_0 \leq t_1$, we can bound the number of objects which are fractional and constant on this time interval $[t_0, t_1]$ (but not at time $t_0 - 1$ and $t_1 + 1$). Then a sum on all the possible choices of (t_0, t_1) gives the global upper bound.

Let us state this rough idea formally. In all this section, we consider a (feasible) solution $\hat{S} = (\hat{x}, \hat{z})$ of $(LP-MK)$, where $\hat{z}_{ti} = 1 - |\hat{x}_{(t+1)i} - \hat{x}_{ti}|$ (variables $\hat{z}_{ti}$ are set to their optimal value w.r.t. $\hat{x}$).

In such a solution $\hat{S} = (\hat{x}, \hat{z})$, let us define as previously an object as *fractional* if at least one variable $\hat{x}_{ti}$ or $\hat{z}_{ti}$ is fractional. Our goal is to show the following result.

▶ **Theorem 6.** *If $\hat{S} = (\hat{x}, \hat{z})$ is a basic solution of $(LP - MK)$, it has at most T^3 fractional objects.*

Before proving the theorem, let us introduce some definitions and show some lemmas. Let t_0, t_1 be two time steps with $1 \leq t_0 \leq t_1 \leq T$.

▶ **Definition 7.** *The set $F(t_0, t_1)$ associated to $\hat{S} = (\hat{x}, \hat{z})$ is the set of objects i (called fractional w.r.t. (t_0, t_1)) such that*

- $0 < \hat{x}_{t_0 i} = \hat{x}_{(t_0+1)i} = \cdots = \hat{x}_{t_1 i} < 1$*;*
- *Either $t_0 = 1$ or $\hat{x}_{(t_0-1)i} \neq \hat{x}_{t_0 i}$;*
- *Either $t_1 = T$ or $\hat{x}_{(t_1+1)i} \neq \hat{x}_{t_1 i}$;*

In other words, we have $\hat{x}_{ti}$ fractional and constant on $[t_0, t_1]$, and $[t_0, t_1]$ is maximal w.r.t. this property.

For $t_0 \leq t \leq t_1$, we note C'_t the remaining capacity of knapsack at time t considering that variables outside $F(t_0, t_1)$ are fixed (to their value in $\hat{x}$):

$$C'_t = C_t - \sum_{i \notin F(t_0, t_1)} w_{ti} \hat{x}_{ti}.$$

As previously, we will see $x_{t_0 i}, \ldots, x_{t_1 i}$ as a single variable y_i. We have to express the fact that this variable y_i cannot "cross" the values $\hat{x}_{(t_0-1)i}$ (if $t_0 > 1$) and $\hat{x}_{(t_1+1)i}$ (if $t_1 < T$), so that everything remains locally (in this range) linear. So we define the lower and upper bounds a_i, b_i induced by Definition 7 as:

- Initialize $a_i \leftarrow 0$. If $\hat{x}_{(t_0-1)i} < \hat{x}_{t_0 i}$ then do $a_i \leftarrow \hat{x}_{(t_0-1)i}$. If $\hat{x}_{(t_1+1)i} < \hat{x}_{t_1 i}$ then do $a_i \leftarrow \max(a_i, \hat{x}_{(t_1+1)i})$.
- Similarly, initialize $b_i \leftarrow 1$. If $\hat{x}_{(t_0-1)i} > \hat{x}_{t_0 i}$ then do $b_i \leftarrow \hat{x}_{(t_0-1)i}$. If $\hat{x}_{(t_1+1)i} > \hat{x}_{t_1 i}$ then do $b_i \leftarrow \min(b_i, \hat{x}_{(t_1+1)i})$.

Note that with this definition $a_i < \hat{x}_{t_0,i} < b_i$. This allows us to define the polyhedron $P(t_0, t_1)$ as the set of $y = (y_i : i \in F(t_0, t_1))$ such that

$$\begin{cases} \sum\limits_{i \in F(t_0,t_1)} w_{ti} y_i & \leq & C'_t & \forall t \in \{t_0, ..., t_1\} \\ a_i \leq y_i \leq b_i & & & \forall i \in F(t_0, t_1) \end{cases}$$

▶ **Definition 8.** *The solution $\hat{y}$ associated to $\hat{S} = (\hat{x}, \hat{z})$ is defined as $\hat{y}_i = \hat{x}_{t_0 i}$ for $i \in F(t_0, t_1)$.*

▶ **Lemma 9.** *If $\hat{S} = (\hat{x}, \hat{z})$ is a basic solution, then the solution $\hat{y}$ associated to $(\hat{x}, \hat{z})$ is feasible of $P(t_0, t_1)$ and basic.*

Proof. Since $(\hat{x}, \hat{z})$ is feasible, then $\hat{y}$ respects the capacity constraints (remaining capacity), and $a_i < \hat{y}_i = \hat{x}_{t_0 i} < b_i$ so $\hat{y}$ is feasible.

Suppose now that $\hat{y} = \frac{y^1 + y^2}{2}$ for two feasible solutions $y^1 \neq y^2$ of $P(t_0, t_1)$. We associate to y^1 a feasible solution $S^1 = (x^1, z^1)$ as follows.

For any object i, we fix $x^1_{ti} = \hat{x}_i$ for $t \notin [t_0, t_1]$, and $x^1_{ti} = y^1_i$ for $t \in [t_0, t_1]$. We fix variables z^1_{it} to their maximal values, i.e. $z^1_{ti} = 1 - |x^1_{(t+1)i} - x^1_{ti}|$. This way, we get a feasible solution (x^1, z^1). Note that:

- $z^1_{ti} = \hat{z}_{ti}$ for $t \notin [t_0 - 1, t_1]$, since coresponding variables x are the same in S^1 and $\hat{S}$;
- $z^1_{ti} = 1 = \hat{z}_{ti}$ for $t \in [t_0, t_1 - 1]$, since variables x are constant on the interval $[t_0, t_1]$.

Then, for variables z, the only modifications between z^1 and $\hat{z}$ concern the "boundary" variables z^1_{ti} for $t = t_0 - 1$ and $t = t_1$.

We build this way two solutions $S^1 = (x^1, z^1)$ and $S^2 = (x^2, z^2)$ of $(LP - MK)$ corresponding to y^1 and y^2. By construction, S^1 and S^2 are feasible. They are also different provided that y^1 and y^2 are different. It remains to prove that $\hat{S} = (S^1 + S^2)/2$.

Let us first consider variables x:

- if $t \notin [t_0, t_1]$, $x^1_{ti} = x^2_{ti} = \hat{x}_{ti}$ so $\hat{x}_{ti} = \frac{x^1_{ti}+x^2_{ti}}{2}$.
- if $t \in [t_0, t_1]$, $x^1_{ti} = y^1_i$ and $x^2_{ti} = y^2_t$, so $\frac{x^1_{ti}+x^2_{ti}}{2} = \frac{y^1_i+y^2_i}{2} = \hat{y}_i = \hat{x}_{ti}$.

Now let us look at variables z: first, for $t \notin \{t_0 - 1, t_1\}$, $z^1_{ti} = z^2_{ti} = \hat{z}_{ti}$ so $\hat{z}_{ti} = \frac{z^1_{ti}+z^2_{ti}}{2}$. The last and main part concerns the last 2 variables $z_{(t_0-1)i}$ (if $t_0 > 1$) and $z_{t_1 i}$ (if $t_1 < T$).

We have $z^1_{(t_0-1)i} = 1 - |x^1_{t_0 i} - x^1_{(t_0-1)i}| = 1 - |x^1_{t_0 i} - \hat{x}_{(t_0-1)i}|$ and $\hat{z}_{(t_0-1)i} = 1 - |\hat{x}_{t_0 i} - \hat{x}_{(t_0-1)i}|$. The crucial point is to observe that thanks to the constraint $a_i \leq y_i \leq b_i$, and by definition of a_i and b_i, $x^1_{t_0,i}$, $x^2_{t_0,i}$ and $\hat{x}_{t_0,i}$ are either all greater than (or equal to) $\hat{x}_{(t_0-1)i}$, or all lower than (or equal to) $\hat{x}_{(t_0-1)i}$.

Suppose first that they are all greater than (or equal to) $\hat{x}_{(t_0-1)i}$. Then $z^1_{(t_0-1)i} - \hat{z}_{(t_0-1)i} = |\hat{x}_{t_0,i} - \hat{x}_{t_0-1,i}| - |x^1_{t_0,i} - \hat{x}_{t_0-1,i}| = \hat{x}_{t_0 i} - x^1_{t_0 i} = \hat{y}_i - y^1_i$.

Similarly, $z^2_{(t_0-1)i} - \hat{z}_{(t_0-1)i} = \hat{y}_i - y^2_i$. So $\frac{z^1_{(t_0-1)i}+z^2_{(t_0-1)i}}{2} = \frac{2\hat{z}_{(t_0-1)i}+2\hat{y}_i-y^1_i-y^2_i}{2} = \hat{z}_{(t_0-1)i}$.

Now suppose that they are all lower than (or equal to) $\hat{x}_{t_0-1,i}$. Then:

$$z^1_{(t_0-1)i} - \hat{z}_{(t_0-1)i} = |\hat{x}_{t_0 i} - \hat{x}_{(t_0-1)i}| - |x^1_{t_0 i} - \hat{x}_{(t_0-1)i}| = x^1_{t_0 i} - \hat{x}_{t_0 i} = y^1_i - \hat{y}_i$$

Similarly, $z^2_{(t_0-1)i} - \hat{z}_{(t_0-1)i} = y^2_i - \hat{y}_i$. So $\frac{z^1_{(t_0-1)i}+z^2_{(t_0-1)i}}{2} = \frac{2\hat{z}_{(t_0-1)i}-2\hat{y}_i+y^1_i+y^2_i}{2} = \hat{z}_{(t_0-1)i}$.

Then, in both cases, $\hat{z}_{(t_0-1)i} = \frac{z^1_{(t_0-1)i}+z^2_{(t_0-1)i}}{2}$.

With the very same arguments we can show that $\frac{z^1_{t_1 i}+z^2_{t_1 i}}{2} = \hat{z}_{t_1 i}$. Then, $\hat{S}$ is the half sum of S^1 and S^2, contradiction with the fact that $\hat{S}$ is basic. ◀

Now we can bound the number of fractional objects w.r.t. (t_0, t_1).

▶ **Lemma 10.** $|F(t_0, t_1)| \leq t_1 + 1 - t_0$.

Proof. $P(t_0, t_1)$ is a polyhedron corresponding to a linear relaxation of a $k - DLP$, with $k = t_1 + 1 - t_0$. Since $\hat{y}$ is basic, using Theorem 3 (and the note after) there are at most $k = t_1 + 1 - t_0$ variables $\hat{y}_i$ such that $a_i < \hat{y}_i < b_i$. But by definition of $F(t_0, t_1)$, for all $i \in F(t_0, t_1)$ $a_i < \hat{y}_i < b_i$. Then $|F(t_0, t_1)| \leq t_1 + 1 - t_0$. ◀

Now we can easily prove Theorem 6.

Proof. First note that if $\hat{x}_{ti}$ and $\hat{x}_{(t+1)i}$ are integral, then so is $\hat{z}_{ti}$. Then, if an object i is fractional at least one $\hat{x}_{ti}$ is fractional, and so i will appear in (at least) one set $F(t_0, t_1)$.

We consider all pairs (t_0, t_1) with $1 \leq t_0 \leq t_1 \leq T$. Thanks to Lemma 10, $|F(t_0, t_1)| \leq t_1 + 1 - t_0$. So, the total number of fractional objects is at most:

$$N_T = \sum_{t_0=1}^{T} \sum_{t_1=t_0}^{T} (t_1 + 1 - t_0) \leq T^3$$

Indeed, there are less than T^2 choices for (t_0, t_1) and at most T fractional objects for each choice. ◀

Note that with standard calculation we get $N_T = \frac{T^3+3T^2+2T}{6}$, so for $T = 2$ time steps $N_2 = 4$: we have at most 4 fractional objects, the same bound as in Proposition 5.

3.2 A PTAS for a constant number of time steps

Now we can describe the $PTAS$. Informally, the algorithm first guesses the ℓ objects with the maximum reward in an optimal solution (where ℓ is defined as a function of ϵ and T), and then finds a solution on the remaining instance using the relaxation of the LP. The fact that the number of fractional objects is small allows to bound the error made by the algorithm.

For a solution S (either fractional or integral) we define $g_i(S)$ as the reward of object i in solution S: $g_i(S) = \sum_{t=1}^{T} p_{ti}x_{ti} + \sum_{t=1}^{T-1} z_{ti}B_{ti}$. The value of a solution S is $g(S) = \sum_{i\in N} g_i(S)$.

Consider the algorithm A^{LP} which, on an instance of Multistage Knapsack:

- Finds an optimal (basic) solution $S^r = (x^r, z^r)$ of the relaxation $(LP - MK)$ of $(ILP - MK)$;
- Takes at step t an object i if and only if $x^r_{ti} = 1$.

Clearly, A^{LP} outputs a feasible solution, the value of which verifies:

$$g(A^{LP}) \geq g(S^r) - \sum_{i\in F} g_i(S^r) \tag{1}$$

where F is the set of fractional objects in S^r. Indeed, for each integral (i.e., not fractional) object the reward is the same in both solutions.

Now we can describe the algorithm **Algorithm $\boldsymbol{PTAS_{ConstantMK}}$**, which takes as input an instance of Multistage Knapsack and an $\epsilon > 0$, and works as follows.

1. Let $\ell := \min\left\{\left\lceil \frac{(T+1)T^3}{\epsilon}\right\rceil, n\right\}$.
2. For all $X \subseteq N$ such that $|X| = \ell$, $\forall X_1 \subseteq X, ..., \forall X_T \subseteq X$:
   If for all $t = 1, \ldots, T$ $w_t(X_t) = \sum_{j\in X_t} w_{tj} \leq C_t$, then:
   - Compute the rewards of object $i \in X$ in the solution $(X_1, \ldots, X_T)$, and find the smallest one, say k, with reward g_k.
   - On the subinstance of objects $Y = N \setminus X$:
     - For all $i \in Y$, for all $t \in \{1, \ldots, T\}$: if $p_{ti} > g_k$ then set $x_{ti} = 0$.
     - apply A^{LP} on the subinstance of objects Y, with the remaining capacity $C'_t = C_t - w_t(X_t)$, where some variables x_{ti} are set to 0 as explained in the previous step.
   - Let $(Y_1, ..., Y_T)$ be the sets of objects taken at time $1, \ldots, T$ by A^{LP}. Consider the solution $(X_1 \cup Y_1, ..., X_T \cup Y_T)$.
3. Output the best solution computed.

▶ **Theorem 11.** *The algorithm $PTAS_{ConstantMK}$ is a $(1-\epsilon)$-approximation algorithm running in time $O\left(n^{O(T^5/\epsilon)}\right)$.*

Proof. (sketch) Let us briefly argue why the claimed ratio holds. We consider the iteration of the algorithm where X equals the set of ℓ objects which have maximum reward in an optimal solution S^*. By exhaustive search, the algorithm considers the case where $X_1, \ldots, X_T$ are exactly as in S^*, so the reward of the algorithm is the same as S^* for these objects. For the remaining objects, we use the relaxation of the linear program to build an integer solution. The loss corresponds to the rewards generated by fractional objects (see Equation 1) in the *fractional solution.*

While these profits could be very high, the preprocessing step fixing some variables with high profit to 0 (x_{ti} is set to 0 if $p_{ti} > g_k$) allows to bound the loss as (roughly) the reward of these fractional objects *in* S^*. Since there is a bounded number of fractional objects, the loss induced by their rewards can be bounded by a fraction ϵ of the optimal value, by choosing a sufficiently large $\ell = |X|$. ◀

3.3 Generalization to an arbitrary number of time steps

We now devise a PTAS for the general problem, for an arbitrary (not constant) number of steps. We actually show how to get such a PTAS provided that we have a PTAS for (any) constant number of time steps. Let A_{ϵ,T_0} be an algorithm which, given an instance of Multistage Knapsack with at most T_0 time steps, outputs a $(1-\epsilon)$-approximate solution in time $O(n^{f(\epsilon,T_0)})$ for some function f.

The underlying idea is to compute (nearly) optimal solutions on subinstances of bounded sizes, and then to combine them in such a way that at most a small fraction of the optimal value is lost.

Let us first give a rough idea of our algorithm $PTAS_{MK}$.

Given an $\epsilon > 0$, let $\epsilon' = \epsilon/2$ and $T_0 = \lceil \frac{1}{\epsilon'} \rceil$. We construct a set of solutions $S^1, \dots, S^{T_0}$ in the following way:

In order to construct S^1, we partition the time horizon $1, \dots, T$ into $\lceil \frac{T}{T_0} \rceil$ consecutive intervals. Every such interval has length T_0, except possibly the last interval that may have a smaller length. We apply A_{ϵ,T_0} at every interval in this partition. S^1 is then just the concatenation of the partial solutions computed for each interval.

The partition used to build the solution S^i, $1 < i \leq T_0$, is made in a similar way. The only difference is that the first interval of the partition of the time horizon $1, \dots, T$ goes from time 1 to time $i-1$. For the remaining part of the time horizon, i.e. for $i, \dots T$, the partition is made as previously, i.e. starting at time step i, every interval will have a length of T_0, except possibly the last one, whose length may be smaller. Once the partition is operated, we apply A_{ϵ,T_0} to every interval of the partition. S^i, $1 < i \leq T_0$, is then defined as the concatenation of the partial solutions computed on each interval. Among the T_0 solutions $S^1, \dots, S^{T_0}$, the algorithm chooses the best solution.

The construction is illustrated on Figure 1, with 10 time steps and $T_0 = 3$. The first solution S^1 is built by applying 4 times A_{ϵ,T_0}, on the subinstances corresponding to time steps $\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 9\}$, and $\{10\}$. The solution S^2 is built by applying 4 times A_{ϵ,T_0}, on the subinstances corresponding to time steps $\{1\}, \{2, 3, 4\}, \{5, 6, 7\}$, and $\{8, 9, 10\}$.

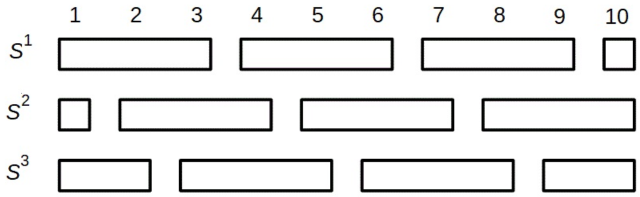


Figure 1 The three solutions for $T_0 = 3$ and $T = 10$.

▶ **Theorem 12.** *The algorithm $PTAS_{MK}$ is a polynomial time approximation algorithm.*

Let us give a rough idea of the proof. As we see from Figure 1, each solution S^t misses some potential transition profit between some time steps (as between 3 and 4, between 6 and 7, and between 9 and 10 for S^1). For each j, such loss between step j and $j+1$ appears in exactly one S^t, so in average we loose a fraction $1/T_0$ of the optimal transition profit (so a fraction at most ϵ'). Another loss is due to the fact that we use an approximation algorithm on the subinstances, inducing also a loss of at most a fraction ϵ' of the optimum value.

4 Pseudo-polynomiality and hardness results

We complement the previous result on approximation scheme by showing the following results for Multistage Knapsack. First, it does not admit an FPTAS (unless $P = NP$), as stated in the following Theorem.

▶ **Theorem 13.** *There is no FPTAS for* Multistage Knapsack *unless $P = NP$, even if there are only two time steps and the bonus is uniform.*

The reduction is from Cardinality-2-KP, and the idea of the proof is to take a sufficiently large (but polynomially bounded) bonus in the multistage instance in order to force the knapsacks of the two time steps to be the same.

The second result states that the problem is pseudo-polynomial for a constant number of time steps. More precisely, with a standard dynamic programming procedure, we have the following.

▶ **Theorem 14.** Multistage Knapsack *is solvable in time $O(T(2C_{max} + 2)^T n)$ where $C_{max} = \max\{C_i, i = 1, \dots, T\}$.*

As a final result, we show that the problem is strongly NP-hard (when the number of steps is not bounded), by showing the NP-hardness of the following subproblem.

▶ **Definition 15.** Binary Multistage Knapsack *is the sub-problem of* Multistage Knapsack *where all the weights, profits and capacities are all equal to 0 or* 1.

For the usual Knapsack problem, the binary case corresponds to a trivial problem. For the multistage case, we have the following:

▶ **Theorem 16.** Binary Multistage Knapsack *is NP-hard, even in the case of uniform bonus.*

Proof. (sketch) We prove the result by a reduction from the Independent Set problem where, given a graph G and an integer K, we are asked if there exists a subset of K pairwise non adjacent vertices (called an independent set). This problem is NP-hard, see [13].

Let (G, K) be an instance of the Independent Set problem, with $G = (V, E)$, $V = \{v_1, \dots, v_n\}$ and $E = \{e_1, \dots, e_m\}$. We build the following instance I' of Binary Multistage Knapsack:

- There are n objects $\{1, 2 \dots, n\}$, one object per vertex;
- There are $T = m$ time steps: each edge (v_i, v_j) in E corresponds to one time step;
- at the time step corresponding to edge (v_i, v_j): objects i and j have weight 1, while the others have weight 0, all objects have profit 1, and the capacity constraint is 1.
- The transition profit is $b_{ti} = B = 2nm$ for all i, t.

Roughly speaking, B is large enough to ensure that in an optimal solution there are no modifications of the knapsack over the time. Then we can show that there is an independent set of size (at least) K if and only if there is a solution for Binary Multistage Knapsack of value (at least) $n(m - 1)B + mK$. ◀

Since B is polynomially bounded in the proof, Multistage Knapsack is strongly NP-hard.

References

1 Susanne Albers. On Energy Conservation in Data Centers. In Christian Scheideler and Mohammad Taghi Hajiaghayi, editors, *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*, pages 35–44. ACM, 2017. doi:10.1145/3087556.

2 Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson. Dynamic Facility Location via Exponential Clocks. *ACM Trans. Algorithms*, 13(2):21:1–21:20, 2017.

3 Barbara M. Anthony and Anupam Gupta. Infrastructure Leasing Problems. In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization, 12th International IPCO Conference, Ithaca, NY, USA, June 25-27, 2007, Proceedings*, volume 4513 of *Lecture Notes in Computer Science*, pages 424–438. Springer, 2007. doi:10.1007/978-3-540-72792-7.

4 Evripidis Bampis, Bruno Escoffier, Michael Lampis, and Vangelis Th. Paschos. Multistage Matchings. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018, June 18-20, 2018, Malmö, Sweden*, volume 101 of *LIPIcs*, pages 7:1–7:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.SWAT.2018.7.

5 Evripidis Bampis, Bruno Escoffier, and Sasa Mladenovic. Fair Resource Allocation Over Time. In Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar, editors, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 766–773. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018. URL: http://dl.acm.org/citation.cfm?id=3237496.

6 Nicolas K. Blanchard and Nicolas Schabanel. Dynamic Sum-Radii Clustering. In Sheung-Hung Poon, Md. Saidur Rahman, and Hsu-Chun Yen, editors, *WALCOM: Algorithms and Computation, 11th International Conference and Workshops, WALCOM 2017, Hsinchu, Taiwan, March 29-31, 2017, Proceedings.*, volume 10167 of *Lecture Notes in Computer Science*, pages 30–41. Springer, 2017. doi:10.1007/978-3-319-53925-6_3.

7 Niv Buchbinder, Shahar Chen, and Joseph Naor. Competitive Analysis via Regularization. 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 436–444. SIAM, 2014. doi:10.1137/1.9781611973402.32.

8 Niv Buchbinder, Shahar Chen, Joseph Naor, and Ohad Shamir. Unified Algorithms for Online Learning and Competitive Analysis. *Math. Oper. Res.*, 41(2):612–625, 2016. doi:10.1287/moor.2015.0742.

9 Alberto Caprara, Hans Kellerer, Ulrich Pferschy, and David Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123(2):333–345, 2000.

10 Edith Cohen, Graham Cormode, Nick G. Duffield, and Carsten Lund. On the Tradeoff between Stability and Fit. *ACM Trans. Algorithms*, 13(1):7:1–7:24, 2016. doi:10.1145/2963103.

11 David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility Location in Evolving Metrics. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2014. doi:10.1007/978-3-662-43951-7_39.

12 Alan M Frieze, MRB Clarke, et al. Approximation algorithms for the m-dimensional 0-1 knapsack problem: Worst-case and probabilistic analyses. *European Journal of Operational Research*, 15:100–109, 1984.

13 Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

14 Albert Gu, Anupam Gupta, and Amit Kumar. The Power of Deferral: Maintaining a Constant-Competitive Steiner Tree Online. *SIAM J. Comput.*, 45(1):1–28, 2016. doi:10.1137/140955276.

15 Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing Bases: Multistage Optimization for Matroids and Matchings. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 563–575. Springer, 2014.

16 Oscar H. Ibarra and Chul E. Kim. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *J. ACM*, 22(4):463–468, 1975. doi:10.1145/321906.321909.

17 Hans Kellerer and Ulrich Pferschy. A New Fully Polynomial Time Approximation Scheme for the Knapsack Problem. *J. Comb. Optim.*, 3(1):59–71, 1999. doi:10.1023/A:1009813105532.

18 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems.* Springer, Berlin, Germany, 2004.

19 Bernhard Korte and Rainer Schrader. On the existence of fast approximation schemes. In O. Magasarian, R. Meyer, and S. Robinson, editors, *Nonlinear Programming 4*, pages 415–437. Academic Press, 1981.

20 Eugene L. Lawler. Fast Approximation Algorithms for Knapsack Problems. *Math. Oper. Res.*, 4(4):339–356, 1979. doi:10.1287/moor.4.4.339.

21 Michael J. Magazine and Osman Oguz. A fully polynomial approximation scheme for the 0-1 knapsack problem. *European Journal of Operational Research*, 8:270–273, 1981.

22 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The Power of Recourse for Online MST and TSP. *SIAM J. Comput.*, 45(3):859–880, 2016. doi:10.1137/130917703.

23 Chandrashekhar Nagarajan and David P Williamson. Offline and online facility leasing. *Discrete Optimization*, 10(4):361–370, 2013.

24 Osman Oguz and Michael J. Magazine. A polynomial time approximation algorithm for the multidimensional knapsack problem. *Working paper, University of Waterloo*, 1980.

25 Cécile Rottner. *Combinatorial Aspects of the Unit Commitment Problem.* PhD thesis, Sorbonne Université, 2018.

26 Baruch Schieber, Hadas Shachnai, Gal Tamir, and Tami Tamir. A Theory and Algorithms for Combinatorial Reoptimization. *Algorithmica*, 80(2):576–607, 2018. doi:10.1007/s00453-017-0274-8.

Recursion Schemes, Discrete Differential Equations and Characterization of Polynomial Time Computations

Olivier Bournez
Laboratoire d'Informatique de l'X (LIX), CNRS, Ecole Polytechnique,
Institut Polytechnique de Paris, 91128 Palaiseau Cedex, France
bournez@lix.polytechnique.fr

Arnaud Durand
Université Paris Diderot, IMJ-PRG, CNRS UMR 7586, Case 7012, 75205 Paris cedex 13, France
durand@math.univ-paris-diderot.fr

Abstract

This paper studies the expressive and computational power of discrete Ordinary Differential Equations (ODEs). It presents a new framework using discrete ODEs as a central tool for computation and algorithm design. We present the general theory of discrete ODEs for computation theory, we illustrate this with various examples of algorithms, and we provide several implicit characterizations of complexity and computability classes.

The proposed framework presents an original point of view on complexity and computation classes. It unifies several constructions that have been proposed for characterizing these classes including classical approaches in implicit complexity using restricted recursion schemes, as well as recent characterizations of computability and complexity by classes of continuous ordinary differential equations. It also helps understanding the relationships between analog computations and classical discrete models of computation theory.

At a more technical point of view, this paper points out the fundamental role of linear (discrete) ordinary differential equations and classical ODE tools such as changes of variables to capture computability and complexity measures, or as a tool for programming many algorithms.

2012 ACM Subject Classification Theory of computation; Theory of computation → Models of computation; Theory of computation → Computability; Theory of computation → Recursive functions; Computer systems organization → Analog computers; Theory of computation → Complexity classes; Theory of computation → Complexity theory and logic; Mathematics of computing → Differential equations; Mathematics of computing → Ordinary differential equations; Mathematics of computing → Differential calculus

Keywords and phrases Implicit complexity, discrete ordinary differential equations, recursion scheme

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.23

Funding *Olivier Bournez*: Supported by RACAF Project from Agence National de la Recherche and Labex Digicosme Project ACDC.

Acknowledgements We would like to thank Sabrina Ouazzani for many scientific discussions about the results in this article.

1 Introduction

Since the beginning of its foundations, classification of the difficulty of problems, with various models of computation, either by their complexity or by their computability properties, is a thriving field of computer science. Nowadays, classical computer science problems also deal with continuous data coming from different areas and modeling involves the use of tools like numerical analysis, probability theory or differential equations. Thus new characterizations

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 23; pp. 23:1–23:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

related to theses fields have been proposed. On a dual way, the quest for new types of computers recently led to revisit the power of some models for analog machines based on differential equations, and to compare them to modern digital models. In both contexts, when discussing the related computability or complexity issues, one has to overcome the fact that today's (digital) computers are in essence discrete machines while the objects under study are continuous and naturally correspond to Ordinary Differential Equations (ODEs).

We consider here an original approach in between the two worlds: discrete oriented computation with differential equations.

ODEs appear to be a natural way of expressing properties and are intensively used, in particular in applied science. The theory of classical (continuous) ODEs has an abundant literature (see e.g. [1, 3, 9]) and is rather well understood under many aspects. We are interested here in a discrete counterpart of classical continuous ODEs: discrete ODEs. Its associated derivative notion, called *finite differences*, has been widely studied in numerical optimization for function approximation [12] and is reminiscent in *discrete calculus* [14, 13, 15, 20] for combinatorial analysis (remark that similarities between discrete and continuous statements have also been historically observed, under the terminology of *umbral* or *symbolic calculus* as early as in the 19th century). However, even if the underlying computational content of finite differences theory is clear and has been pointed out many times, no fundamental connections with algorithms and complexity have been exhibited so far.

In this article, our goal is to demonstrate that discrete ODEs is a very natural tool for algorithm design and to prove that complexity and computability notions can be elegantly and simply captured using discrete ordinary differential equations. We illustrate this by providing a characterization of **FPTIME**, the class of polynomial time computable functions, and of its non deterministic analog **FNP**. To this aim, we will also demonstrate how some notions from the analog world such as linearity of differential equations or derivatives along some particular functions (i.e. changes of variables) are representative of a certain computational hardness and can be used to solve efficiently some (classical, digital) problems.

As far as we know, this is the first time that computations with discrete ODEs and their related complexity aspects are considered. By contrast, complexity results have been recently obtained about classical (continuous) ODEs for various classes of functions, mostly in the framework of computable analysis. The hardness of solving continuous ODEs has been intensively discussed: for example [19] establishes some bases of the complexity aspects of ODEs and more recent works like [18] or [10] establish links between complexity or effective aspects of such differential equations. We believe that investigating the expressive power of discrete ODE, can help to better understand complexity of computation for both the discrete and continuous settings. Indeed, on the one hand, our results offers a new machine independent perspective on classical discrete computations, i.e. computations that deal with bits, words, or integers. And, on the other hand, it relates classical (discrete) complexity classes to analog computations, i.e. computations over the reals, as analog computation have been related in various ways to continuous ordinary differential equations, and as discrete ordinary differential equations provide clear hints about their continuous counterparts. A mid-term goal of this line of research is also to bring insights from complexity theory to the problem of solving ODE (discrete and, hopefully also, numerical). A descriptive approach such as the one initiated in the paper could help classifying large classes of ODE by their computational hardness and bring some uniformity to methods of this field.

From restricted recursion scheme to discrete differential equations

Recursion schemes constitutes a major approach of computability theory and to some extent of complexity theory. A foundational result in that spirit is due to Cobham, who gave in [8] a characterization of functions computable in polynomial time through the notion of *bounded recursion on notations* (BRN, for short). A function f is defined by BRN from g, h_0, h_1, k if $f(0, \mathbf{y}) = g(\mathbf{y})$; $f(\mathbf{0}(x), \mathbf{y}) = h_0(f(x, \mathbf{y}), x, \mathbf{y})$ for $x \neq 0$; $f(\mathbf{1}(x), \mathbf{y}) = h_1(f(x, \mathbf{y}), x, \mathbf{y})$ with, for all $x, \mathbf{y}$: $f(x, \mathbf{y}) \leq k(x, \mathbf{y})$. Here $\mathbf{0}(.)$ and $\mathbf{1}(.)$ denote the successor functions defined by $\mathbf{0}(x) = 2.x$ and $\mathbf{1}(x) = 2.x + 1$. In this approach, the number of steps is controlled through the use of the alternative successors $\mathbf{0}(.)$ and $\mathbf{1}(.)$ (which increase the size of their argument by one in each round) and the size of objects through an explicit upper bound by a given function k. Later, notions such as safe recursion [2] or ramification ([22, 21] have allowed syntactical characterizations of polynomial time or other classes [23] that do not require the use of an explicit bound but at the expense of rather sophisticated schemes. These works have therefore been at the origin of the very vivid field of *implicit complexity* at the interplay of logic and theory of programming.

A discrete function can also be described by its derivative i.e. the value $\mathbf{f}(x+1, \mathbf{y}) - \mathbf{f}(x, \mathbf{y})$. It is straightforward to rewrite primitive recursion in this setting, though one may have to cope with possibly negative values. To capture more fine grained time and space measures we come up in the proposed context of discrete ODEs with at least two original concepts which are very natural in the continuous setting:

- deriving along a function; when such a function is suitably chosen this allows to control the number of steps in the computation;
- linearity that permits to control object sizes.

By combining these two approaches, we provide a characterization of **FPTIME** that does not require to specify an explicit bound in the recursion, in contrast to Cobham's work, nor to assign a specific role or type to variables, in contrast to safe recursion or ramification. The characterization happens to be very simple from a syntactical point of view using only natural notions from the world of ODE.

This characterization is also a first step to convince the reader that deep connections between complexity and ODE solving do exist and that these connections are worth to be further studied.

Related works on analog computations

As many historical or even possibly futuristic analog machines are naturally described by (continuous) ODEs, the quest of understanding how the computational power of analog models compare to classical digital ones have led to several results relating classical complexity to various classes of continuous ODEs. In particular, a series of papers has been devoted to study various classes of the so-called $\mathbb{R}$-recursive functions, after their introduction in [25] as a theoretical model for computations over the reals. At the complexity level, characterizations of complexity classes such as **PTIME** and **NPTIME** using $\mathbb{R}$-recursive algebra have been obtained [27], motivated in particular by the idea of transferring classical questions from complexity theory to the context of real and complex analysis [24, 27, 26]. But this has been done with the addition of limit schemata and with a rather different settings.

More recently, revisiting the model of General Purpose Analog Computer of Claude Shannon, it is has been proved that polynomial differential equations can be considered as a very simple and elegant model in which computable functions over the reals and polynomial time computable functions over the reals can be defined without any reference to concepts from discrete computation theory [4, 29]. We believe the current work is a substantial step to understand the underlying power and theory of such analog models, by providing concepts, definitions and results relating the two worlds.

Refer to [5] for an up to date survey about various works on analog computations, in particular in a computation theory perspective.

Structure of the paper

In Section 2 a short introduction to discrete differentiability is given followed in Section 3 by an illustration, through examples, of the programming ability of discrete ODE. Formal definitions of discrete ODE schemas are given in Section 4 together with characterizations of classical computability classes. Our objective, from Section 2 to 4, is to expose in a simple way (using sometimes well-known concepts) the basics of the theory of discrete ODE and its computational content, before getting to more technical results. Section 5 introduces the notion of length-ODE which is central, together with the that of (essentially) linear differential equation, for the characterization of **FPTIME** (Section 6) and **FNP** (Sections 7). In Section 8 we discuss some extensions of the results.

2 Discrete differentiability and ODEs

In this section, we review some basic notions of discrete calculus to help intuition in the rest of the paper (refer to [16, 12] for a more complete review). Discrete derivatives are usually intended to concern functions over the integers of type $\mathbf{f}: \mathbb{N}^p \rightarrow \mathbb{Z}^d$, but the statements and concepts considered in our discussions are also valid more generally for functions of type $\mathbf{f}: \mathbb{Z}^p \rightarrow \mathbb{Z}^d$, for some integers p, d, or even functions $\mathbf{f}: \mathbb{R}^p \rightarrow \mathbb{R}^d$. The basic idea is to consider the following concept of derivative

▶ Remark 1. We first discuss the case where $p = 1$, i.e. functions $\mathbf{f}: \mathbb{N} \rightarrow \mathbb{Z}^d$. We will later on consider more general functions, with partial derivatives instead of derivatives.

▶ **Definition 2** (Discrete Derivative). *The discrete derivative of $\mathbf{f}(x)$ is defined as $\Delta\mathbf{f}(x) = \mathbf{f}(x+1) - \mathbf{f}(x)$. We will also write $\mathbf{f}'$ for $\Delta\mathbf{f}(x)$ to help to understand statements with respect to their classical continuous counterparts.*

Several results from classical derivatives generalize to this settings: this includes linearity of derivation $(a \cdot f(x) + b \cdot g(x))' = a \cdot f'(x) + b \cdot g'(x)$, formulas for products and division such as $(f(x) \cdot g(x))' = f'(x) \cdot g(x+1) + f(x) \cdot g'(x) = f(x+1)g'(x) + f'(x)g(x)$.

A fundamental concept is the following:

▶ **Definition 3** (Discrete Integral). *Given some function $\mathbf{f}(x)$, we write $\int_a^b \mathbf{f}(x)\delta x$ as a synonym for $\int_a^b \mathbf{f}(x)\delta x = \sum_{x=a}^{x=b-1} \mathbf{f}(x)$ with the convention that it takes value 0 when $a = b$ and $\int_a^b \mathbf{f}(x)\delta x = -\int_b^a \mathbf{f}(x)\delta x$ when $a > b$.*

The telescope formula yields the so-called Fundamental Theorem of Finite Calculus:

▶ **Theorem 4** (Fundamental Theorem of Finite Calculus). *Let $\mathbf{F}(x)$ be some function. Then, $\int_a^b \mathbf{F}'(x)\delta x = \mathbf{F}(b) - \mathbf{F}(a)$.*

As for classical functions, a given function has several primitives. These primitives are defined up to some additive constant. Several techniques from the classical settings generalize to the discrete settings: this includes the technique of integration by parts.

A classical concept in discrete calculus is the one of falling power defined as $x^{\underline{m}} = x \cdot (x-1) \cdot (x-2) \cdots (x-(m-1))$. This notion is motivated by the fact that it satisfies a derivative formula $(x^{\underline{m}})' = m \cdot x^{\underline{m-1}}$ similar to the classical one for powers in the continuous setting. In a similar spirit, we introduce the concept of falling exponential.

▶ **Definition 5** (Falling exponential). *Given some function $\mathbf{U}(x)$, the expression $\mathbf{U}$ to the falling exponential x, denoted by $\overline{2}^{\mathbf{U}(x)}$, stands for $\overline{2}^{\mathbf{U}(x)} = (1 + \mathbf{U}'(x-1)) \cdots (1 + \mathbf{U}'(1)) \cdot (1 + \mathbf{U}'(0)) = \prod_{t=0}^{t=x-1}(1 + \mathbf{U}'(t))$, with the convention that $\prod_0^0 = \mathbf{id}$, where $\mathbf{id}$ is the identity (sometimes denoted 1 hereafter)*

This is motivated by the remarks that $2^x = \overline{2}^x$, and that the discrete derivative of a falling exponential is given by $\left(\overline{2}^{\mathbf{U}(x)}\right)' = \mathbf{U}'(x) \cdot \overline{2}^{\mathbf{U}(x)}$ for all $x \in \mathbb{Z}$.

We will focus in this article on discrete Ordinary Differential Equations (ODE) on functions with several variables, that is to say for example on equations of the (possibly vectorial) form:

$$\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial x} = \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}), \tag{1}$$

where $\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial x}$ stands as expected for the derivative of functions $\mathbf{f}(x, \mathbf{y})$ considered as a function of x, when $\mathbf{y}$ is fixed i.e. $\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial x} = \mathbf{f}(x+1, \mathbf{y}) - \mathbf{f}(x, \mathbf{y})$. When some initial value $\mathbf{f}(0, \mathbf{y}) = \mathbf{g}(\mathbf{y})$ is added, this is called an *Initial Value Problem (IVP)* or a *Cauchy Problem*. An IVP can always be put in integral form

$$\mathbf{f}(x, \mathbf{y}) = \mathbf{f}(0, \mathbf{y}) + \int_0^x \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}) \delta x.$$

Our aim here is to discuss total functions whose domain and range is either of the form $\mathcal{D} = \mathbb{N}$, $\mathbb{Z}$, or possibly a finite product $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_k$ where each $\mathcal{D}_i = \mathbb{N}$, $\mathbb{Z}$. By considering that $\mathbb{N} \subset \mathbb{Z}$, we assume that the range is always $\mathbb{Z}^d$ for some d. The concept of solution for such ODEs is as expected: given $h : \mathbb{Z}^d \times \mathbb{N} \times \mathbb{Z}^p \to \mathbb{Z}$ (or $h : \mathbb{Z}^d \times \mathbb{Z} \times \mathbb{Z}^p \to \mathbb{Z}$), a solution over $\mathcal{D}$ is a function $f : \mathcal{D} \times \mathbb{Z}^p \to \mathbb{Z}^d$ that satisfies the equations for all $x, \mathbf{y}$.

We will only consider well-defined ODEs such as above in this article (but variants with partially defined functions could be considered as well). Observe that an IVP of the form (1) always admits a (necessarily unique) solution over $\mathbb{N}$ since f can be defined inductively with $\mathbf{f}(0, \mathbf{y}) = \mathbf{g}(\mathbf{y})$ and $\mathbf{f}(x+1, \mathbf{y}) = \mathbf{f}(x, \mathbf{y}) + \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y})$.

▶ Remark 6. Notice that this is not necessarily true over $\mathbb{Z}$: As an example, consider $f'(x) = -f(x) + 1$, $f(0) = 0$. By definition of $f'(x)$, we must have $f(x+1) = 1$ for all x, but if $x = -1$, $f(0) = 1 \neq 0$.

▶ Remark 7 (Sign function). It is very instructive to realize that the solution of this IVP over $\mathbb{N}$ is the sign $\mathsf{sg}_{\mathbb{N}}(x)$ function defined by $\mathsf{sg}_{\mathbb{N}}(x) = 1$ if $x > 0$ and $\mathsf{sg}_{\mathbb{N}}(x) = 0$ in the other case.

Linear (also called *Affine*) ODEs will play a very important role in what follows, i.e. discrete ordinary differential equations of the form $\mathbf{f}'(x) = \mathbf{A}(x) \cdot \mathbf{f}(x) + \mathbf{B}(x)$.

▶ Remark 8. Recall that the solution of $f'(x) = a(x)f(x) + b(x)$ for classical continous derivatives turns out to be given by (usually obtained using the method of variation of parameters): $f(x) = f(0)e^{\int_0^x a(t)dt} + \int_0^x b(u)e^{\int_u^x a(t)dt} du$. This generalizes with our definitions to discrete ordinary differential equations, and this works even vectorially:

▶ **Lemma 9** (Solution of linear ODE). *For matrices $\mathbf{A}$ and vectors $\mathbf{B}$ and $\mathbf{G}$, the solution of equation $\mathbf{f}'(x, \mathbf{y}) = \mathbf{A}(x, \mathbf{y}) \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}(x, \mathbf{y})$ with initial conditions $\mathbf{f}(0, \mathbf{y}) = \mathbf{G}(\mathbf{y})$ is $\mathbf{f}(x, \mathbf{y}) = \left(\overline{2}^{\int_0^x \mathbf{A}(t,\mathbf{y})\delta t}\right) \cdot \mathbf{G}(\mathbf{y}) + \int_0^x \left(\overline{2}^{\int_{u+1}^x \mathbf{A}(t,\mathbf{y})\delta t}\right) \cdot \mathbf{B}(u, \mathbf{y}) \delta u$.*

▶ Remark 10. Notice that this can be rewritten as $\sum_{u=-1}^{x-1}\left(\prod_{t=u+1}^{x-1}(1+\mathbf{A}(t,\mathbf{y}))\right)\cdot\mathbf{B}(u,\mathbf{y})$ with the conventions that for any function $\kappa(\cdot)$, $\prod_{x}^{x-1}\kappa(x)=1$ and $\mathbf{B}(-1,\mathbf{y})=\mathbf{G}(\mathbf{y})$. Such equivalent expressions both have a clear computational content. They can be interpreted as an algorithm unrolling the computation of $\mathbf{f}(x+1,\mathbf{y})$ from the computation of $\mathbf{f}(x,\mathbf{y}),\mathbf{f}(x-1,\mathbf{y}),\dots,\mathbf{f}(0,\mathbf{y})$. The next section will build on that remark through some examples.

3 Programming with discrete ODE

The computational dimension of calculus of finite differences has been widely stressed in mathematical analysis. However, no fundamental connection has been established with algorithmic and complexity. In this section, we show that several algorithms can actually be naturally expressed as discrete ODEs.

For now, we suppose that composition of functions, constant and the following basic functions can be used freely as functions from $\mathbb{Z}$ to $\mathbb{Z}$: arithmetic operations: $+$, $-$, $\times$; $\ell(x)$ returns the length of $|x|$ written in binary; $\mathsf{sg}(x):\mathbb{Z}\to\mathbb{Z}$ (respectively: $\mathsf{sg}_{\mathbb{N}}(x):\mathbb{N}\to\mathbb{Z}$) that takes value 1 for $x>0$ and 0 in the other case; From these basic functions, for readability, one may define useful functions as synonyms: $\bar{\mathsf{sg}}(x)$ stands for $\bar{\mathsf{sg}}(x)=(1-\mathsf{sg}(x))\times(1-\mathsf{sg}(-x))$: it tests if $x=0$ for $x\in\mathbb{Z}$; $\bar{\mathsf{sg}}_{\mathbb{N}}(x)$ stands for $\bar{\mathsf{sg}}_{\mathbb{N}}(x)=1-\mathsf{sg}_{\mathbb{N}}(x)$: it tests if $x=0$ for $x\in\mathbb{N}$. $\mathsf{if}(x,y,z)$ stands for $\mathsf{if}(x,y,z)=y+\bar{\mathsf{sg}}(x)\cdot(z-y)$ and $\mathsf{if}_{\mathbb{N}}(x,y,z)$ stands for $\mathsf{if}_{\mathbb{N}}(x,y,z)=y+\bar{\mathsf{sg}}_{\mathbb{N}}(x)\cdot(z-y)$. $\mathsf{if}(x<x',y,z)$ will be a synonym for $\mathsf{if}(\mathsf{sg}(x'-x+1),y,z)$ and $\mathsf{if}(x=x',y,z)$ will be a synonym for $\mathsf{if}(1-\bar{\mathsf{sg}}(x-x'),y,z)$.

First observe that discrete ODEs allow to express easily search functions:

▶ **Example 11** (Computing the minimum of a function)**.** The minimum of a function $\min f: x\mapsto min\{f(y):0\leq y\leq x\}$ is given by $F(x,x)$ where F is solution of the discrete ODE $F(0,x)=f(0)$; $\frac{\partial F(t,x)}{\partial t}=H(F(t,x),f(x),t,x)$, where $H(F,f,t,x)=0$ if $F<f$, $f-F$ if $F\geq f$. In integral form, we have: $F(x,y)=F(0)+\int_0^x H(F(t,y),t,y)\delta t$.

Conversely such an integral above (equivalently discrete ODE) can always be considered as a (recursive) algorithm: compute the integral from its definition as a sum. On this example, this corresponds basically to compute $F(x,x)$ recursively by $F(t+1,x)=\mathsf{if}(F(t,x)<f(x),F(t,x),f(x))$.

▶ Remark 12. Note that this algorithm is not polynomial in the length of its argument x, as it takes time x to compute $\min f$. Getting to polynomial algorithms will be at the heart of coming discussions in the two next examples and in Section 5.

As shown in the following two examples, discrete ODEs can express more sophisticated functions and turn out to be very natural in many other contexts, in particular non numerical ones, where they would probably not be expected.

▶ **Example 13** (Computing the integer part and divisions, going to Length-ODE)**.** Suppose that we want to compute $\lfloor\sqrt{x}\rfloor=\max\{y\leq x:y\cdot y\leq x\}$ and $\left\lfloor\frac{x}{y}\right\rfloor=\max\{z\leq x:z\cdot y\leq x\}$. It can be done by the following general method. Let f,h be some functions with h being non decreasing. We compute some_h with $\mathsf{some}_h(x)=y$ s.t. $|f(x)-h(y)|$ is minimal. When $h(x)=x^2$ and $f(x)=x$, it holds that: $\lfloor\sqrt{x}\rfloor=\mathsf{if}(\mathsf{some}_h(x)^2\leq x,\mathsf{some}_h(x),\mathsf{some}_h(x)-1)$. The function some_h can be computed (in non-polynomial time) as a solution of an ODE as in Example 11. However, there is a more efficient (polynomial time) way to do it based on what one usually does with classical ordinary differential equations: performing a change of variable so that the search becomes logarithmic in x. Indeed, we can write $\mathsf{some}_h(x)=G(\ell(x),x)$ for some function $G(t,x)$ that is a solution of $G(0,x)=x$; $\frac{\partial G(t,x)}{\partial t}=E(G(t,x),t,x)$ where $E(G,t,x)$ takes value $2^{\ell(x)-t-1}$ whenever $h(G>f(x))$, 0 whenever $h(G)=f(x)$ and $-2^{\ell(x)-t-1}$ whenever $h(G)<f(x)$.

▶ **Example 14** (Computing suffixes with discrete ODEs)**.** The suffix function, $\mathsf{suffix}(x, y)$ takes as input two integers x and y and outputs the $\ell(y) = t$ least significant bits of the binary decomposition of x. We describe below a way to compute a suffix working over a parameter t, that is logarithmic in x. Consider the following unusual algorithm that can be interpreted as a fix-point definition of the function: $\mathsf{suffix}(x, y) = F(\ell(x), y)$ where $F(0, x) = x$; $F(t + 1, x) = \mathsf{if}(\ell(F(t, x)) = 1, F(t, x), F(t, x) - 2^{\ell(F(t,x))-1})$.This can be interpreted as a differential equation, whose solution is converging fast again (i.e. in polynomial time) to what we want. In other words, $\mathsf{suffix}(x, y) = F(\ell(x), x)$ using the solution of the IVP: $F(0, y) = x$, $\frac{\partial F(T,y)}{\partial T} = \mathsf{if}(\ell(F(t, x)) = 1, 0, -2^{\ell(F(t,x))-1})$.

4 Computability and Discrete ODEs

Here, we consider functions defined by ODEsunder the prism of computability. It serves as an introductory illustration of the close relationship between discrete ODEs and recursive schemata before moving to efficient algorithms and complexity classes characterizations with a finer approach. This part is clearly inspired by ideas from [6, 7], but adapted here for our framework of discrete ODEs. We refer to [30, 28] for basic definitions from computability.

▶ **Definition 15** ((Scalar) Discrete ODE schemata)**.** *Given $g : \mathbb{N}^p \to \mathbb{N}$ and $h : \mathbb{Z} \times \mathbb{N}^{p+1} \to \mathbb{Z}$, we say that f is defined by discrete ODE solving from g and h, denoted by $f = \mathrm{ODE}(g, h)$, if $f : \mathbb{N}^{p+1} \to \mathbb{Z}$ corresponds to the (necessarily unique) solution of Initial Value Problem*

$$\frac{\partial f(x,\mathbf{y})}{\partial x} = h(f(x, \mathbf{y}), x, \mathbf{y}), \qquad f(0, \mathbf{y}) = g(\mathbf{y}). \tag{2}$$

▶ Remark 16. To be more general, we could take $g : \mathbb{N}^p \to \mathbb{Z}$. However, this would be of no use in the context of this paper. We could also consider vectorial equations as in what follows which are very natural in the context of ODEs, and we would get similar statements.

It is clear that primitive recursion schemes can be reformulated as discrete ODE schemata. The restriction to Linear ODEs is very natural, in particular as this class of ODEs has a highly developed theory for the continuous setting. It is very instructive to realize that the class of functions definable by Linear ODEs is exactly the well-known class of elementary functions [17] (we will see later how additional requirements lead to a characterization of polynomial time functions).

▶ **Definition 17** (Linear ODE schemata)**.** *Given a vector $\mathbf{G} = (G_i)_{1\leq i\leq k}$ matrix $\mathbf{A} = (A_{i,j})_{1\leq i,j\leq k}$, $\mathbf{B} = (B_i)_{1\leq i\leq k}$ whose coefficients corresponds to functions $g_i : \mathbb{N}^p \to \mathbb{N}^k$, and $a_{i,j} : \mathbb{N}^{p+1} \to \mathbb{Z}$ and $b_{i,j} : \mathbb{N}^{p+1} \to \mathbb{Z}$ respectively, we say that $\mathbf{f}$ is obtained by linear ODE solving from g, A and B, denoted by $\mathbf{f} = \mathrm{LI}(\mathbf{G}, \mathbf{A}, \mathbf{B})$, if $f : \mathbb{N}^{p+1} \to \mathbb{Z}^k$ corresponds to the (necessarily unique) solution of Initial Value Problem*

$$\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial x} = \mathbf{A}(x, \mathbf{y}) \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}(x, \mathbf{y}), \qquad \mathbf{f}(0, \mathbf{y}) = \mathbf{G}(\mathbf{y}). \tag{3}$$

▶ **Theorem 18** (A discrete ODE characterization of elementary functions)**.** *The set of elementary functions $\mathcal{E}$ is the intersection with $\mathbb{N}^{\mathbb{N}}$ of the smallest set of functions that contains the zero functions $\mathbf{0}$, the projection functions π_i^p, the successor function $\mathbf{s}$, addition $+$, subtraction $-$, and that is closed under composition and discrete linear ODE schemata (respectively: scalar discrete linear ODE schemata)* LI.

By adding suitable towers of exponential, the above result can be generalized to characterize the various levels of the Grzegorczyk hierarchy. We can also reexpress Kleene's minimization:

▶ **Theorem 19** (Discrete ODE computability and classical computability are equivalent). *A total function $f : \mathbb{N}^p \to \mathbb{N}$ is total recursive iff there exist some functions $h_1, h_2 : \mathbb{N}^{p+1} \to \mathbb{N}^2$ in the smallest set of functions that contains the zero functions $\mathbf{0}$, the projection functions π_i^p, the successor function $\mathbf{s}$, and that is closed under composition and discrete* ODE *schemata such that: for all $\mathbf{y}$, there exists some $T = T(\mathbf{y})$ with $h_2(T, \mathbf{y}) \neq 0$; $f(\mathbf{y}) = h_1(T, \mathbf{y})$ where T is the smallest such T.*

5 Restricted recursion and integration schemes

In order to talk about complexity instead of computability, we need to add some restrictions on the integration scheme. This follows from the following remark:

▶ **Example 20.** The solution of the (degree 2) polynomial ODE $y_1' = y_1$, $y_i' = y_{i-1}' y_i'$ for $i = 1, \ldots d$ is growing faster than a tower of falling exponentials. Consequently, it is not polynomial time computable as just outputting its value in binary cannot be done in polynomial time.

We consequently propose to introduce the following variation on the notion of derivation: derivation along some function $\mathcal{L}(x, \mathbf{y})$.

▶ **Definition 21** ($\mathcal{L}$-ODE). *Let $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$. We write*

$$\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \mathcal{L}} = \frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \mathcal{L}(x,\mathbf{y})} = \mathbf{h}(\mathbf{f}(x,\mathbf{y}), x, \mathbf{y}), \tag{4}$$

as a formal synonym for $\mathbf{f}(x+1,\mathbf{y}) = \mathbf{f}(x,\mathbf{y}) + (\mathcal{L}(x+1,\mathbf{y}) - \mathcal{L}(x,\mathbf{y})) \cdot \mathbf{h}(\mathbf{f}(x,\mathbf{y}), x, \mathbf{y})$.

▶ Remark 22. This is motivated by the fact that the latter expression is similar to classical formula for classical continuous ODEs:

$$\frac{\delta f(x,\mathbf{y})}{\delta x} = \frac{\delta \mathcal{L}(x,\mathbf{y})}{\delta x} \cdot \frac{\delta f(x,\mathbf{y})}{\delta \mathcal{L}(x,\mathbf{y})}.$$

This will allow us to simulate suitable changes of variables using this analogy. We will talk about $\mathcal{L}$-IVP when some initial condition is added. An important special case is when $\mathcal{L}(x, \mathbf{y})$ corresponds to the length $\mathcal{L}(x, \mathbf{y}) = \ell(x)$ function: we will call this special case length-ODEs.

5.1 General theory

The main result of this part illustrates one key property of the $\mathcal{L}$-ODE scheme from a computational point of view: its dependence on the number of distinct values of function $\mathcal{L}$.

▶ **Definition 23** ($Jump_{\mathcal{L}}$). *Let $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$ be some function. Fixing $\mathbf{y} \in \mathbb{N}^p$, let $Jump_{\mathcal{L}}(x, \mathbf{y}) = \{0 \leq i \leq x-1 | \mathcal{L}(i+1, \mathbf{y}) \neq \mathcal{L}(i, \mathbf{y})\}$ be the set of positive integers less than x after which the value of $\mathcal{L}$ changes.*

We also write: $J_{\mathcal{L}} = |Jump_{\mathcal{L}}(x, \mathbf{y})|$ for its cardinality; $\alpha : [0..J_{\mathcal{L}}-1] \to Jump_{\mathcal{L}}(x, \mathbf{y})$ for an increasing function enumerating the elements of $Jump_{\mathcal{L}}(x, \mathbf{y})$: If $i_0 < i_1 < i_2 < \cdots < i_{J_{\mathcal{L}}-1}$ denote all elements of $Jump_{\mathcal{L}}(x, \mathbf{y})$, then $\alpha(j) = i_j \in Jump_{\mathcal{L}}(x, \mathbf{y})$.

▶ **Lemma 24.** *Let $\mathbf{f} : \mathbb{N}^{p+1} \to \mathbb{Z}^d$ and $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$ be some functions. Assume that (4) holds. Then $\mathbf{f}(x, \mathbf{y})$ is equal to:*

$$\mathbf{f}(0,\mathbf{y}) + \int_0^{J_{\mathcal{L}}} \Delta \mathcal{L}(\alpha(u), \mathbf{y}) \cdot \mathbf{h}(\mathbf{f}(\alpha(u), \mathbf{y}), \alpha(u), \mathbf{y}) \delta u.$$

▶ Remark 25. Note that the above result would still hold with $\mathcal{L}$ and h taking their images in $\mathbb{R}$.

Its proof is based on (and illustrates) some fundamental aspect of $\mathcal{L}$-ODE from their definition: for fixed $\mathbf{y}$, the value of $\mathbf{f}(x, \mathbf{y})$ only changes when the value of $\mathcal{L}(x, \mathbf{y})$ changes. If the previous hypotheses hold, there is then an alternative view to understand the integral, by using a change of variable, and by building a discrete ODE that mimics the computation of the integral.

▶ **Lemma 26** (Alternative view). *Let $k \in \mathbb{N}$, $f : \mathbb{N}^{p+1} \to \mathbb{Z}^d$, $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$ be some functions and assume that* (4) *holds. Then $\mathbf{f}(x, \mathbf{y})$ is given by $\mathbf{f}(x, \mathbf{y}) = \mathbf{F}(\mathcal{L}(x, \mathbf{y}), \mathbf{y})$ where $\mathbf{F}$ is the solution of initial value problem*

$$\frac{\partial \mathbf{F}(t,\mathbf{y})}{\partial t} \quad = \quad \Delta\mathcal{L}(t, \mathbf{y}) \cdot \mathbf{h}(\mathbf{F}(t, \mathbf{y}), t, \mathbf{y}) \qquad \mathbf{F}(0, \mathbf{y}) \quad = \quad \mathbf{f}(\mathcal{L}(0, \mathbf{x}), \mathbf{y}).$$

Conversely, if there is such a function $\mathbf{F}$, then a discrete ODE of the type of (4) *can easily be derived.*

5.2 Linear length-ODEs

▶ Remark 27. In all previous reasonings, we considered that a function over the integers is polynomial time computable if it is in the length of all its arguments, as this is the usual convention. When not explicitly stated, this is our convention. As usual, we also say that some vectorial function (respectively: matrix) is polynomial time computable if all its components are. We need sometimes to consider also polynomial dependency directly in some of the variables and not on their length: This happens in the next fundamental lemma.

We write $\| \cdots \|$ for the sup norm: given some matrix $\mathbf{A} = (A_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m}$, $\|A\| = \max_{i,j} A_{i,j}$.

▶ **Lemma 28** (Fundamental observation). *Consider the ODE*

$$\mathbf{f}'(x, \mathbf{y}) = \mathbf{A}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}). \tag{5}$$

Assume:

1. *The initial condition $\mathbf{G}(\mathbf{y}) =^{def} \mathbf{f}(0, \mathbf{y})$, as well as Matrix $\mathbf{A}$ and vector $\mathbf{B}$ are polynomial time computable.*
2. *$\ell(\|\mathbf{A}(f, x, \mathbf{y})\|) \leq \ell(\|\mathbf{f}\|) + p_{\mathbf{A}}(x, \ell(\mathbf{y}))$ for some polynomial p_A*
3. *$\ell(\|\mathbf{B}(f, x, \mathbf{y})\|) \leq \ell(\|\mathbf{f}\|) + p_{\mathbf{B}}(x, \ell(\mathbf{y}))$ for some polynomial p_B*

Then its solution $\mathbf{f}(x, \mathbf{y})$ is polynomial time computable in x and the length of $\mathbf{y}$.

Proof sketch. This comes from the explicit form of solutions provided by Lemma 9, considered then as an algorithm (see Remark 10). The point is then to analyse the size of all the involved quantities in order to state that this remains indeed polynomial time feasible. ◀

We now go to specific forms of linear ODEs.

▶ **Definition 29.** *A* sg*-polynomial expression $P(x_1, ..., x_h)$ is an expression built-on $+, -, \times$ (often denoted $\cdot$) and* sg() *functions over a set of variables $V = \{x_1, ..., x_h\}$ and integer constants. The degree $\deg(x, P)$ of a term $x \in V$ in P is defined inductively as follows:*

- $\deg(x, x) = 1$ *and for $x' \in X \cup \mathbb{Z}$ such that $x' \neq x$,* $\deg(x, x') = 0$
- $\deg(x, P + Q) = \max\{\deg(x, P), \deg(x, Q)\}$
- $\deg(x, P \times Q) = \deg(x, P) + \deg(x, Q)$
- $\deg(x, \mathit{sg}(P)) = 0$

A sg*-polynomial expression P is essentially constant in x if* $\deg(x, P) = 0$.

Compared to the classical notion of degree in polynomial expression, all subterms that are within the scope of a sign function contributes 0 to the degree. A vectorial function (resp. a matrix or a vector) is said to be a sg-polynomial expression if all its coordinates (resp. coefficients) are. It is said to be *essentially constant* if all its coefficients are.

▶ **Definition 30.** *A (possibly vectorial) sg-polynomial expression $\mathbf{g}(\mathbf{f}(x,\mathbf{y}),x,\mathbf{y})$ is essentially linear in $\mathbf{f}(x,\mathbf{y})$ if it is of the form $\mathbf{g}(\mathbf{f}(x,\mathbf{y}),x,\mathbf{y}) = \mathbf{A}[\mathbf{f}(x,\mathbf{y}),x,\mathbf{y}] \cdot \mathbf{f}(x,\mathbf{y}) + \mathbf{B}[\mathbf{f}(x,\mathbf{y}),x,\mathbf{y}]$ where $\mathbf{A}$ and $\mathbf{B}$ are sg-polynomial expressions essentially constant in $\mathbf{f}(x,\mathbf{y})$.*

▶ **Example 31.** The expression $P(x,y,z) = x \cdot \mathsf{sg}((x^2-z)\cdot y) + y^3$ is linear in x, essentially constant in z and not linear in y. The expression $P(x, 2^{\ell(y)}, z) = \mathsf{sg}(x^2-z)\cdot z^2 + 2^{\ell(y)}$ is essentially constant in x, essentially linear in $2^{\ell(y)}$ (but not essentially constant) and not essentially linear in z. The expression: $\mathsf{if}(x,y,z) = y + \bar{\mathsf{sg}}(x)\cdot(z-y) = y + (1-\mathsf{sg}(x))\cdot(z-y)$ is essentially constant in x and linear in y and z.

▶ **Definition 32.** *Function $\mathbf{f}$ is linear $\mathcal{L}$-ODE definable (from $\mathbf{u}$, $\mathbf{g}$ and $\mathbf{h}$) if it corresponds to the solution of $\mathcal{L}$-IVP*

$$\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \mathcal{L}} = \mathbf{u}(\mathbf{f}(x,\mathbf{y}),\mathbf{h}(x,\mathbf{y}),x,\mathbf{y}) \qquad f(0,\mathbf{y}) = \mathbf{g}(\mathbf{y}) \tag{6}$$

where $\mathbf{u}$ is essentially linear in $\mathbf{f}(x,\mathbf{y})$. When $\mathcal{L}(x,\mathbf{y}) = \ell(x)$, such a system is called linear length-ODE.

The previous statements lead to the following:

▶ **Lemma 33** (Key Observation for linear $\mathcal{L}$-ODE). *Assume that $\mathbf{f}$ is the solution of (6) and that functions $\mathbf{g},\mathbf{h},\mathcal{L}$ and $Jump_{\mathcal{L}}$ are computable in polynomial time. Then, $\mathbf{f}$ is computable in polynomial time.*

6 A characterization of polynomial time

The above result shows that functions defined by linear length-ODE from functions computable in polynomial time, are indeed polynomial time. We are now ready to introduce a recursion scheme based on solving linear differential equations to capture polynomial time.

▶ Remark 34. Since the functions we define take their values in $\mathbb{N}$ and have output in $\mathbb{Z}$, composition is an issue. Instead of considering restrictions of these functions with output in $\mathbb{N}$ (which is always possible, even by syntactically expressible constraints), we simply admit that composition may not be defined in some cases. In other words, we consider that composition is a partial operator.

▶ **Definition 35** ($\mathbb{DL}$). *Let $\mathbb{DL}$ be the smallest subset of functions, that contains $\mathbf{0}$, $\mathbf{1}$, projections π_i^p, the length function $\ell(x)$, the addition function $x{+}y$, the subtraction function $x{-}y$, the multiplication function $x \times y$ (often denoted $x \cdot y$), the sign function $\mathsf{sg}(x)$ and closed under composition (when defined) and linear length-ODE scheme.*

▶ Remark 36. As our proofs show, the definition of $\mathbb{DL}$ would remain the same by considering closure under any kind of $\mathcal{L}$-ODE with $\mathcal{L}$ satisfying the hypothesis of Lemma 33.

▶ **Example 37.** A number of natural functions are in $\mathbb{DL}$. Functions $2^{\ell(x)}$, $2^{\ell(x)\cdot\ell(y)}$, $\mathsf{if}(x,y,z)$, $\mathsf{suffix}(x,y)$, $\lfloor\sqrt{x}\rfloor$, $\lfloor\frac{x}{y}\rfloor$, $2^{\lfloor\sqrt{x}\rfloor}$ all belong to $\mathbb{DL}$ by some linear length-ODE easy to establish.

▶ **Theorem 38.** $\mathbb{DL} = \mathbf{FPTIME}$

sketch. $\boxed{\subseteq}$ Consequence of Lemma 33, on the fact that arithmetic operations that are allowed can be computed in polynomial time and that **FPTIME** is closed under composition.

$\boxed{\supseteq}$ A register machine program is a finite sequence of ordered labeled instructions acting on a finite set of registers $R_0,...,R_k$. Such a machine is equipped with assignment instructions ($R_j := R_j + R_i$, $R_j := R_j - R_i$, $R_j := i$ for $i \in \{0,1\}$), control flow statement (if$R_j = 0$ goto p) and halt instruction.

Let $f : \mathbb{N}^p \longrightarrow \mathbb{N}$ be computable in polynomial time and M a k registers machine that computes f in time $\ell(\mathbf{x})^c$ for some $c \in \mathbb{N}$ (with $\mathbf{x}$ in registers $R_1, ..., R_p$, $p \leq k$, at startup and $f(\mathbf{x})$ in R_0 at the end). It is well known that polynomial time on register machines and on Turing machines is the same.

The computation of M is defined by simultaneous recursion scheme on length for functions $R_0(t,\mathbf{x}), ..., R_k(t,\mathbf{x})$ and $\mathsf{inst}(t,\mathbf{x})$: it gives, respectively, the values of each register and the label of the current instruction at time $\ell(t)$. It is shown to correspond to some length-linear ODE. As an example, we describe function $\mathsf{inst}(t,\mathbf{x})$ below.

Let next_l^I, next_l^h, $h \leq k$ describe the evolution of function inst and of register R_h between two instants of time after firing instruction with label l. If l is of the type $R_j := R_j - R_i$ (or is any assignment), then $\mathsf{next}_l^I = 1$ since $\mathsf{inst}(t+1,\mathbf{x}) = \mathsf{inst}(t,\mathbf{x}) + 1$. If l if of the type if $R_j = 0$ goto p, $\mathsf{next}_l^I = \mathsf{if}(R_j(t,\mathbf{x}), p - \mathsf{inst}(t,\mathbf{x}), 1)$ since, in case $R_j(t,\mathbf{x}) = 0$ instruction number goes from $\mathsf{inst}(t,\mathbf{x})$ to p. The definition of each next_l^h is similar.

The value of $\mathsf{inst}(t+1,\mathbf{x})$ can then be described from $\mathsf{inst}(t,\mathbf{x})$ and $R_h(t,\mathbf{x})$, $h \leq k$, by cases depending on the type of the current instruction. Expanded as an arithmetic expression, this gives:

$$\frac{\partial \mathsf{inst}}{\partial \ell}(t,\mathbf{x}) = \sum_{l=0}^{m} \Big(\prod_{i=0}^{l-1} \mathsf{sg}(\mathsf{inst}(t,\mathbf{x}) - i) \Big) \cdot \bar{\mathsf{sg}}(\mathsf{inst}(t,\mathbf{x}) - l) \cdot \mathsf{next}_l^I.$$

It is proved to be of the form

$$\frac{\partial R_j}{\partial \ell}(t,\mathbf{x}) = \sum_{l=0}^{m} \Big(\prod_{i=0}^{l-1} \mathsf{sg}(\mathsf{inst}(t,\mathbf{x}) - i) \Big) \cdot \bar{\mathsf{sg}}(\mathsf{inst}(t,\mathbf{x}) - l) \cdot \mathsf{next}_l^j$$

for some next_l^j encoding the various types of instructions.

The definition is similar (and easier) for each function $R_j(t,\mathbf{x})$. Observe that in the expression above, there is at most one occurrence of $\mathsf{inst}(t,\mathbf{x})$ and $R_j(t,\mathbf{x})$ that is not under the scope of an essentially constant function (i.e. the sign functions). Hence, the expressions are of the prescribed form.

We know M works in time $\ell(\mathbf{x})^c$ for some fixed $c \in \mathbb{N}$. Both functions $\ell(\mathbf{x}) = \ell(x_1) + \ldots + \ell(x_p)$ and $B(\mathbf{x}) = 2^{\ell(\mathbf{x}) \cdot \ell(\mathbf{x})}$ are in $\mathbb{DL}$. It is easily seen that : $\ell(\mathbf{x})^c \leq B^{(c)}(\ell(\mathbf{x})))$ where $B^{(c)}$ is the c-fold composition of function B. We conclude by setting $f(\mathbf{x}) = R_0(B^{(c)}(\max(\mathbf{x})), \mathbf{x})$. ◀

▶ **Definition 39** (Normal linear $\mathcal{L}$-ODE (N$\mathcal{L}$-ODE)). *Function $\mathbf{f}$ is definable by a normal linear $\mathcal{L}$-ODE if it corresponds to the solution of $\mathcal{L}$-IVP $\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \mathcal{L}} = \mathbf{u}(\mathbf{f}(x,\mathbf{y}), x, \mathbf{y})$, $\quad \mathbf{f}(0,\mathbf{y}) = \mathbf{v}(\mathbf{y})$ where $\mathbf{u}$ is essentially linear in $\mathbf{f}(x,\mathbf{y})$ and $\mathbf{v}$ is either the identity, a projection or a constant function.*

From the proof of Theorem 38 the result below can be easily obtained. It expresses that composition needs to be used only once as exemplified in the above definition.

▶ **Theorem 40** (Alternative characterization of **FPTIME**). *A function $\mathbf{f} : \mathbb{N}^p \to \mathbb{Z}$ is in* **FPTIME** *iff $f(\mathbf{y}) = \mathbf{g}(\ell(\mathbf{y})^c, \mathbf{y})$ for some integer c and some $\mathbf{g} : \mathbb{N}^{p+1} \to \mathbb{Z}^k$ solution of a normal linear length-ODE $\frac{\partial \mathbf{g}(x,\mathbf{y})}{\partial \ell(x)} = \mathbf{u}(\mathbf{g}(x,\mathbf{y}), x, \mathbf{y})$.*

▶ Remark 41. From similar arguments, **FPTIME** also corresponds to the class defined as $\mathbb{DL}$ but where linear length-ODE scheme is replaced by normal linear length-ode scheme: this forbids a function already defined with some ODE scheme to be used into some other ODE scheme.

7 A characterization of FNP

This can be extended to more general class such as **FNP**. A function can always seen by its graph Q: A relation Q is in **FNP** if and only if there is a deterministic time verifier, given an arbitrary input (x, y) determines whether $(x, y) \in Q$. Equivalently, Q is **FNP** if and only if there is a non deterministic polynomial time algorithm that, given an arbitrary input x, can find some y such that $(x, y) \in Q$. A function $w : \mathbb{N}^{p+1} \to \mathbb{Z}$ is bounded if for all $\mathbf{x} \in \mathbb{N}^p$, $w(\mathbf{x}) \leq \|\mathbf{x}\|$.

▶ **Definition 42** (Normal linear $\mathcal{L}$-ODE with parameter). *Let $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$. Function $\mathbf{f} : \mathbb{N}^{p+1} \to \mathbb{Z}^k$ is definable by a $\mathcal{L}$-ODE with parameter if there exists a bounded function $w : \mathbb{N}^{p+1} \to \mathbb{Z}$ such that $\mathbf{f}$ is the solution of an equation $\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \mathcal{L}} = \mathbf{u}(\mathbf{f}(x, \mathbf{y}), w(x, \mathbf{y}), x, \mathbf{y})$ $\mathbf{f}(0, \mathbf{y}) = \mathbf{v}(\mathbf{y})$, where $\mathbf{u}$ is essentially linear in $\mathbf{f}(x, \mathbf{y})$ and $\mathbf{v}$ is either the identity, a projection or a constant function.*

▶ **Theorem 43** (Characterization of **FNP**). *A function $\mathbf{f} : \mathbb{N}^p \to \mathbb{N}^k$ is in* **FNP** *iff $f(\mathbf{y}) = \mathbf{g}(2^{\ell(\mathbf{y})^c}, \mathbf{y})$ for some integer c and some $\mathbf{g} : \mathbb{N}^{p+1} \to \mathbb{N}^k$, solution of a normal linear length-ODE with parameter $\frac{\partial \mathbf{g}(x,\mathbf{y})}{\partial \ell(x)} = \mathbf{u}(\mathbf{g}(x, \mathbf{y}), w(x, \mathbf{y}), x, \mathbf{y})$ for some bounded $w : \mathbb{N}^{p+1} \to \mathbb{N}$.*

8 Discussions and further works

Our aim in this article was to give the basis of a presentation of complexity theory based on discrete Ordinary Differential Equations and their basic properties. We demonstrated the particular role played by affine ordinary differential equations in complexity theory, as well as the concept of derivation along some particular function (i.e. change of variable) to guarantee a low complexity.

Previous ideas used here for **FPTIME** can also be extended to provide a characterization of other complexity classes: This includes the possibility of characterizing the class $\mathbf{P}_{[0,1]}$ of functions computable in polynomial time over the reals in the sense of computable analysis, or more general classes of classical discrete complexity theory such as **FPSPACE**. For the clarity of the current exposition, as this would require to introduce other types of schemata of ODEs, we leave this characterization (and improvement of the corresponding schemata to "the most natural and powerful form") for future work, but we believe the current article basically provides the key types of arguments to conceive that this is indeed possible.

More generally, it is also very instructive to revisit classical algorithmic under this viewpoint, and for example one may realize that even inside class **PTIME**, the Master Theorem (see e.g. [11, Theorem 4.1] for a formal statement) can be basically read as a result on (the growth of) a particular class of discrete time length ODEs. Several recursive algorithms can then be reexpressed as particular discrete ODEs of specific type.

References

1 V. I. Arnold. *Ordinary Differential Equations.* MIT Press, 1978.

2 S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

3 G. Birkhoff and G.-C. Rota. *Ordinary Differential Equations.* John Wiley & Sons, 4th edition, 1989.

4 O. Bournez, D. S. Graça, and A. Pouly. Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. *Journal of the ACM*, 64(6):38:1–38:76, 2017. `doi:10.1145/3127496`.

5 Olivier Bournez and Amaury Pouly. A Survey on Analog Models of Computation. In Vasco Brattka and Peter Hertling, editors, *Handbook of Computability and Complexity in Analysis.* Springer. To appear, 2018.

6 Manuel L. Campagnolo. *Computational Complexity of Real Valued Recursive Functions and Analog Circuits.* PhD thesis, Universidade Técnica de Lisboa, 2001.

7 Manuel L. Campagnolo, Cristopher Moore, and José Félix Costa. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity*, 18(4):977–1000, 2002.

8 A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.

9 E. A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations.* Mc-Graw-Hill, 1955.

10 Pieter Collins and Daniel S Graça. Effective computability of solutions of ordinary differential equations the thousand monkeys approach. *Electronic Notes in Theoretical Computer Science*, 221:103–114, 2008.

11 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms (third edition).* MIT press, 2009.

12 AO Gelfand. *Calcul des différences finies.* Dunod, 1963.

13 David Gleich. Finite calculus: A tutorial for solving nasty sums. *Stanford University*, 2005.

14 Ronald L Graham, Donald E Knuth, Oren Patashnik, and Stanley Liu. Concrete mathematics: a foundation for computer science. *Computers in Physics*, 3(5):106–107, 1989.

15 FA Izadi, N Aliev, and G Bagirov. *Discrete Calculus by Analogy.* Bentham Science Publishers, 2009.

16 Charles Jordan and Károly Jordán. *Calculus of finite differences*, volume 33. American Mathematical Soc., 1965.

17 L. Kalmár. Egyzzerü példa eldönthetetlen aritmetikai problémára. *Mate és Fizikai Lapok*, 50:1–23, 1943.

18 A. Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. In *2009 24th Annual IEEE Conference on Computational Complexity*, pages 149–160. IEEE, 2009.

19 Ker-I Ko. On the Computational Complexity of Ordinary Differential Equations. *Information and Control*, 58(1-3):157–194, 1983.

20 Gustavo Lau. Discrete calculus. URL: `http://www.acm.ciens.ucv.ve/main/entrenamiento/material/DiscreteCalculus.pdf`.

21 D. Leivant. Predicative recurrence and computational complexity I: Word recurrence and poly-time. In Peter Clote and Jeffery Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1994.

22 D. Leivant and J-Y Marion. Lambda Calculus Characterizations of Poly-Time. *Fundamenta Informatica*, 19(1,2):167,184, September 1993.

23 Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: substitution and poly-space. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th Workshop, CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 369–380, Kazimierz, Poland, 1995. Springer.

24 Bruno Loff, José Félix Costa, and Jerzy Mycka. The New Promise of Analog Computation. In *Computability in Europe 2007: Computation and Logic in the Real World.*, 2007.

25 Cristopher Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, 162(1):23–44, August 1996.

26 Jerzy Mycka and José Félix Costa. What lies beyond the mountains? Computational systems beyond the Turing limit. *European Association for Theoretical Computer Science Bulletin*, 85:181–189, February 2005.

27 Jerzy Mycka and José Félix Costa. The $P \neq NP$ conjecture in the context of real and complex analysis. *Journal of Complexity*, 22(2):287–303, 2006.

28 P. Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the foundations of mathematics*. North-Holland, April 1992.

29 Amaury Pouly. *Continuous models of computation: from computability to complexity*. PhD thesis, Ecole Polytechnique and Unidersidade Do Algarve, 2015. https://pastel.archives-ouvertes.fr/tel-01223284, Ackermann Award 2017.

30 H. E. Rose. *Subrecursion, Functions and Hierarchies*. Clarendon Press, Oxford, 1984.

On the Coalgebra of Partial Differential Equations

Michele Boreale

Università di Firenze, Dipartimento di Statistica, Informatica, Applicazioni (DiSIA) “G. Parenti”,
Viale Morgagni 65, I-50134 Firenze, Italy
michele.boreale@unifi.it

Abstract

We note that the coalgebra of formal power series in commutative variables is final in a certain subclass of coalgebras. Moreover, a system Σ of polynomial PDEs, under a coherence condition, naturally induces such a coalgebra over differential polynomial expressions. As a result, we obtain a clean coinductive proof of existence and uniqueness of solutions of initial value problems for PDEs. Based on this characterization, we give complete algorithms for checking equivalence of differential polynomial expressions, given Σ.

2012 ACM Subject Classification Theory of computation → Invariants; Theory of computation → Operational semantics

Keywords and phrases coalgebra, partial differential equations, polynomials

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.24

Acknowledgements Three anonymous reviewers have provided valuable comments.

1 Introduction

The last two decades have seen an impressive growth of formal methods and tools for continuous and hybrid systems, centered around techniques for reasoning on ordinary differential equations (ODEs), see e.g. [29, 28, 21, 11, 15, 6] and references therein. On the other hand, formal methods for systems defined by *partial* differential equations (PDEs) have not undergone a comparable development. The present paper is meant as an initial contribution towards this development.

Like in our previous works on ODEs [5, 7], our starting point is a simple operational view of differential equations as programs for calculating the Taylor coefficients of a function. Taking a transition in such a program corresponds to taking a function's derivative. An output is returned as the result of evaluating the current state (function) at a fixed expansion point, for example the origin. This idea is certainly not new: it is for example at the root of classical methods to numerically solve ODEs.

We focus here on polynomial PDEs, which are expressive enough for the vast majority of problems arising in applications, and systematically pursue the above operational view in the framework of coalgebras. We first introduce a subclass of coalgebras that enjoy a commutativity property of transitions, then note that formal power series in commutative variables (CFPSs) are final for this subclass (Section 2). A system Σ of PDEs and a specification of initial data together form an initial value problem. Under a coherence condition (Section 3), an initial value problem induces a coalgebra structure over the set of differential polynomials. The solution of the initial value problem is therefore obtained as the unique coalgebra morphism from the set of such polynomials to the final coalgebra of CFPSs. This way, we obtain an elementary and clean proof of existence and uniqueness of solutions as CFPSs (Section 4). We also show that coherence is an essential requirement for this result. From a pragmatical point of view, we note that as solutions of PDEs, CFPSs may be viewed as a conservative extension of analytic functions: *if* an analytic solution of Σ exists, *then* its Taylor expansion from 0, seen as a formal power series, coincides with the unique solution in our sense.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 24; pp. 24:1–24:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This characterization is the basis of an algorithm to automatically check polynomial equalities – e.g. conservation laws – valid among the functions defined by given system Σ and a specification of initial data (Section 5). Just like in certain on-the-fly algorithms for bisimulation checking, the underlying idea is, based on the introduced transition structure, to incrementally build a relation until it "closes up", but working here modulo sum and product of polynomials. Concepts from algebraic geometry, notably Gröbner bases, are used to prove the termination and correctness of this algorithm. In fact, we are more general than this, and also give a method to automatically compute the weakest precondition (= set of initial data specifications) under which a given equality is valid in Σ. These algorithms are complete, under a certain finite-parameter condition. This way one can, for example, automatically check conservation laws of a given physical system. Relationship with our previous work on ODEs [5, 7], as well as with related work by other authors, is discussed in the concluding section (Section 6). Proofs omitted from the present version will appear in a forthcoming online version of the paper.

2 Commutative coalgebras

Let X be a finite nonempty set of *actions (or variables)*, ranged over by $x, y, ...$ and O a nonempty set. We recall that a *(Moore) coalgebra*[1] with actions in X and outputs in O is a triple $C = (\mathcal{S}, \delta, o)$ where: $\mathcal{S}$ is a set of *states*, $\delta : \mathcal{S} \times X \to \mathcal{S}$ is a *transition* function, and $o : \mathcal{S} \to O$ is an *output* function (see e.g. [27]). A *bisimulation* in C is a binary relation $R \subseteq \mathcal{S} \times \mathcal{S}$ such that whenever $s\, R\, t$ then: (a) $o(s) = o(t)$, and (b) for each x, $\delta(s, x)\, R\, \delta(t, x)$. It is an (easy) consequence of the general theory of bisimulation that a largest bisimulation over C, called bisimilarity and denoted by $\sim_C$, exists, is the union of all bisimulation relations, and is an equivalence relation over $\mathcal{S}$. Given two coalgebras with actions in X and outputs in O, C_1 and C_2, a *morphism* from C_1 to C_2 is a function $\mu : \mathcal{S}_1 \to \mathcal{S}_2$ that: (1) preserves outputs ($o_1(s) = o_2(\mu(s))$), and (2) preserves transitions ($\mu(\delta_1(s, x)) = \delta_2(\mu(s), x)$), for each state s and action x). It is an easy consequence of this definition that a morphism preserves bisimulation in both directions, that is: $s \sim_{C_1} t$ if and only if $\mu(s) \sim_{C_2} \mu(t)$.

We introduce now the subclass of Moore coalgebras we will focus on. We say a coalgebra C has *commutative actions* (or just that is *commutative*) if for each state s and actions x, y, it holds that $\delta(\delta(s, x), y) \sim_C \delta(\delta(s, y), x)$. We will introduce below an example of commutative coalgebra. In what follows, we let σ range over X^*, and, for any state s, let $s(\sigma)$ be defined inductively as: $s(\epsilon) \stackrel{\triangle}{=} s$ and $s(x\sigma) \stackrel{\triangle}{=} \delta(s, x)(\sigma)$.

▶ **Lemma 2.1.** *Let C be a commutative coalgebra. If $\sigma, \sigma' \in X^*$ are permutations of one another then for any state $s \in \mathcal{S}$, $s(\sigma) \sim_C s(\sigma')$.*

We now introduce the coalgebra of formal power series in commutative variables with outputs in $O = \mathbb{R}$. Let $X^{\otimes}$, ranged over by $\tau, \tau', ...$, be the set of *monomials*[2] that can be formed from $X = \{x_1, ..., x_n\}$, in other words, the commutative monoid freely generated by X.

▶ **Definition 2.2** (commutative formal power series). *Let X be a finite nonempty alphabet. A* commutative formal power series *(CFPS) with indeterminates in X and coefficients in $\mathbb{R}$ is a total function $f : X^{\otimes} \to \mathbb{R}$. The set of these CFPSs will be denoted by $\mathcal{F}(X)$, or simply $\mathcal{F}$ if X is understood from the context.*

[1] In the paper, we only consider Moore coalgebras. For brevity, we shall omit the qualification "Moore".

[2] In general, we shall adopt for monomials the same notation we use for strings, as the context is sufficient to disambiguate. In particular, we overload the symbol ϵ to denote both the empty string and the empty monomial.

In the rest of the section, we fix an arbitrary X. We will sometimes use the suggestive notation

$$\sum_\tau f(\tau) \cdot \tau$$

to denote a CFPS $f = \lambda\tau.f(\tau)$. By slight abuse of notation, for each $r \in \mathbb{R}$, we will denote the CFPS that maps ϵ to r and anything else to 0 simply as r; while x_i will denote the i-th identity, the CFPS that maps x_i to 1 and anything else to 0. In the sequel, $\delta(f,x) \stackrel{\triangle}{=} \frac{\partial f}{\partial x}$ denotes the CFPS obtained by the usual (formal) partial derivative of f along x. For a more workable formulation of this definition, let us introduce the following notation. Let us fix any total order $\mathbf{x} = (x_1, ..., x_n)$ of the variables in X. Given a vector $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_n)$ of nonnegative integers (a *multi-index*), we let $\mathbf{x}^{\boldsymbol{\alpha}}$ denote the monomial $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$. Then $\frac{\partial f}{\partial x_i}$ is defined by the following, for each $\tau = \mathbf{x}^{(\alpha_1,\ldots,\alpha_n)}$

$$\frac{\partial f}{\partial x_i}(\tau) \quad \stackrel{\triangle}{=} \quad (\alpha_i + 1) f(x_i \tau) \,. \tag{1}$$

Finally, we define the coalgebra of CFPSs, $C_\mathcal{F}$

$$C_\mathcal{F} \quad \stackrel{\triangle}{=} \quad (\mathcal{F}, \delta_\mathcal{F}, o_\mathcal{F})$$

where $\delta_\mathcal{F}(f,x) = \frac{\partial f}{\partial x}$ and $o_\mathcal{F}(f) = f(\epsilon)$ (the constant term of f). Bisimilarity in $C_\mathcal{F}$, denoted by $\sim_\mathcal{F}$, coincides with equality. It is easily seen that for each x, y, $\frac{\partial}{\partial y}\frac{\partial f}{\partial x} = \frac{\partial}{\partial x}\frac{\partial f}{\partial y}$, so that $C_\mathcal{F}$ is a commutative coalgebra. Now fix any commutative coalgebra $C = (\mathcal{S}, \delta, o)$. We define the function $\mu : \mathcal{S} \to \mathcal{F}$ as follows. For each $\tau = \mathbf{x}^{\boldsymbol{\alpha}}$

$$\mu(s)(\tau) \quad \stackrel{\triangle}{=} \quad \frac{o(s(\tau))}{\boldsymbol{\alpha}!} \tag{2}$$

where $\boldsymbol{\alpha}! \stackrel{\triangle}{=} \alpha_1! \cdots \alpha_n!$. Here, abusing slightly notation, we let $o(s(\tau))$ denote $o(s(\sigma))$, for some string σ obtained by arbitrarily ordering the elements in τ: the specific order does not matter, in view of Lemma 2.1 and of condition (a) in the definition of bisimulation.

▶ **Lemma 2.3.** *Let C be a commutative coalgebra and $f = \mu(s)$. For each x, $\frac{\partial f}{\partial x} = \mu(\delta(s,x))$.*

Based on the above lemma and the fact that $\sim_\mathcal{F}$ is equality, we can prove the following corollary, saying that $C_\mathcal{F}$ is *final* in the class of *commutative* coalgebras.

▶ **Corollary 2.4** (coinduction and finality of $C_\mathcal{F}$)**.** *Let C be a commutative coalgebra. The function μ in (2) is the unique coalgebra morphism from C to $C_\mathcal{F}$. Moreover, the following* coinduction *principle is valid: $s \sim_C t$ if and only if $\mu(s) = \mu(t)$ in $\mathcal{F}$.*

Proof. We have: (1) $o(s) = \mu(s)(\epsilon)$ by the definition of μ, and (2) $\mu(\delta(s,x)) = \delta_\mathcal{F}(\mu(s), x)$, by Lemma 2.3. This proves that μ is a coalgebra morphism. Next, we prove that $\sim_\mathcal{F}$ coincides with equality in $\mathcal{F}$. More precisely, we prove that for each τ and for each f, g: $f \sim_\mathcal{F} g$ implies $f(\tau) = g(\tau)$. Proceeding by induction on the length of τ, we see that the base case is trivial, while for the induction step $\tau = x_i\tau'$ we have: $f \sim_\mathcal{F} g$ implies $\frac{\partial f}{\partial x_i} \sim_\mathcal{F} \frac{\partial g}{\partial x_i}$ (bisimilarity), which in turn implies $\frac{\partial f}{\partial x_i}(\tau') = \frac{\partial g}{\partial x_i}(\tau')$ (induction hypothesis); but by (1), $f(x_i\tau') = (\frac{\partial f}{\partial x_i}(\tau'))/(\alpha_i + 1)$ and $g(x_i\tau') = (\frac{\partial g}{\partial x_i}(\tau'))/(\alpha_i + 1)$, and this completes the induction step. From the coincidence of $\sim_\mathcal{F}$ with equality in $\mathcal{F}$, and the fact that any morphism preserves bisimilarity in both directions, the last part of the statement (coinduction) follows immediately. Finally, let ν be any morphism from C to $C_\mathcal{F}$. From the definitions of bisimulation and morphism it is easy to see that for each s, $\mu(s) \sim_\mathcal{F} \nu(s)$: this implies $\mu(s) = \nu(s)$ by coinduction, and proves uniqueness of μ. ◀

We end this section by recalling the sum and product operations on $\mathcal{F}$. For any $\xi = \mathbf{x}^{\boldsymbol{\alpha}}$ and $\tau = \mathbf{x}^{\boldsymbol{\beta}}$, let $\xi \leq \tau$ if for each $i = 1, ..., n$, $\alpha_i \leq \beta_i$; in this case τ/ξ denotes the monomial $\mathbf{x}^{(\beta_1-\alpha_1,...,\beta_n-\alpha_n)}$. We have the following definitions of sum and product. For each $\tau \in X^{\otimes}$:

$$(f+g)(\tau) \triangleq f(\tau)+g(\tau) \qquad\qquad (f\cdot g)(\tau) \triangleq \sum_{\xi\leq\tau} f(\xi)\cdot g(\tau/\xi)\,. \tag{3}$$

These operations correspond to the usual sum and product of functions, when (convergent) CFPS are interpreted as analytic functions. These operations enjoy associativity, commutativity and distributivity. Moreover, if $f(\epsilon) \neq 0$ there exists a unique CFPS $f^{-1} \in \mathcal{F}$ that is a multiplicative inverse of f, that is $f \cdot f^{-1} = 1$. Finally, the following familiar rules of differentiation are satisfied:

$$\frac{\partial(f+g)}{\partial x} \triangleq \frac{\partial f}{\partial x} + \frac{\partial g}{\partial x} \qquad\qquad \frac{\partial(f\cdot g)}{\partial x} \triangleq \frac{\partial f}{\partial x}\cdot g + f\cdot\frac{\partial g}{\partial x}\,. \tag{4}$$

If the *support* of f, $\mathrm{supp}(f) \triangleq \{\tau : f(\tau) \neq 0\}$, is finite, we will call f a *polynomial.* The set of polynomials, denoted by $\mathbb{R}[X]$, is closed under the above defined operations of partial derivative, sum and product (but in general not inverse). Moreover, note that, when confining to polynomials, these operations are well defined even in case the cardinality of the set X of indeterminates is infinite.

3 Coherent systems of PDEs

We first review some notation and terminology from the formal theory of PDEs; these are standard notions, see e.g. [23, 18]. Like in the previous section, assume we are given a finite nonempty set X, which we will call here the *independent* variables. Another nonempty, finite set U of *dependent* variables, disjoint from X, is given; U is ranged over by $u, v, ...$ $\mathcal{D} \triangleq \{u_\tau : u \in U, \tau \in X^{\otimes}\}$ is the set of the *derivatives*; here u_ϵ will be identified with u. $E, F, ...$ range over $\mathcal{P} \triangleq \mathbb{R}[X \cup \mathcal{D}]$, the set of *(differential, multivariate) polynomials* with coefficients in $\mathbb{R}$ and indeterminates in $X \cup \mathcal{D}$. Considered as formal objects, polynomials are just finite-support CFPSs in $\mathcal{F}(X \cup \mathcal{D})$ (see Section 2). As such, they inherit from CFPSs the operations of sum, product and partial derivative, along with the corresponding properties. Syntactically, we shall write polynomials as expressions of the form $\sum_{\alpha\in M} \lambda_\alpha \cdot \alpha$, for $0 \neq \lambda_\alpha \in \mathbb{R}$ and $M \subseteq_{\mathrm{fin}} (X \cup \mathcal{D})^{\otimes}$. For example, $E = v_z u_{xy} + v_y^2 + u + 5x$ is a polynomial[3]. For an independent variable $x \in X$, the *total derivative* of $E \in \mathcal{P}$ along x is just the derivative of E along x, taking into account that $\frac{\partial u_\tau}{\partial x} = u_{x\tau}$. Formally, we have the following.

▶ **Definition 3.1** (total derivative). *The operator $D_x : \mathcal{P} \to \mathcal{P}$ is defined by (note $\sum$ below has only finitely many nonzero terms)*

$$D_x E \triangleq \frac{\partial E}{\partial x} + \sum_{u,\tau} u_{x\tau} \cdot \frac{\partial E}{\partial u_\tau}$$

where $\frac{\partial E}{\partial a}$ denotes the partial derivative of polynomial E along $a \in X \cup \mathcal{D}$.

D_x inherits from partial derivatives rules for sum and product that are the analog of (4). As an example, for the polynomial E above, we have $D_x E = v_{xz} u_{xy} + v_z u_{xxy} + 2v_y v_{xy} + u_x + 5$.

[3] Real arithmetic expressions will be used as a meta-notation for polynomials: e.g. $(u+u_x+1)\cdot(x+u_y)$ denotes the polynomial $xu + uu_y + xu_x + u_x u_y + x + u_y$.

In particular, $D_x u_\tau = u_{x\tau}$ and $D_x x^k = kx^{k-1}$. Just as partial derivatives, total derivatives commute with each other, that is $D_x D_y F = D_y D_x F$. This suggests to extend the notation to monomials: for any monomial $\tau = x_1 \cdots x_m$, we let $D_\tau F$ be $D_{x_1} \cdots D_{x_m} F$, where the order of the variables is irrelevant.

▶ **Definition 3.2** (system of PDEs). *A* system of PDEs *is a nonempty set Σ of equations (pairs) of the form $u_\tau = E$, with $E \in \mathcal{P}$. The set of derivatives u_τ that appear as left-hand sides of equations in Σ is denoted* $\mathrm{dom}(\Sigma)$. *Based on Σ, the set $\mathcal{D}$ can be partitioned into two sets as follows:*

- $\mathcal{P}\mathrm{r}(\Sigma) \overset{\triangle}{=} \{u_\xi \,:\, \tau \leq \xi$ *for some τ s.t.* $u_\tau \in \mathrm{dom}(\Sigma)\}$ *is the set of* principal derivatives *of Σ;*
- $\mathcal{P}\mathrm{a}(\Sigma) \overset{\triangle}{=} \mathcal{D} \setminus \mathcal{P}\mathrm{r}(\Sigma)$ *is the set of* parametric derivatives *of* Σ.

We let $\mathcal{P}_0(\Sigma) \overset{\triangle}{=} \mathbb{R}[X \cup \mathcal{P}\mathrm{a}(\Sigma)]$.

The intuition of $\mathcal{P}\mathrm{a}(\Sigma)$ is that, once we fix the corresponding values, the rest of the solution, hence $\mathcal{P}\mathrm{r}(\Sigma)$, will be uniquely determined (see Example 3.4 below). Note that we do not require that each derivative occurs at most once as left-hand side in Σ. The *infinite prolongation* of a system Σ, denoted Σ^∞, is the system of PDEs of the form $u_{\xi\tau} = D_\xi F$, where $u_\tau = F$ is in Σ and $\xi \in X^\otimes$. Of course, $\Sigma^\infty \supseteq \Sigma$. Moreover, Σ and Σ^∞ induce the *same* sets of principal and parametric derivatives.

A *ranking* is a total order $\prec$ of $\mathcal{D}$ such that: (a) $u_\tau \prec u_{x\tau}$, and (b) $u_\tau \prec v_\xi$ implies $u_{x\tau} \prec v_{x\xi}$, for each $x \in X$, $\tau, \xi \in X^\otimes$ and $u, v \in U$. Dickson's lemma [10] implies that $\mathcal{D}$ with $\prec$ is a well-order, and in particular that there is no infinite descending chain in it. The system Σ is $\prec$*-normal* if, for each equation $u_\tau = E$ in Σ, $u_\tau \succ v_\xi$, for each v_ξ appearing in E. An easy but important consequence of condition (b) above is that if Σ is normal then also its prolongation Σ^∞ is normal.

Now, consider the equational theory over $\mathcal{P}$ induced by the equations in Σ^∞. More precisely, write $E \to_\Sigma F$ if F is the polynomial that is obtained from E by replacing one occurrence of u_τ with G, for some equation $u_\tau = G \in \Sigma^\infty$. Note, in particular, that $E \in \mathcal{P}$ cannot be rewritten if and only if $E \in \mathcal{P}_0(\Sigma)$. We let $=_\Sigma$ denote the reflexive, symmetric and transitive closure of $\to_\Sigma$. The following definition formalizes the key concepts of consistency and coherence of Σ. Basically, as we will show, under the syntactic requirement of normality, which is natural from an algorithmic point of view, consistency is a necessary and sufficient condition for Σ to admit a unique solution under *all* initial data specifications.

▶ **Definition 3.3** (consistency and coherence). *Let Σ be a system of PDEs.*

- *We say Σ is* consistent *if for each $E \in \mathcal{P}$ there is a unique $F \in \mathcal{P}_0(\Sigma)$ such that $E =_\Sigma F$.*
- *Let $\prec$ be a ranking. A system Σ is* $\prec$-coherent *if it is $\prec$-normal and consistent.*

For a consistent system, we can define a *normal form function*

$$S_\Sigma : \mathcal{P} \to \mathcal{P}_0(\Sigma)$$

by letting $S_\Sigma E = F$, for the unique $F \in \mathcal{P}_0(\Sigma)$ such that $E =_\Sigma F$. The term $S_\Sigma E$ will be often abbreviated as SE, if Σ is understood from the context. Deciding if a (finite) system Σ is coherent, for a suitable ranking $\prec$, is of course a nontrivial problem. In a *normal* system, since $\prec$ is a well-order, there are no infinite sequences of rewrites $E_1 \to_\Sigma E_2 \to_\Sigma E_3 \to_\Sigma \cdots$: therefore it is possible to rewrite any E into some $F \in \mathcal{P}_0(\Sigma)$ in a finite number of steps. Proving coherence in this case reduces basically to ensure that Σ contains "enough equations" to make $\to_\Sigma$ confluent. In fact, an even more general problem than checking coherence is completing a normal, non coherent system by new equations so as to make it coherent; or

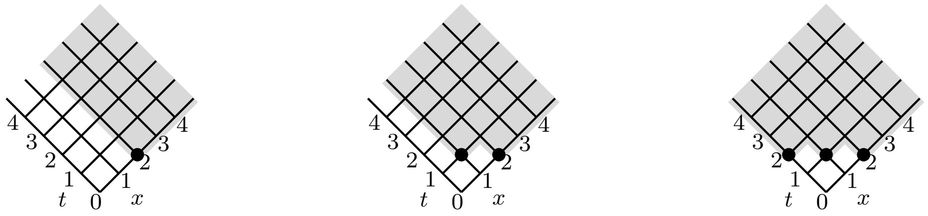


Figure 1 Lattices of u-derivatives, partially ordered by $u_\xi \leq u_\tau$ if and only if $\xi \leq \tau$. With reference to Example 3.4, black circles correspond to left-hand sides of equations, shaded regions to principal derivatives, and non-shaded regions to parametric derivatives.

deciding if this is impossible at all, because the system is intrinsically inconsistent. There is a rich literature on these problems, which we briefly review in the concluding section. The following simple example is enough to demonstrate these concepts for our purposes.

▶ **Example 3.4** (coherence). Consider the heat equation in one spatial dimension, where $X = \{x, t\}$, $U = \{u\}$ and Σ is given by the single equation (for a real parameter $a \neq 0$)

$$u_{xx} \;=\; au_t\,. \tag{5}$$

Here, the principal derivatives are $\mathcal{P}\mathrm{r}(\Sigma) = \{u_{xx\tau} : \tau \in X^{\otimes}\}$, and the parametric ones are $\mathcal{P}\mathrm{a}(\Sigma) = \{u_{t^j} : j \geq 0\} \cup \{u_{xt^j} : j \geq 0\}$ (see Figure 1, left). Since the system has just one equation, it is clearly consistent: indeed, its prolongation Σ^∞ has precisely one equation $u_{xx\tau} = D_\tau\,(au_t)$ for each principal derivative $u_{xx\tau}$. Concerning coherence, we consider the following ranking. Ordering the independent variables as $t < x$ induces a *graded* lexicographic order $\prec_{\mathrm{grlex}}$ over $X^{\otimes}$: that is, monomials are compared first by their total degree, and then lexicographically. We lift $\prec_{\mathrm{grlex}}$ to $\mathcal{D}$ as expected: explicitly, $u_\xi \prec u_\tau$ if, for $\xi = t^i x^j$ and $\tau = t^{i'} x^{j'}$, it holds that either $i + j < i' + j'$ or $(i + j = i' + j'$ and $j' > j)$. Σ is clearly $\prec$-normal.

Next, suppose we build a new system Σ_1 by adding the new equation (with no physical significance)

$$u_{tx} \;=\; u\,.$$

Now the parametric derivatives are u_{t^j} for $j \geq 0$ and u_x, while the remaining derivatives are principal (see Figure 1, center). The prolongation of the new system, Σ_1^∞, has both $u_{txx} = u_x$ and $u_{txx} = au_{tt}$ as equations, which implies $u_x =_{\Sigma_1} au_{tt}$. As $u_x, u_{tt} \in \mathcal{P}\mathrm{a}(\Sigma_1) \subseteq \mathcal{P}_0(\Sigma_1)$, we conclude that Σ_1 is *not* consistent, hence not coherent: indeed, there are two distinct but equivalent normal forms. This suggests that we can complete Σ_1 by inserting a third equation, a so-called *integrability condition*

$$u_{tt} \;=\; \frac{u_x}{a}\,.$$

In the resulting system Σ_2 the set of parametric derivatives has changed to $\{u, u_t, u_x\}$ (see Figure 1, right), and u_{txx} has the (only) normal form u_x. The system Σ_2 can be indeed checked to be consistent, hence coherent. Finally, consider adding to the original system Σ the two equations below, thus obtaining Σ_3

$$u_{tx} \;=\; t \qquad\qquad u_{tt} \;=\; 1\,.$$

Together with (5) these two equations imply $a =_{\Sigma_3} 0$: Σ_3 is not consistent, moreover there is no way of completing it so as to get a consistent system. That is, Σ_3 is (informally speaking) intrinsically inconsistent.

For our purposes, it is enough to know that completing a given system of equations to make it coherent, or deciding that this is impossible, can be achieved by one of many existing computer algebra algorithms. For example, there is a completion procedure by Marvan [18], for which a Maple implementation is also available. See also Reid et al.'s method of reduction to *reduced involutive form* [23], implemented in the Maple rif package. An alternative to these methods is applying a procedure similar to the Knuth-Bendix completion algorithm [13] to the given system. Further references are discussed in the concluding section. In practice, in many cases arising from applications (e.g. mathematical physics), transforming the system into a coherent form for an appropriate ranking can be accomplished manually, without much difficulty. We shall not further dwell on algorithms for coherence checking in the rest of the paper. We end the section with a technical result about normal forms in coherent systems that will be used in the next section.

▶ **Lemma 3.5.** *Let Σ be coherent. For each $x \in X$ and $F \in \mathcal{P}$, $SD_xSF = SD_xF$.*

4 Coalgebraic semantics of initial value problems

We will provide differential polynomials with a coalgebra structure depending on Σ: from this, existence and uniqueness of solutions of initial value problems will follow almost immediately by coinduction (Corollary 2.4). The essential point is that coherence allows for the definition of a transition function based on total derivatives. In the definition below, it is useful to bear in mind that, informally, for a parametric derivative u_τ, the initial data value $\rho(u_\tau)$ specifies the value of $\frac{\partial u}{\partial \tau}$ at the origin.

▶ **Definition 4.1** (initial value problem). *Let Σ be a system of PDEs. A* specification of initial data *for Σ is a mapping $\rho : \mathcal{P}\mathrm{a}(\Sigma) \to \mathbb{R}$. An* initial value problem *is a pair $\mathbf{B} = (\Sigma, \rho)$.*

In what follows, for any function $\psi : U \to \mathcal{F}$, we can consider its homomorphic extension $\mathcal{P} \to \mathcal{F}$, obtained by interpreting each expression E in the obvious way: replace u_τ by $\frac{\partial \psi(u)}{\partial \tau}$, and sum and product by the corresponding operations in $\mathcal{F}$ (see Section 2); an independent variable $x_i \in X$ is interpreted as the i-th identity CFPS. By slight abuse of notation, we will still denote by "ψ" the homomorphic extension of ψ. In part (a) below, recall that for a CFPS f, $f(\epsilon)$ represents, informally, the value of function f at the origin.

▶ **Definition 4.2** (solution of $\mathbf{B}$). *A* solution *of a initial value problem $\mathbf{B} = (\Sigma, \rho)$ is a mapping $\psi : U \to \mathcal{F}$ such that: (a) the initial data specifications are satisfied, that is $\psi(u_\tau)(\epsilon) = \rho(u_\tau)$ for each $u_\tau \in \mathcal{P}\mathrm{a}(\Sigma)$; and (b) all equations are satisfied, that is $\psi(u_\tau) = \psi(F)$ for each $u_\tau = F$ in Σ^∞.*

The following lemma about solutions will be used to prove uniqueness of the solution of $\mathbf{B}$.

▶ **Lemma 4.3.** *Let $\mathbf{B} = (\Sigma, \rho)$ and ψ a solution of $\mathbf{B}$. For each $E, F \in \mathcal{P}$, $E =_\Sigma F$ implies $\psi(E) = \psi(F)$.*

With any coherent (w.r.t. some ranking) Σ and initial data specification ρ, $\mathbf{B} = (\Sigma, \rho)$, we can associate a coalgebra as follows. The initial data specification $\rho : \mathcal{P}\mathrm{a}(\Sigma) \to \mathbb{R}$ can be extended homomorphically to $\mathcal{P}_0(\Sigma) \to \mathbb{R}$, by interpreting $+$ and $\cdot$ as the usual sum and product over $\mathbb{R}$, respectively, and by letting $\rho(x) \overset{\triangle}{=} 0$ for each independent variable $x \in X$. Now we define a coalgebra depending on $\mathbf{B}$:

$$C_{\mathbf{B}} \overset{\triangle}{=} (\mathcal{P}, \delta_\Sigma, o_\rho)$$

where $\delta_\Sigma(E,x) \stackrel{\triangle}{=} SD_xE$ and $o_\rho(E) \stackrel{\triangle}{=} \rho(SE)$. We will denote by $\sim_{\mathbf{B}}$ bisimilarity in $C_{\mathbf{B}}$. As an example of transition, for the heat equation $\Sigma = \{u_{xx} = au_t\}$, one has $\delta_\Sigma(u_{xx}, t) = au_{tt}$.

▶ Remark 4.4. An obvious alternative to the above definition of transition function of $C_{\mathbf{B}}$ would be just letting $\delta_\Sigma(E,x) = D_xE$: this definition in fact would work as well, but it has the computational disadvantage of making the order of the derivatives higher at each step, which would be inconvenient for the algorithms to be developed later on (Section 5).

As expected, $C_{\mathbf{B}}$ is a commutative coalgebra. Moreover, each expression is bisimilar to its normal form. This is the content of the following lemma.

▶ **Lemma 4.5.** *Let $\mathbf{B} = (\Sigma, \rho)$, with Σ coherent. Then: (1) $C_{\mathbf{B}}$ is commutative; and (2) For each $E \in \mathcal{P}$, $E \sim_{\mathbf{B}} SE$.*

Note that since $\delta_\Sigma(\delta_\Sigma(E,x),y) = \delta_\Sigma(\delta_\Sigma(E,y),x)$, for any monomial τ, the notation $\delta_\Sigma(E,\tau)$ is well defined. As a consequence of the previous lemma, part 1, and of Corollary 2.4, there exists a unique morphism from $C_{\mathbf{B}}$ to $C_{\mathcal{F}}$. This morphism is the unique solution of $\mathbf{B}$ we are after. We need a lemma, saying that the unique morphism ϕ from $C_{\mathbf{B}}$ to $C_{\mathcal{F}}$ is compositional.

▶ **Lemma 4.6.** *Let $\mathbf{B} = (\Sigma, \rho)$, with Σ coherent, and let $\phi_{\mathbf{B}}$ be the unique morphism from $C_{\mathbf{B}}$ to $C_{\mathcal{F}}$. Then $\phi_{\mathbf{B}}$ is a homomorphism over $\mathcal{P}$.*

▶ **Theorem 4.7** (coalgebraic semantics of PDEs). *Let $\mathbf{B} = (\Sigma, \rho)$, with Σ coherent. Let $\phi_{\mathbf{B}}$ denote the unique morphism from $C_{\mathbf{B}}$ to $C_{\mathcal{F}}$. Then $\phi_{\mathbf{B}}$ (restricted to U) is the unique solution of $\mathbf{B}$.*

Proof. By virtue of Lemma 4.6, $\phi_{\mathbf{B}}$ coincides with the homomorphic extension of $(\phi_{\mathbf{B}})_{|U}$. We first prove that that $\phi_{\mathbf{B}}$ respects the initial data specification. Let u_τ be parametric. By the definition of coalgebra morphism and of output functions in $C_{\mathcal{F}}$ and $C_{\mathbf{B}}$, we have

$$\phi_{\mathbf{B}}(u_\tau)(\epsilon) \quad = \quad o_{\mathcal{F}}(\phi_{\mathbf{B}}(u_\tau)) \quad = \quad o_\rho(u_\tau) \quad = \quad \rho(Su_\tau) \quad = \quad \rho(u_\tau)$$

which proves the wanted condition. Next, we have to prove that $\phi_{\mathbf{B}}$ satisfies the equations in Σ^∞. But for each such equation, say $u_\tau = F$, we have $Su_\tau =_\Sigma SF$ by the definition of $=_\Sigma$, hence $u_\tau \sim_{\mathbf{B}} F$ by Lemma 4.5(2), hence the thesis by coinduction (Corollary 2.4). We finally prove uniqueness of the solution. Assume ψ is a solution of $\mathbf{B}$, and consider the homomorphic extension of ψ to $\mathcal{P}$, still denoted by ψ. We prove that ψ is a coalgebra morphism from $C_{\mathbf{B}}$ to $C_{\mathcal{F}}$, hence $\psi = \phi_{\mathbf{B}}$ will follow by coinduction (Corollary 2.4). Let $E \in \mathcal{P}$. There are two steps in the proof.

- $\psi(E)(\epsilon) = \rho(SE) = o_\rho(E)$. This follows directly from Lemma 4.3, since $\psi(E) = \psi(SE)$.
- For each x, $\frac{\partial\psi(E)}{\partial x} = \psi(\delta_\Sigma(E,x))$. First, we note that $\frac{\partial\psi(E)}{\partial x} = \psi(D_xE)$. This is proven by induction on the size of E: in the base case when $E = u_\tau$, just use the fact that, by the definition of solution, $\frac{\partial\psi(u_\tau)}{\partial x} = \frac{\partial}{\partial x}\frac{\partial\psi(u)}{\partial\tau} = \frac{\partial\psi(u)}{\partial\tau x} = \psi(u_{\tau x}) = \psi(D_xu_\tau)$; in the induction step, use the fact that ψ is an homomorphism over $\mathcal{P}$, and the derivation rules of D_x and $\frac{\partial}{\partial x}$ for sum and product. Now applying Lemma 4.3, we get $\psi(D_xE) = \psi(SD_xE) = \psi(\delta_\Sigma(E,x))$, which is the wanted equality. ◀

▶ Remark 4.8 (analyticity). The previous theorem guarantees that formal power series solutions of a coherent system of PDEs always exist and are unique. In general, there is no guarantee of analyticity for such solutions. However, *if* a solution ψ of Σ in the usual sense exists that is analytic around the origin, *then* its Taylor expansion from the origin, seen as a CFPS, coincides with our solution $\phi_{\mathbf{B}}$: for each u and $f = \psi(u)$, we have that $f = \sum_{\boldsymbol{\alpha}}(\frac{\partial f}{\partial\tau})(0)\cdot\frac{\mathbf{x}^{\boldsymbol{\alpha}}}{\boldsymbol{\alpha}!} = \phi_{\mathbf{B}}(u)$. Riquier's theorem [24] gives sufficient syntactic conditions for the existence of analytic solutions; see [26, 16, 17] for further discussion of this point.

The computational content of Theorem 4.7 is twofold. One one hand, we can use coinduction as a technique to prove semantically valid identities $E = F$ for the initial value problem at hand, as bisimulations $E \sim_{\mathbf{B}} F$ (via Corollary 2.4). On the other hand, we can calculate mechanically the coefficients of the Taylor expansion of the solution. This will also be key to the algorithms presented in Section 5.

▶ **Corollary 4.9** (Taylor coefficients). *Let Σ be coherent, let ρ be an initial data specification and let $\phi_{\mathbf{B}}$ be the unique solution of $\mathbf{B} = (\Sigma, \rho)$. Then, for each $E \in \mathcal{P}$, $\phi_{\mathbf{B}}(E) = \sum_{\tau = \mathbf{x}^{\boldsymbol{\alpha}}} c_\tau \cdot \tau$ where*

$$c_\tau = \frac{\rho(\delta_\Sigma(E, \tau))}{\boldsymbol{\alpha}!} \,. \tag{6}$$

Proof. This follows immediately from Theorem 4.7 and from the definitions of unique morphism (2) and of the coalgebra $C_{\mathbf{B}}$. ◀

The terms $\frac{\delta_\Sigma(E,\tau)}{\boldsymbol{\alpha}!} \in \mathcal{P}_0(\Sigma)$ appearing in (6) provide a “symbolic” representation of the Taylor coefficients of solutions, independent of ρ.

▶ **Example 4.10.** Consider $U = \{f, g, i, j, h, k\}$, $X = \{x, y\}$ and $\Sigma = \{\, f_x = -g \,,\, f_y = -g \,,\, g_x = f \,,\, g_y = f \,,\, i_x = -j \,,\, i_y = 0 \,,\, j_x = i \,,\, j_y = 0 \,,\, h_x = 0 \,,\, h_y = -k \,,\, k_x = 0 \,,\, k_y = h \,\}$. Note that $\mathcal{P}\mathrm{a}(\Sigma) = U$. The system is consistent because Σ^∞ has just one equation for each $u_\tau \in \mathcal{P}\mathrm{r}(\Sigma)$. Moreover, it is normal, hence coherent, with respect to any graded ranking. Consider now the initial value problem $\mathbf{B} = (\Sigma, \rho)$ where ρ is defined by $\rho(f) = \rho(i) = \rho(h) = 1$ and $\rho(g) = \rho(j) = \rho(k) = 0$. Let $E \triangleq ih - jk$, $F \triangleq ik + jh$ and $R \subseteq \mathcal{P} \times \mathcal{P}$, $R \triangleq \{(f, E), (g, F), (-f, -E), (-g, -F)\}$: it is immediate to check that R is a bisimulation in $C_{\mathbf{B}}$. By coinduction and Theorem 4.7, we have therefore $\phi_{\mathbf{B}}(f) = \phi_{\mathbf{B}}(E)$ and $\phi_{\mathbf{B}}(g) = \phi_{\mathbf{B}}(F)$. Note that, in the given $\mathbf{B}$, the variables in U encode $\cos(x+y), \sin(x+y), \cos(x), \sin(x), \cos(y), \sin(y)$, respectively. Therefore e.g. $\phi_{\mathbf{B}}(f) = \phi_{\mathbf{B}}(E)$ actually proves that $\cos(x+y) = \cos(x)\cos(y) - \sin(x)\sin(y)$, a well-known trigonometric identity.

Finally, a more refined argument leads to a precise characterization of the systems that admit (unique) solutions for every initial data specification, under normality.

▶ **Theorem 4.11** (consistency, existence and uniqueness). *Let Σ be a normal system. Σ is coherent if and only if for each ρ, $\mathbf{B} = (\Sigma, \rho)$ has a solution. Moreover, for each such $\mathbf{B}$ the solution is unique.*

5 Equivalence checking

In this section, based on Theorem 4.7 and on an algebraic characterization of bisimilarity we shall discuss below, we will provide a decision algorithm for the equivalence problem $\phi_{\mathbf{B}}(E) = \phi_{\mathbf{B}}(F)$, limited to the following subclass of PDE systems.

▶ **Definition 5.1** (finite-parameter systems). *A system Σ is* finite-parameter *if $\mathcal{P}\mathrm{a}(\Sigma)$ is finite.*

For instance, with reference to Example 3.4, Σ_2 is finite-parameter, while Σ and Σ_1 are not. We need to introduce now some additional, mostly standard notation about polynomials. According to (6), the calculation of the Taylor coefficients of a solution of an initial value problem $\mathbf{B} = (\Sigma, \rho)$ involves evaluating expressions in $\mathcal{P}_0(\Sigma) = \mathbb{R}[X \cup \mathcal{P}\mathrm{a}(\Sigma)]$. As $k \triangleq |X \cup \mathcal{P}\mathrm{a}(\Sigma)| < +\infty$, elements of $\mathcal{P}_0(\Sigma)$ can be treated as usual multivariate polynomials in a finite number of indeterminates. In particular, we can identify initial data specifications

ρ with points in $\mathbb{R}^k$ that vanish in the x-coordinates ($x \in X$). In this section we will let ρ range over $\mathbb{R}^k$. Moreover, we let $\mathbb{R}^k_0 \stackrel{\triangle}{=} \{\rho \in \mathbb{R}^k \,:\, \rho(x) = 0$ for each $x \in X\}$ and, for polynomials $E \in \mathcal{P}_0(\Sigma)$ and initial data specifications $\rho \in \mathbb{R}^k_0$, write $\rho(E)$ as $E(\rho)$ – that is the value in $\mathbb{R}$ obtained by evaluating E at point ρ.

In what follows, we shall also make use a few elementary notions from algebraic geometry [10]. In particular, an *ideal* $J \subseteq \mathcal{P}_0(\Sigma)$ is a nonempty set of polynomials closed under addition, and under multiplication by polynomials in $\mathcal{P}_0(\Sigma)$. For $P \subseteq \mathcal{P}_0(\Sigma)$, $\langle\ P\ \rangle \stackrel{\triangle}{=} \{\sum_{i=1}^{m} G_i \cdot E_i \,:\, m \geq 0,\ G_i \in \mathcal{P}_0(\Sigma),\ E_i \in P\}$ denotes the smallest ideal which includes P, and $V(P) \subseteq \mathbb{R}^k$ the *(affine) variety* induced by P: $V(P) \stackrel{\triangle}{=} \{\rho \in \mathbb{R}^k \,:\, E(\rho) = 0$ for each $E \in P\} \subseteq \mathbb{R}^k$. For $W \subseteq \mathbb{R}^k$, $I(W) \stackrel{\triangle}{=} \{E \in \mathcal{P}_0(\Sigma) \,:\, E(\rho) = 0$ for each $\rho \in V\}$. We will use a few basic facts about ideals and varieties: (a) both $I(\cdot)$ and $V(\cdot)$ are inclusion reversing: $P_1 \subseteq P_2$ implies $V(P_1) \supseteq V(P_2)$ and $W_1 \subseteq W_2$ implies $I(W_1) \supseteq I(W_2)$; (b) any ascending chain of ideals $I_0 \subseteq I_1 \subseteq \cdots$ stabilizes in a finite number of steps (Hilbert's basis theorem); (c) for finite $P \subseteq \mathcal{P}_0(\Sigma)$, the problem of deciding if $E \in \langle\ P\ \rangle$ is decidable, by computing a Gröbner basis (a set of generators with special properties) of $\langle\ P\ \rangle$. See [10] for a comprehensive treatment.

Given a coherent, finite-parameter Σ and an initial data specification $\rho \in \mathbb{R}^k_0$, let us denote by $\phi_{\mathbf{B}}$ the unique solution of the initial value problem $\mathbf{B} = (\Sigma, \rho)$ (Theorem 4.7). Since by definition $\phi_{\mathbf{B}}$ is a homomorphism, for any given $E, F \in \mathcal{P}$, establishing that $\phi_{\mathbf{B}}(E) = \phi_{\mathbf{B}}(F)$ is equivalent to establish that $\phi_{\mathbf{B}}(E - F) = 0$. In other words, we can identify polynomial equations with polynomials, and valid polynomial equations under ρ with polynomials $E \in \mathcal{Z}_{\mathbf{B}} \subseteq \mathcal{P}$, where (below, 0 denotes the zero CFPS in $\mathcal{F}(X)$)

$$\mathcal{Z}_{\mathbf{B}} \stackrel{\triangle}{=} \phi_{\mathbf{B}}^{-1}(0)\,.$$

The equality problem reduces therefore to the *membership* problem for $\mathcal{Z}_{\mathbf{B}}$, for which we will now give an algorithm. In general terms, given $E \in \mathcal{P}$, suppose we want to decide if $E \in \mathcal{Z}_{\mathbf{B}}$. Note that, by virtue of Lemma 4.3, we can assume w.l.o.g. that $E \in \mathcal{P}_0(\Sigma)$. We shall rely mainly on Corollary 4.9. Consider now the chain of sets $A_0 \subseteq A_1 \subseteq \cdots \subseteq \mathcal{P}_0(\Sigma)$ defined as:

$$A_0 \stackrel{\triangle}{=} \{E\} \qquad\qquad A_{i+1} \stackrel{\triangle}{=} A_i \cup \{\delta_\Sigma(F, x) \,:\, F \in A_i,\, x \in X\}\,. \tag{7}$$

Let $m \geq 0$ be the least integer such that either: (a) there exists $F \in A_m$ s.t. $F(\rho) \neq 0$; or (b) no such $F \in A_m$ exists, but $A_{m+1} \subseteq I_m$, where, for each $i \geq 0$, $I_i \stackrel{\triangle}{=} \langle\ A_i\ \rangle$ is the ideal in $\mathcal{P}_0(\Sigma)$ generated by A_i. The algorithm returns "No" if (a) occurs, and "Yes" if (b) occurs. Note that the I_i's, $i \geq 0$, form an ascending chain of ideals in $\mathcal{P}_0(\Sigma)$, which must stabilize in a finite numbers of steps (by Hilbert's basis theorem). Moreover, the inclusion $A_{m+1} \subseteq I_m$ is decidable (by Gröbner basis construction). This ensures termination and effectiveness of the outlined algorithm. Concerning its correctness, we premise the following lemma, which implies that we can effectively detect stabilization of the sequence of the ideals I_i s.

▶ **Lemma 5.2.** *Let Σ be coherent and finite-parameter. Suppose $A_{m+1} \subseteq I_m$. Then $I_m = I_{m+j}$ for each $j \geq 1$.*

▶ **Corollary 5.3** (membership checking). *Let Σ be coherent and finite-parameter, ρ an initial data specification for Σ and $E \in \mathcal{P}_0(\Sigma)$. Then $\phi_{\mathbf{B}}(E) = 0$ if and only if the above algorithm returns "Yes".*

Proof. "No" is returned in case (a) occurs: as A_m consists precisely of all $\delta_\Sigma(E, \tau)$ such that $|\tau| \leq m$, this implies that $\phi_{\mathbf{B}}(E) \neq 0$, by virtue of (6). Assume on the other hand "Yes" is returned, that is case (b) of the algorithm occurs. Then by Lemma 5.2, $I_0 \subseteq \cdots \subseteq I_m =$

$I_{m+1} = I_{m+2} = \cdots$: therefore I_m contains in effect *every* $\delta_\Sigma(E, \tau)$ such that $\tau \in X^\otimes$. As ρ makes all polynomials in A_m vanish, by the definition of ideal ρ also makes all polynomials in $I_m = \langle\ A_m\ \rangle$ vanish. As a consequence, $(\delta_\Sigma(E, \tau))(\rho) = 0$ for all $\tau \in X^\otimes$, which, from Corollary 4.9, implies that $\phi_{\mathbf{B}}(E) = 0$. ◀

▶ **Example 5.4** (Burgers' equation)**.** Consider the following system, which is a special case of Burgers' equation [1, 9]

$$u_t \ = \ -u \cdot u_x \qquad u_x \ = \ \frac{1}{t+1}\,.$$

To code up this system, we fix $X = \{t, x\}$ and $U = \{u, v\}$, where v represents $1/(t+1)$, and let

$$\Sigma \ = \ \{u_t = -u \cdot u_x,\ u_x = v,\ v_t = -v^2,\ v_x = 0\}\,. \tag{8}$$

As $\mathcal{P}\mathrm{a}(\Sigma) = \{u, v\}$, the system is finite-parameter. Σ can be checked to be consistent – in particular there is a unique equation in Σ^∞ for u_{tx}, that is $u_{tx} = -v^2$. Moreover, with the lexicographic order induced by $u > v$ and $t > x$, Σ is coherent. Now fix $\rho(u) = \rho(v) = 1$ as an initial data specification and let $E \triangleq u - (x+1)v$. We check that $E = 0$ is a valid equation for the initial value problem $\mathbf{B} = (\Sigma, \rho)$, that is $E \in \mathcal{Z}_{\mathbf{B}}$, applying the above algorithm. We have: $A_0 = \{E\}$ and $A_1 = \{E\} \cup \{\delta_\Sigma(E, t), \delta_\Sigma(E, x)\} = \{E\} \cup \{-v \cdot E, 0\}$. As trivially $\{0, -v \cdot E\} \subseteq \langle\ \{E\}\ \rangle = I_0$, and $E(\rho) = (-v \cdot E)(\rho) = 0$, the algorithm returns "Yes". Note that the validity of $E = 0$ implies that $u = (x+1)v = \frac{x+1}{t+1}$, $v = \frac{1}{t+1}$ yield a solution of $\mathbf{B}$.

Finally, relying on the ascending chain (7), we devise a complete algorithm to compute the set of initial data specifications under which a given equation E is valid in Σ – so to speak, the weakest precondition of E.

▶ **Corollary 5.5** (weakest precondition)**.** *Let Σ be coherent and finite-parameter, $E \in \mathcal{P}_0(\Sigma)$. Let $I_{m+1} = I_m$. Then $\{\rho \in \mathbb{R}_0^k : \phi_{(\Sigma,\rho)}(E) = 0\} = V(X \cup A_m)$.*

▶ **Example 5.6** (Burgers' equation, continued)**.** Consider again the system Σ in (8) and $E = u - v \cdot (x+1)$. As $I_1 = I_0 = \langle\ \{E\}\ \rangle$, we see that the set of ρ's under which E is valid is $V(X \cup \{E\})$; that is, those ρ's such that $\rho(t) = \rho(x) = 0$ and $\rho(u) = \rho(v)$.

▶ Remark 5.7 (complexity). Procedures for computing Gröbner bases, such as Buchberger's algorithm, have a very high *worst*-case time and space complexity – exponential in the number of variables and the degree of the involved polynomials, see [10]. The number of steps m before the chain (7) stabilizes can also depend super-exponentially on the number of variables and their degrees, see [19]. This theoretical complexity is of course inherited by our algorithms. On the other hand, Gröbner basis algorithms have shown to behave well in many practical cases: this suggests that an assessment of our algorithms based on concrete case studies might be more informative than a purely complexity theoretical one. We leave a systematic exploration of this more pragmatical aspect for future work.

6 Conclusion, further and related work

We have put forward a coalgebraic framework for PDEs, that yields a clean proof of existence and uniqueness of solutions of initial value problems, and complete algorithms for checking equivalence of differential polynomial expressions, under a finite-parameters assumption. To the best of our knowledge, no such characterization and completeness results for PDEs exist in the literature. As for future work, we plan to explore extensions of the present results to *postconditions*: computing at once the most general, in a precise sense, consequences of a given algebraic set of initial data specifications. This would also permit to automatically discover valid equations, rather than just check given ones.

Conceptually, the present development parallels in part our previous work on polynomial ODEs [5, 7]. Technically, the case of PDEs is by far more challenging, for the following reasons. (a) Existence of solutions, and of the transition structure itself, depends now on coherence, which is trivial in ODEs. (b) In PDEs, differential polynomials live in the infinite-indeterminates space $\mathcal{P}$, which requires reduction to $\mathcal{P}_0(\Sigma)$ via S and a finiteness assumption on parametric derivatives; in ODEs, $\mathcal{P} = \mathcal{P}_0(\Sigma)$ always has finitely many indeterminates.

An operational view of functions and differential equations similar to ours has been considered elsewhere in the literature on coalgebras [20, 27]. In particular, it is at the basis of Rutten's calculus of *behavioural differential equations* [27]. In this calculus, neither PDEs nor equivalence algorithms are considered, though. Algorithms for equivalence checking are presented in [4, 3], limited to linear *weighted automata*: in terms of differential equations, these basically correspond to linear ODEs.

In the field of formal methods, we are aware of the work by Boldo et al. [2], who apply theorem proving to the formal verification of a numerical PDE integrator written in C. Platzer, in his work on differential hybrid games [22], relies on certain Hamilton-Jacobi type PDEs in order to define a solution concept for differential games. These works pursue goals rather different from ours, though.

Our work is also related to the field of Differential Algebra. In the classical exposition, coherent systems correspond to Riquier-Janet's *orthonomic passive* systems [24, 12], further developed by Thomas [30]. A modern presentation of orthonomic passive systems is in Marvan's [18]. A more geometrical approach is followed by Reid et al. [23, 26]. The work of Riquier, Janet and Thomas is the root of what is nowadays known as Ritt-Kolchin's Differential Algebra (DA) [25, 14]. Recent developments of DA include the work by the French school, especially Boulier et al., see e.g. [8, 16]. The relationship of our coalgebraic framework with DA is not yet entirely clear and deserves further investigation.

References

1 H. Bateman. Some recent researches on the motion of fluids. *Monthly Weather Review*, 43(4):163–170, 1915.

2 S. Boldo, F. Clément, J.-C. Filliâtre, M. Mayero, G. Melquiond, and P. Weis. Trusting Computations: a Mechanized Proof from Partial Differential Equations to Actual Program. *Computers and Mathematics with Applications*, 68(3):325–352, 2014.

3 F. Bonchi, M. M. Bonsangue, M. Boreale, J. J. M. M. Rutten, and A. Silva. A coalgebraic perspective on linear weighted automata. *Inf. Comput.*, 211:77–105, 2012.

4 M. Boreale. Weighted Bisimulation in Linear Algebraic Form. In *CONCUR 2009*, volume 5710 of *LNCS*, pages 163–177. Springer, 2009.

5 M. Boreale. Algebra, coalgebra, and minimization in polynomial differential equations. In *FoSSACS 2017*, volume 10203 of *LNCS*, pages 71–87. Springer, 2017. Full version in *Logical Methods in Computer Science* 15(1) arXiv.org:1710.08350, 2019.

6 M. Boreale. Algorithms for exact and approximate linear abstractions of polynomial continuous systems. In *HSCC 2018*, pages 207–216. ACM, 2018.

7 M. Boreale. Complete algorithms for algebraic strongest postconditions and weakest preconditions in polynomial ODE's. In SOFSEM 2018: Theory and Practice of Computer Science - 44th International Conference on Current Trends in Theory and Practice of Computer Science, volume 10706 of *LNCS*, pages 442–455. Springer, 2018.

8 F. Boulier, D. Lazard, F. Ollivier, and M. Petitot. Computing representations for radicals of finitely generated differential ideals. *Appl. Algebra Engrg. Comm. Comput.*, 20(1):73–121, 2009.

9 J. M. Burgers. A mathematical model illustrating the theory of turbulence. In *Advances in applied mechanics*, volume 1, pages 171–199. Elsevier, 1948.

10 D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra.* Undergraduate Texts in Mathematics. Springer, 2007.

11 K. Ghorbal and A. Platzer. Characterizing Algebraic Invariants by Differential Radical Invariants. In *TACAS 2014*, volume 8413 of *LNCS*, pages 279–294. Springer, 2014. URL: http://reports-archive.adm.cs.cmu.edu/anon/2013/CMU-CS-13-129.pdf.

12 M. Janet. *Sur les systèmes d'équations aux dérivées partielles.* Thèses françaises de l'entre-deux-guerres. Gauthiers-Villars, Paris, 1920. URL: http://www.numdam.org/item?id=THESE_1920__19__1_0.

13 D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. W. Leech, editor, *Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967)*, pages 263–297. Pergamon, Oxford, 1970.

14 E. R. Kolchin. *Differential algebra and algebraic groups*, volume 54 of *Pure and Applied Mathematics*. Academic Press, New York-London, 1973.

15 H. Kong, S. Bogomolov, Ch. Schilling, Yu Jiang, and Th.A. Henzinger. Safety Verification of Nonlinear Hybrid Systems Based on Invariant Clusters. In *HSCC 2017*, pages 163–172. ACM, 2017.

16 F. Lemaire. *Contribution à l'algorithmique en algèbre différentielle.* Génie logiciel [cs.SE]. Université des Sciences et Technologie de Lille - Lille, 2002. URL: https://tel.archives-ouvertes.fr/tel-00001363/document.

17 F. Lemaire. An Orderly Linear PDE System with Analytic Initial Conditions with a Non-analytic Solution. *J. Symb. Comput.*, 35(5):487–498, May 2003. doi:10.1016/S0747-7171(03)00017-8.

18 M. Marvan. Sufficient Set of Integrability Conditions of an Orthonomic System. *Foundations of Computational Mathematics*, 9(6):651–674, 2009.

19 D. Novikov and S. Yakovenko. Trajectories of polynomial vector fields and ascending chains of polynomial ideals. *Annales de l'Institut Fourier*, 49(2):563–609, 1999. doi:10.5802/aif.1683.

20 D. Pavlovic and M.H. Escardó. Calculus in Coinductive Form. In *LICS 1998*, pages 408–417. IEEE, 1998.

21 A. Platzer. Logics of dynamical systems. In *LICS 2012*, pages 13–24. IEEE, 2012.

22 A. Platzer. Differential hybrid games. *ACM Trans. Comput. Log.*, 18(3):19–44, 2017.

23 G. Reid, A. Wittkopf, and A. Boulton. Reduction of systems of nonlinear partial differential equations to simplified involutive forms. *European Journal of Applied Mathematics*, 7(6):635–666, 1996.

24 C. Riquier. *Les systèmes d'équations aux dérivèes partielles.* Gauthiers-Villars, Paris, 1910.

25 J. F. Ritt. *Differential Algebra*, volume XXXIII. American Mathematical Society Colloquium Publications, American Mathematical Society, New York, N. Y, 1950.

26 C. J. Rust, G. J. Reid, and A. D. Wittkopf. Existence and Uniqueness Theorems for Formal Power Series Solutions of Analytic Differential Systems. In *ISSAC 1999*, pages 105–112, 1999.

27 J. J. M. M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theoretical Computer Science*, 308(1–3):1–53, 2003.

28 S. Sankaranarayanan. Automatic invariant generation for hybrid systems using ideal fixed points. In *HSCC 2010*, pages 221–230. ACM, 2010.

29 S. Sankaranarayanan, H. Sipma, and Z. Manna. Non-linear loop invariant generation using Gröbner bases. In *POPL 2004*. ACM, 2004.

30 J. M. Thomas. *Differential Systems*, volume XXI. American Mathematical Society Colloquium Publications, American Mathematical Society, New York, N. Y, 1937.

Random Subgroups of Rationals

Ziyuan Gao
Department of Mathematics, National University of Singapore, Singapore
matgaoz@nus.edu.sg

Sanjay Jain
School of Computing, National University of Singapore, Singapore
sanjay@comp.nus.edu.sg

Bakhadyr Khoussainov
Department of Computer Science, University of Auckland, New Zealand
bmk@cs.auckland.ac.nz

Wei Li
Department of Mathematics, National University of Singapore, Singapore
matliw@nus.edu.sg

Alexander Melnikov
Institute of Natural and Mathematical Sciences, Massey University, New Zealand
A.Melnikov@massey.ac.nz

Karen Seidel
Hasso Plattner Institute, University of Potsdam, Germany
karen.seidel@hpi.uni-potsdam.de

Frank Stephan
Department of Mathematics, National University of Singapore, Singapore
fstephan@comp.nus.edu.sg

Abstract

This paper introduces and studies a notion of *algorithmic randomness* for subgroups of rationals. Given a randomly generated additive subgroup $(G, +)$ of rationals, two main questions are addressed: first, what are the model-theoretic and recursion-theoretic properties of $(G, +)$; second, what learnability properties can one extract from G and its subclass of finitely generated subgroups? For the first question, it is shown that the theory of $(G, +)$ coincides with that of the additive group of integers and is therefore decidable; furthermore, while the word problem for G with respect to any generating sequence for G is not even semi-decidable, one can build a generating sequence β such that the word problem for G with respect to β is co-recursively enumerable (assuming that the set of generators of G is limit-recursive). In regard to the second question, it is proven that there is a generating sequence β for G such that every non-trivial finitely generated subgroup of G is recursively enumerable and the class of all such subgroups of G is behaviourally correctly learnable, that is, every non-trivial finitely generated subgroup can be semantically identified in the limit (again assuming that the set of generators of G is limit-recursive). On the other hand, the class of non-trivial finitely generated subgroups of G cannot be syntactically identified in the limit with respect to any generating sequence for G. The present work thus contributes to a recent line of research studying algorithmically random infinite structures and uncovers an interesting connection between the arithmetical complexity of the set of generators of a randomly generated subgroup of rationals and the learnability of its finitely generated subgroups.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization; Theory of computation → Inductive inference

Keywords and phrases Martin-Löf randomness, subgroups of rationals, finitely generated subgroups of rationals, learning in the limit, behaviourally correct learning

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.25

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 25; pp. 25:1–25:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of the paper is available at `https://hal.archives-ouvertes.fr/hal-01970683v2/document`

Funding Sanjay Jain was supported in part by NUS grant C252-000-087-001. Furthermore, Ziyuan Gao, Sanjay Jain and Frank Stephan have been supported in part by the Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2016-T2-1-019 / R146-000-234-112. Bakhadyr Khoussainov was supported by the Marsden fund of Royal Society of New Zealand. Karen Seidel was supported by the German Research Foundation (DFG) under Grant KO 4635/1-1 (SCL) and by the Marsden fund of Royal Society of New Zealand.

Acknowledgements The authors would like to thank Philipp Schlicht and Tin Lok Wong for helpful discussions, as well as thank Timo Kötzing for pointers to the literature.

1 Introduction

The concept of *algorithmic randomness*, particularly for strings and infinite sequences, has been extensively studied in recursion theory and theoretical computer science [6, 15, 19]. It has also been applied in a wide variety of disciplines, including formal language and automata theory [14], machine learning [29], and recently even quantum theory [20]. An interesting and long open question is whether the well-established notions of randomness for infinite sequences have analogues for infinite structures such as graphs and groups. Intuitively, it might be reasonable to expect that a collection of random infinite structures possesses the following characteristics: (1) randomness should be an isomorphism invariant property; in particular, random structures should not be computable; (2) the collection of random structures (of any type of algebraic structure) should have cardinality equal to that of the continuum. The standard random infinite graph thus does not qualify as an algorithmically random structure; in particular, it is isomorphic to a computable graph and has a countable categorical theory. Very recently, Khoussainov [12, 13] defined algorithmic randomness for infinite structures that are akin to graphs, trees and finitely generated structures.

This paper addresses the following three open questions in algorithmic randomness: (A) is there a reasonable way to define algorithmically random structures for standard algebraic structures such as groups; (B) can one define algorithmic randomness for groups that are not necessarily finitely generated; (C) what are the model-theoretic properties of algorithmically random structures? The main contribution of the present paper is to answer the first two questions positively for a fundamental and familiar algebraic structure, the *additive group of rationals*, denoted $(\mathbb{Q}, +)$, and to answer the third question with respect to this structure. Prior to this work, question (A) was answered for structures such as *finitely generated* universal algebras, connected graphs, trees of bounded degree and monoids [12]. Concerning question (C), it is still unknown whether the first order theory of algorithmically random graphs (or trees) is decidable. In fact, it is not even known whether any two algorithmically random graphs (of the same bounded degree) are elementarily equivalent [12].

As mentioned earlier, one goal of this work is to formulate a notion of randomness for subgroups of $(\mathbb{Q}, +)$. This is a fairly natural class of groups to consider, given that the isomorphism types of its subgroups have been completely classified, as opposed to the current limited state of knowledge about the isomorphism types of even rank 2 groups. As has been known since the work of Baer [1], the subgroups of $(\mathbb{Q}, +)$ coincide, up to isomorphism, with the torsion-free Abelian groups of rank 1. Moreover, the group $(\mathbb{Q}, +)$ is robust enough that it has uncountably many algorithmically random subgroups (according to our definition of algorithmically random subgroups of $(\mathbb{Q}, +)$), which contrasts with the fact that there is a unique standard random graph up to isomorphism. At the same time, the algorithmically random subgroups of $(\mathbb{Q},+)$ are not too different from one other in the sense that they are all elementarily equivalent (a fact that will be proven later), which is similar to the case of standard random graphs being elementarily equivalent.

The properties of the subgroups of $(\mathbb{Q},+)$ were first systematically studied by Baer [1] and then later by Beaumont and Zuckerman [3]. Later, the group $(\mathbb{Q},+)$ was studied in the context of automatic structures [28]. An early definition of a random group is due to Gromov [10]. According to this definition, random groups are those obtained by first fixing a set of generators, and then randomly choosing (according to some probability distribution) the relators specifying the quotient group. An alternative definition of a general random infinite structure was proposed by Khoussainov [12, 13]; this definition is based on the notion of a *branching class*, which is in turn used to define Martin-Löf tests for infinite structures entirely in analogy to the definition of a Martin-Löf test for sequences. An infinite structure is then said to be Martin-Löf random if it passes every Martin-Löf test in the preceding sense.

Like Gromov's definition of a random group, the one adopted in the present work is syntactic, in contrast to the semantic and algebraic definition due to Khoussainov. However, rather than selecting the relators at random according to a prescribed probability distribution for a fixed set of generators, our approach is to directly encode a Martin-Löf random binary sequence into the generators of the subgroup. More specifically, we fix any binary sequence R, and define the group G_R as that generated by all rationals of the shape $p_i^{-n_i}$, where p_i denotes the $(i+1)$-st prime and n_i is the number of ones occurring between the i-th and $(i+1)$-st occurrences of zero in R; n_0 is the number of starting ones, and if there is no $(i+1)$-st zero then n_j is defined to be zero for all j greater than i and G_R is generated by all $p_{i'}^{-n_{i'}}$ with i' less than i and all $p_i^{-n'}$ such that n' is any positive integer. G_R is then said to be *randomly generated* if and only if R is Martin-Löf random. In order to derive certain computability properties, it will always be assumed in the present paper that any Martin-Löf random sequence associated to a randomly generated subgroup of $(\mathbb{Q},+)$ is also limit-recursive. It may be observed that no finitely generated subgroup of $(\mathbb{Q},+)$ is randomly generated in the sense adopted here; this corresponds to the intuition that in any "random" infinite binary sequence R, the fraction of zeroes in the first n bits should tend to a number strictly smaller than one as n grows to infinity.

The first main part of this work is devoted to the study of the model-theoretic and recursion-theoretic properties of randomly generated subgroups of $(\mathbb{Q},+)$. It is shown that the theory of any randomly generated subgroup coincides with that of the integers with addition (denoted $(\mathbb{Z},+)$), and is therefore decidable[1]. Next, we define the notion of a *generating sequence* for a randomly generated group G_R; this is an infinite sequence β such that G_R is generated by the terms of β. We then consider the word problem for G_R with respect to β: this is the problem of determining, given any two finite integer sequence representations σ and τ of elements of G_R with respect to β, whether or not σ and τ represent the same element of G_R. We show that the word problem for G_R with respect to *any* generating sequence β is never recursively enumerable (r.e.); on the other hand, one can construct a generating sequence β' for G_R such that the corresponding word problem for G_R is co-r.e. Moreover, one can build a generating sequence β'' for G_R such that the word problem for the quotient group of G_R by $\mathbb{Z}$ with respect to β'' is r.e.

The second main part of this paper investigates the learnability of non-trivial finitely generated subgroups of randomly generated subgroups of $(\mathbb{Q},+)$ from positive examples, also

[1] For a proof of the decidability of the theory of $(\mathbb{Z},+)$, often known as *Presburger Arithmetic*, see [16, pages 81–84].

known as learning from text. Stephan and Ventsov [25] examined the learnability of classes of substructures of algebraic structures; the study of more general classes of structures was undertaken in the work of Martin and Osherson [17, Chapter III]. The general objective is to understand how semantic knowledge of a class of concepts can be exploited to learn the class; in the context of the present problem, semantic knowledge refers to the properties of every finitely generated subgroup of any randomly generated subgroup of rationals, such as being generated by a single rational [1]. It may be noted that the present work considers learning of the actual representations of finitely generated subgroups, as opposed to learning their structures up to isomorphism, as is considered in the learning framework of Martin and Osherson [17]. Various positive learnability results are obtained: it will be proven, for example, that for any randomly generated subgroup G_R of $(\mathbb{Q}, +)$, there is a generating sequence β for G_R such that the set of representations of every non-trivial finitely generated subgroup of G_R with respect to β is r.e.; furthermore, the class of all such representations can be identified in the limit up to semantic equivalence. On the other hand, it will be seen that the class of all such representations can never be learnable in the limit. Similar results hold for the class of non-trivial finitely generated subgroups of the quotient group of G_R by $\mathbb{Z}$. Thus this facet of our work implies a connection between the limit-recursiveness of the set of generators of a randomly generated subgroup of $(\mathbb{Q}, +)$ and the learnability of its non-trivial finitely generated subgroups.

2 Preliminaries

Any unexplained recursion-theoretic notation may be found in [22, 24, 21]. For background on algorithmic randomness, we refer the reader to [6, 19]. We use $\mathbb{N} = \{0, 1, 2, \ldots\}$ to denote the set of all natural numbers and $\mathbb{Z}$ to denote the set of all integers. The $(i+1)$-st prime will be denoted by p_i. $\mathbb{Z}^{<\omega}$ denotes the set of all finite sequences of integers. Throughout this paper, $\varphi_0, \varphi_1, \varphi_2, \ldots$ is a fixed acceptable programming system of all partial recursive functions and $W_0, W_1, W_2, \ldots$ is a fixed *acceptable numbering of all recursively enumerable* (abbr. r.e.) *sets* of natural numbers. We will occasionally work with objects belonging to some countable class X different from $\mathbb{N}$; in such a case, by abuse of notation, we will use the same symbol W_e to denote the set of objects obtained from W_e by replacing each member x with $F(x)$ for some fixed bijection F between $\mathbb{N}$ and X.

Given any set S, S^* denotes the set of all finite sequences of elements from S. By $D_0, D_1, D_2, \ldots$ we denote any fixed *canonical indexing of all finite sets* of natural numbers. Cantor's pairing function $\langle \cdot, \cdot \rangle \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is given by $\langle x, y \rangle = \frac{1}{2}(x+y)(x+y+1) + y$ for all $x, y \in \mathbb{N}$. The symbol K denotes the *diagonal halting problem*, i.e., $K = \{e \mid e \in \mathbb{N},\ \varphi_e(e) \text{ converges}\}$. The *jump* of K, that is, the relativised halting problem $\{e \mid e \in \mathbb{N}; \varphi_e^K(e)\downarrow\}$, will be denoted by K'.

For $\sigma \in (\mathbb{N} \cup \{\#\})^*$ and $n \in \mathbb{N}$ we write $\sigma(n)$ to denote the element in the n-th position of σ. Further, $\sigma[n]$ denotes the sequence $\sigma(0), \sigma(1), \ldots, \sigma(n-1)$. Given a number $a \in \mathbb{N}$ and some fixed $n \in \mathbb{N}$, $n \geq 1$, we denote by a^n the finite sequence $a, \ldots, a$, where a occurs exactly n times. Moreover, we identify a^0 with the empty string ε. For any finite sequence σ we use $|\sigma|$ to denote the length of σ. The concatenation of two sequences σ and τ is denoted by $\sigma \circ \tau$; for convenience, and whenever there is no possibility of confusion, this is occasionally denoted by $\sigma\tau$. For any sequence β (infinite or otherwise) and $s < |\beta|$, $\beta\restriction_s$ denotes the initial segment of β of length $s+1$. For any $m \geq 1$ and $p \in \mathbb{Z}$, $I_m(p)$ denotes the vector of length m whose first $m-1$ coordinates are 0 and whose last coordinate is p. Furthermore, given two vectors $\alpha = (a_i)_{0 \leq i \leq m}$ and $\beta = (b_i)_{0 \leq i \leq m}$ of equal length, $\alpha \cdot \beta$ denotes the scalar product of α and β, that is, $\alpha \cdot \beta := \sum_{i=0}^{m} a_i b_i$. For any $c \in \mathbb{Z}$ and $\sigma := (b_i)_{0 \leq i \leq m} \in \mathbb{Z}^{<\omega}$, $c\sigma$ denotes the vector obtained from σ by coordinatewise multiplication with c, that is, $c\sigma := (cb_0, cb_1, \ldots, cb_m)$. For any non-empty $S \subseteq \mathbb{Q}$, $\langle S \rangle$ denotes $\{\sum_{i=0}^{k} c_i s_i \mid k \in \mathbb{N} \wedge c_i \in \mathbb{Z} \wedge s_i \in S\}$.

Cantor space, the set of all infinite binary sequences, will be denoted by 2^ω. The set of finite binary strings will be denoted by $2^{<\omega}$. For any binary string σ, $[\sigma]$ denotes the cylinder generated by σ, that is, the set of infinite binary sequences with prefix σ. For any $U \subseteq 2^{<\omega}$, the open set generated by U is $[U] := \bigcup_{\sigma \in U} [\sigma]$. The Lebesgue measure on 2^ω will be denoted by λ; that is, for any binary string σ, $\lambda([\sigma]) = 2^{-|\sigma|}$. By the Carathéodory Theorem, this uniquely determines the Lebesgue measure on the Cantor space.

3 Randomly Generated Subgroups of Rationals

We first review some basic definitions and facts in algorithmic randomness which in our setting is always understood w.r.t the Lebesgue measure. An *r.e. open set* R is an open set generated by an r.e. set of binary strings. Regarding W_e as a subset of $2^{<\omega}$, one has an enumeration $[W_0], [W_1], [W_2], \ldots$ of all r.e. open sets. A *uniformly r.e. sequence* $(G_m)_{m<\omega}$ *of open sets* is given by a recursive function f such that $G_m = [W_{f(m)}]$ for each m. As infinite binary sequences may be viewed as characteristic functions of subsets of $\mathbb{N}$, we will often use the term "set" interchangeably with "infinite binary sequence"; in particular, the subsequent definitions apply equally to subsets of $\mathbb{N}$ and infinite binary sequences.

Martin-Löf [18] defined randomness based on tests. A *Martin-Löf test* is a uniformly r.e. sequence $(G_m)_{m<\omega}$ of open sets such that $(\forall m < \omega)[\lambda(G_m) \leq 2^{-m}]$. A set $Z \subseteq \mathbb{N}$ *fails* the test if $Z \in \bigcap_{m<\omega} G_m$; otherwise Z *passes* the test. Z is *Martin-Löf random* if Z passes each Martin-Löf test. Schnorr [23] showed that Martin-Löf random sets can be described via martingales. A *martingale* is a function $\mathrm{mg} : 2^{<\omega} \to \mathbb{R}^+ \cup \{0\}$ that satisfies for every $\sigma \in 2^{<\omega}$ the equality $\mathrm{mg}(\sigma \circ 0) + \mathrm{mg}(\sigma \circ 1) = 2\mathrm{mg}(\sigma)$. For a martingale mg and a set Z, the martingale mg *succeeds* on Z if $\sup_n \mathrm{mg}(Z(0) \ldots Z(n)) = \infty$.

▶ **Theorem 1.** *[23] For any set Z, Z is Martin-Löf random iff no r.e. martingale succeeds on Z.*

The following characterisation of all subgroups of $(\mathbb{Q}, +)$ forms the basis of our definition of a random subgroup.

▶ **Theorem 2.** *[3] Let G be any subgroup of $(\mathbb{Q}, +)$. Then there is an integer z, as well as a sequence $(n_i)_{i<\omega}$ with $n_i \in \mathbb{N} \cup \{\infty\}$ such that $G = \left\{ \frac{a \cdot z}{\Pi_{i=0}^{k} p_i^{m_i}} \mid a \in \mathbb{Z} \wedge k \in \mathbb{N} \wedge (\forall i \leq k)[m_i \in \mathbb{N} \wedge m_i < n_i] \right\}$.*

▶ **Definition 3.** *Let $R \in 2^\omega$ be a real in the Cantor space, i.e. an infinite sequence of 0's and 1's. Then the group G_R is the subgroup of the rational numbers $(\mathbb{Q}, +)$ generated by $a_0, a_1, \ldots$ with $a_i = \frac{1}{p_i^{n_i}}$ for all $i \in \mathbb{N}$, where for each $i \in \mathbb{N}$, by p_i we denote the $(i+1)$-st prime and by n_i the number of consecutive 1's in R between the i-th and $(i+1)$-st zero in R, with which we let n_0 count the number of starting 1's. If there is no $(i+1)$-st zero, we let $n_i := \infty$, meaning that for all n the fraction $\frac{1}{p_i^n}$ is in G_R.*

Clearly, $(\mathbb{Z}, +)$ is always a subgroup of G_R and $\frac{1}{p_i} \notin G_R$ if and only if the i-th and $(i+1)$-st zero in R are consecutive. Thus, if R ends with infinitely many zeros, then G_R is isomorphic to $(\mathbb{Z}, +)$. Moreover, there is a prime p_i such that $\frac{1}{p_j} \notin G_R$ for all $j > i$ and $\frac{1}{p_i^n} \in G_R$ for all $n \in \mathbb{N}$, for short p_i infinitely divides G_R, if and only if R ends with an infinite sequence of 1's.

▶ **Lemma 4.** *If $R \in 2^\omega$ is Martin-Löf random, then n_i is finite for every $i \in \mathbb{N}$, where n_i is defined as in Definition 3. In other words, the group G_R is not infinitely divisible by any prime.*

Proof. This is an easy observation, as in no Martin-Löf random w.r.t the Lebesgue measure only finitely many 0's occur. ◀

A similar argument shows that for Martin-Löf random R there are infinitely many primes occurring as basis of a denominator of a generator.

▶ **Definition 5.** *Fix a probability distribution μ on the natural numbers and let $X = (X_i)_{i\in\mathbb{N}}$ be a sequence of iid random variables taking values in $\mathbb{N}$ with distribution $X_i \sim \mu$ for all $i \in \mathbb{N}$. Denote by H_X the subgroup of $(\mathbb{Q}, +)$ generated by $\{p_i^{-X_i} \mid i \in \mathbb{N}\}$, where p_i denotes the $(i+1)$-st prime.*

The so obtained random group might follow a more uniform process.

▶ **Lemma 6.** *If μ is the distribution on $\mathbb{N}$ assigning 0 probability $\frac{1}{2}$, 1 probability $\frac{1}{4}$, 2 probability $\frac{1}{8}$ and n probability 2^{-n-1}, then with probability 1 holds $H_X = G_R$ for some Martin-Löf random R.*

Proof. This follows immediately, as the set of ML-randoms has measure 1 with respect to the Lebesgue measure. From $X_0 = n_0$, $X_1 = n_1$, …, $X_i = n_i$, … we obtain an infinite binary sequence $R \in 2^\omega$ by recursively appending $1^{n_i}0$ in step i to the already established initial segment of R, starting with the empty string. By definition the Lebesgue measure assigns probability $\frac{1}{2^{n+1}}$ to having the (intermediate) subsequence 1^n0 in R. This is exactly the probability of the event $X_i = n$. ◀

A *generating sequence for G_R* is an infinite sequence $(b_i)_{i<\omega}$ such that $\langle b_i \mid i < \omega\rangle = G_R$. We will often deal with generating sequences rather than minimal generating sets for G_R, mainly due to the fact that if the terms of a sequence β are carefully chosen based on a limiting recursive programme for R (so that β itself is limiting recursive), then, as will be seen later, the set of representations of elements of G_R with respect to β can have certain desirable computability properties, such as equality being co-r.e.

▶ **Proposition 7.** *Suppose $R \leq_T K$ is Martin-Löf random. Then there does not exist any strictly increasing recursive enumeration $i_0, i_1, i_2, \ldots$ such that for each j, there is some $n_{i_j} \geq 1$ with $p_{i_j}^{-n_{i_j}} \in G_R$.*

▶ **Theorem 8.** *If $R \leq_T K$ is Martin-Löf random, then $(G_R, +)$ is co-r.e., meaning that $+$ is recursive and there is a generating sequence with respect to which equality is co-r.e.*

Proof. For a fixed generating sequence $(q_i)_{i<\omega}$ of G_R there is an epimorphism from the set of finite sequences of integers $\mathbb{Z}^{<\omega}$ to G_R by identifying $\sigma = (\sigma(0), \ldots, \sigma(|\sigma|-1))$ with $x = \sum_{i=0}^{|\sigma|-1} \sigma(i)q_i$. We call σ a representation of x w.r.t. $(q_i)_{i<\omega}$ or $(q_i)_{i<n+1}$.

Obviously, for any generating sequence of G_R addition is recursive as only the components of the representations have to be added as integers.

In order to prove that equality is co-r.e., we construct a specific generating sequence $(b_i)_{i<\omega}$. Based on the result R^s of the computation of R after s steps, we are going to define finite sequences β_s of rational numbers recursively, such that $|\beta_s| = s+1$ and inequality on $\{-s-1, \ldots, s+1\}^{s+1} \subseteq \mathbb{Z}^{s+1}$, interpreted as representations w.r.t. β_s, is decided and extends the inequalities on $\{-s, \ldots, s\}^s$, even though they originate from an interpretation as representations according to β_{s-1}. With this in the limit we obtain a generating sequence of G_R, meaning that for every i there is some $s_i > i$ such that for all $s \geq s_i$ the i-th element of β_s is the same as the i-th element of β_{s_i}, which we denote by b_i. Further, $(b_i)_{i\in\mathbb{N}}$ generates G_R and for this generating sequence equality will be co-r.e.

In the following we write $n_{i,s}$ for n_i according to R^s, i.e. the number of 1's between the i-th and $(i+1)$-st zero in R^s, as introduced in Definition 3. As R^s does not end with infinitely many 1's, $n_{i,s}$ can be computed in finitely many steps for every i and s.

$s = 0$. Let $\beta_0 = (1)$.

$s \rightsquigarrow s+1$. Check for every $i \leq s$ whether $n_{i,s} = n_{i,s+1}$. If $n_{i,s} = n_{i,s+1}$ let $\beta_{s+1}(i) = \beta_s(i)$. Replace all $\frac{1}{p_i^{n_{i,s}}}$ occurring in β_s with $n_{i,s} \neq n_{i,s+1}$ by some respective integer, for which existence we argue below, such that

$$\Delta_{(q_i)_{i<s+1}} = \{\, (\sigma_0, \sigma_1) \in (\,\{-s-1, \ldots, s+1\}^{s+1}\,)^2 \mid \\ \sigma_0, \sigma_1 \text{ represent different elements w.r.t. } (q_i)_{i<s+1} \,\}$$

stays the same or enlarges if $(q_i)_{i<s+1}$ equals the first $s+1$ entries of β_{s+1} instead of β_s. Further, let

$$\beta_{s+1}(s+1) = \frac{1}{p_j^{n_{j,s+1}}},$$

where $j \leq s+1$ is minimal such that $\frac{1}{p_j^{n_{j,s+1}}}$ is an element of $G_{R^{s+1}}$ and does not yet occur in $\beta_{s+1}\restriction(s+1)$. If there is no such j, let $\beta_{s+1}(s+1) = 1$.

For example, if the tape after stage $s = 2$ started with $1111010\ldots$, after 3 steps contained $1101010\ldots$ and $\beta_2 = (1, \frac{1}{2^4}, \frac{1}{3})$, then in β_3 we would have to replace $\frac{1}{2^4}$ by an integer w such that for arbitrary integers $u_0, u_1, u_2, v_0, v_1, v_2$ between -3 and 3 we have

$$u_0 + u_1\frac{1}{2^4} + u_2\frac{1}{3} \neq v_0 + v_1\frac{1}{2^4} + v_2\frac{1}{3} \quad \Rightarrow \quad u_0 + u_1 w + u_2\frac{1}{3} \neq v_0 + v_1 w + v_2\frac{1}{3}$$

and $\beta_3(3)$ would be $\frac{1}{2^2}$.

We proceed by showing that there is always such an integer w.

▷ Claim 9. For every $s \in \mathbb{N}$ in step $s+1$ it is possible to alter finitely many entries of β_s to obtain $\beta_{s+1}\restriction(s+1)$ such that $\Delta_{\beta_s} \subseteq \Delta_{\beta_{s+1}\restriction(s+1)}$.

Proof of the Claim. Let $s \in \mathbb{N}$. It suffices to show that one entry can be replaced in this desired way. As the argument does not depend on the position, we further assume that it is the last entry. For all $(\sigma_0, \sigma_1) \in \Delta_{\beta_s}$ we want to prevent

$$\sum_{i=0}^{s-1} \sigma_0(i)\beta_s(i) + \sigma_0(s)w = \sum_{i=0}^{s-1} \sigma_1(i)\beta_s(i) + \sigma_1(s)w.$$

This is a linear equation having zero or one solution in $\mathbb{Q}$. As there are only finitely many choices for the pair (σ_0, σ_1), an integer not fulfilling any of these equations can be found in a computable way. ◁

We continue by proving that the entries of the β_s stabilize, such that in the limit we obtain a sequence $(b_i)_{i<\omega}$ of elements of G_R.

▷ Claim 10. For every $i \in \mathbb{N}$ there is some $s_i \geq i$ such that for all $s \geq s_i$ we have $\beta_s(i) = b_i$, with $b_i = \beta_{s_i}(i)$.

Proof of the Claim. Let $i \in \mathbb{N}$. If there is $s_i > i$ such that the entry $\beta_{s_i-1}(i)$ had to be changed, then $\beta_{s_i}(i)$ is an integer and thus, it will never be changed lateron. In case this does not happen, we obtain $\beta_s(i) = \beta_i(i)$ for all $s \geq i$ and therefore $s_i = i$. ◁

By the next claim the just constructed sequence generates the random group.

▷ Claim 11. The sequence $(b_i)_{i<\omega}$ generates G_R.

Proof of the Claim. Let $i \in \mathbb{N}$ and a_i as in Definition 3. We argue that there is some j with $a_i = b_j$. Let m_i be the position of the $(i+1)$-st zero in the Martin-Löf random R. Then there is s' such that after s' computation steps $R \upharpoonright (m_i + 1)$ is not changed any more. Thus, after at most i additional steps all generators of G_R having one of the first i primes as denominator are in the range of $\beta_{s'+i}$. ◁

Finally, we observe that w.r.t. the generating sequence $(b_i)_{i<\omega}$ all pairs of unequal elements of G_R can be recursively enumerated.

▷ Claim 12. Equality in $(G_R, +)$ is co-r.e.

Proof of the Claim. We run the algorithm generating $(b_i)_{i<\omega}$ and in step s return all elements of the finite set Δ_{β_s}. As inequalities w.r.t β_s yield inequalities w.r.t. $(b_i)_{i<\omega}$, we only enumerate correct information. Further, for every two elements x, y of G_R fix representations w.r.t. $(b_i)_{i<\omega}$ and s' large enough such that not more than the first s' of the b_i occur in these representations, all of these have stabilized up to stage s' and all coefficients in the representations take values between $-s' - 1$ and $s' + 1$. Then $x \neq y$ if and only if the tuple of their representations is in $\Delta_{\beta_{s'}}$. ◁

This finishes the proof of the theorem. ◀

As there are K-recursive Martin-Löf random reals, we obtain the following corollary.

▶ **Corollary 13.** *There exists a co-r.e. random subgroup of the rational numbers.*

▶ Remark 14. Proposition 7 implies, in particular, that if $R \leq_T K$ is Martin-Löf random, then there cannot exist any generating sequence for G_R with respect to which equality of members of G_R is r.e. Indeed, suppose that such a generating sequence β did exist, so that $E := \{(\sigma, \tau) \in \mathbb{Z}^{<\omega} \times \mathbb{Z}^{<\omega} \mid \sigma \cdot \beta \upharpoonright_{|\sigma|-1} = \tau \cdot \beta \upharpoonright_{|\tau|-1}\}$ is r.e. Fix any $\sigma_0 \in \mathbb{Z}^{<\omega}$ such that $\sigma_0 \cdot \beta_{|\sigma_0|-1} = 1$ (since $1 \in G_R$, such a σ_0 must exist). Then there is a strictly increasing recursive enumeration $i_0, i_1, i_2, \dots$ such that for all j, i_j is the first ℓ found for which the following hold: (i) $\ell > i_{j'}$ whenever $j' < j$; (ii) there are $n_\ell \geq 1$ and relatively prime positive integers q, r with $p_\ell \nmid q$ and $p_\ell \nmid r$ such that for some m, $(q\sigma_0, I_m(rp_\ell^{n_\ell})) \in E$. Note that

$$\begin{aligned}(q\sigma_0, I_m(rp_\ell^{n_\ell})) \in E \Leftrightarrow q = (q\sigma_0) \cdot \beta_{|\sigma_0|-1} = I_m(rp_\ell^{n_\ell}) \cdot \beta_{m-1} = rp_\ell^{n_\ell} b_{m-1} \\ \Leftrightarrow b_{m-1} = qp_\ell^{-n_\ell} r^{-1}.\end{aligned}$$

The Martin-Löf randomness of R implies that β contains infinitely many terms of the form $\frac{q'}{r' p_{\ell'}^{n'_{\ell'}}}$ with $n'_{\ell'} \geq 1$, q' and r' relatively prime and positive, $p_{\ell'} \nmid q'$ and $p_{\ell'} \nmid r'$. Thus i_j is defined for all j, and by Proposition 7 this contradicts the Martin-Löf randomness of R.

Further, a variation of the algorithm yields that equality of the proper rational part is r.e. on random groups.

▶ **Theorem 15.** *If $R \leq_T K$ is Martin-Löf random, then equality modulo 1 on $(G_R, +)$ is r.e. with respect to some generating sequence.*

The next main result is concerned with the model-theoretic properties of random subgroups of rationals. We recall that two structures (in the model-theoretic sense) M and N with the same set σ of non-logical symbols are *elementarily equivalent* (denoted $M \equiv N$) iff they satisfy the same first-order sentences over σ; the *theory* of a structure M (denoted $\mathrm{Th}(M)$) is the set of all first-order sentences (over the set of non-logical symbols of M) that are satisfied by M. The reader is referred to [16] for more background on model theory. We will prove a result that may appear a bit surprising: even though Martin-Löf random subgroups of $(\mathbb{Q}, +)$ (viewed as classes of integer sequence representations) are not computable, any such subgroup is elementarily equivalent to $(\mathbb{Z}, +)$ - the additive group of integers - and thus has a decidable theory. In other words, the incomputability of a random subgroup of rationals, at least according to the notion of "randomness" adopted in the present work, has little or no bearing on the decidability of its first-order properties. We begin by showing that the theory of any subgroup G of rationals reduces to that of the subgroup of $(\mathbb{Q}, +)$ generated by the set of all rationals either equal to 1 or of the shape p^{-n}, where p is a prime infinitely dividing G and $n \in \mathbb{N}$. Our proof of this fact rests on a sufficient criterion due to Szmielew [27] for the elementary equivalence of two groups; this result will be stated as it appears in [11].

▶ **Theorem 16.** *([27], as cited in [11]) Let p be a prime number and G be a group. For all $n \geq 1$, $k \geq 1$ and elements $g_1, \ldots, g_k \in G$, define $G[p^n] := \{x \in G \mid p^n x = 0\}$ and the following predicate $C(p; g_1, \ldots, g_k)$:*

$C(p; g_1, \ldots, g_k) \Leftrightarrow$ the images $g'_1, \ldots, g'_k$ of $g_1, \ldots, g_k$ in the factor group $\overline{G} := G/G[p^n]$are such that $g'_1 + p\overline{G}, \ldots, g'_k + p\overline{G}$ are linearly independent in $\overline{G}/p\overline{G}$.

Define the parameters $\alpha_{p,n}(G), \beta_p(G)$ and $\gamma_p(G)$ as follows.

$$\begin{aligned} \alpha_{p,n}(G) &:= \sup\{k \in \mathbb{N} \mid G \text{ contains } \mathbb{Z}^k_{p^n} \text{ as a pure subgroup}\}, \\ \beta_p(G) &:= \inf\{\sup\{k \in \mathbb{N} \mid \mathbb{Z}^k_{p^n} \text{ is a subgroup of } G\} \mid n \in \mathbb{N}\}, \\ \gamma_p(G) &:= \inf\{\sup\{k \in \mathbb{N} \mid (\exists x_1, \ldots, x_k) C(p; x_1, \ldots, x_k)\} \mid n \in \mathbb{N}\}. \end{aligned}$$

(Here $pG := \{pg \mid g \in G\}$ and $\mathbb{Z}^k_{p^n}$ is the k-th power of the primary cyclic group on p^n elements, that is, it consists of all elements $(a_0, \ldots, a_{k-1})$ such that $a_0, \ldots, a_{k-1} \in \mathbb{Z}_{p^n}$.) Then any two groups H and L are elementarily equivalent iff $\alpha_{q,m}(H) = \alpha_{q,m}(L)$, $\beta_q(H) = \beta_q(L)$ and $\gamma_q(H) = \gamma_q(L)$ for all primes q and all $m \geq 1$.

The definition of a *pure* subgroup will not be used in the proof of the subsequent theorem; it will be observed that if G is a subgroup of the rationals, then for $k \geq 1$ and $n \geq 1$, it cannot contain $\mathbb{Z}^k_{p^n}$ as a subgroup in any case, so that $\alpha_{p,n}(G) = \beta_p(G) = 0$.

▶ **Theorem 17.** *Let G be a subgroup of $(\mathbb{Q}, +)$. Then $G \equiv [\mathbb{Z}]_{P(G)}$, where $P(G) := \{i \in \mathbb{N} \mid (\forall x \in G)(\forall n \in \mathbb{N})[\frac{x}{p_i^n} \in G]\}$ denotes the set of all primes infinitely dividing G and for a set of primes P we write $[\mathbb{Z}]_P$ for the subgroup of $(\mathbb{Q}, +)$ generated by $\{1\} \cup \{\frac{1}{p^k} \mid p \in P, k \in \mathbb{N}\}$.*

Note that $\mathrm{Th}([\mathbb{Z}]_K, +)$ is undecidable; in contrast, for R Martin-Löf random we have $P(G_R) = \varnothing$, so the promised corollary follows.

▶ **Corollary 18.** *Let $R \in 2^\omega$ be Martin-Löf random. Then $(G_R, +)$ and $(\mathbb{Z}, +)$ have the same theories.*

One may ask whether this still holds for richer structures. This is not the case, as for example the theory of $(G, +, <)$ is different from $\mathrm{Th}(\mathbb{Z}, +, <)$, as in the latter $x = 1$ is a satisfying assignment for the formula $x + x > x \wedge \forall y < x \neg y + y > y$. There does not exist an $x \in G_R$ with this property for a ML-random R.

4 Learning Finitely Generated Subgroups of a Random Subgroup of Rationals

In this section, we investigate the learnability of non-trivial finitely generated subgroups of any group G_R generated by a Martin-Löf random sequence R such that $R \leq_T K$. First, we introduce some additional notation.

▶ **Notation 19.** *Let $R \leq_T K$ be Martin-Löf random and let $\beta := (b_i)_{i<\omega}$ be any generating sequence of G_R. For any subgroup F of G_R, F_β denotes the set of all representations of elements of F with respect to β, that is, $F_\beta := \{\sigma \in \mathbb{Z}^{<\omega} \mid \sum_{i=0}^{|\sigma|-1} \sigma(i) b_i \in F\}$. Furthermore, define $\mathcal{F}_\beta := \{F_\beta \mid F$ is a non-trivial finitely generated subgroup of $G_R\}$.*

We will consider learning from *texts*, where a text is an infinite sequence that contains all elements of F_β for the F to be learnt and may contain the symbol #, which indicates a pause in the data presentation and thus no new information. For any text T and $n \in \mathbb{N}$, $T(n)$ denotes the $(n+1)$-st term of T and $T[n]$ denotes the finite sequence $T(0), \ldots, T(n-1)$, i.e., the *initial segment* of length n of T; $\mathrm{content}(T[n])$ denotes the set of non-pause elements occurring in $T[n]$. A *learner* M is a recursive function mapping $(\mathbb{Z}^{<\omega} \cup \{\#\})^*$ into $\mathbb{N} \cup \{?\}$; the ? symbol permits M to abstain from conjecturing at any stage. A learner is fed successively with growing initial segments of the text and it produces a sequence of conjectures $e_0, e_1, e_2, \ldots$, which are interpreted with respect to a fixed *hypothesis space*. In the present paper, we stick to the standard hypothesis space, a fixed Gödel numbering $W_0, W_1, W_2, \ldots$ of all r.e. subsets of $\mathbb{Z}^{<\omega}$. In our setting from the generator $\frac{q}{m}$ of F we can immediately derive an index e for F_β and therefore in the proofs we argue for learning q and m. The learner is said to *behaviourally correctly* (denoted **Bc**) learn the representation F_β of a finitely generated subgroup F with respect to a fixed generating sequence β for G_R iff on every text for F_β, the sequence of conjectures output by the learner converges to a correct hypothesis; in other words, the learner almost always outputs an r.e. index for F_β [7, 5, 2]. If almost all of the learner's hypotheses on the given text are equal in addition to being correct, then the learner is said to *explanatorily* (denoted **Ex**) learn F_β (or it learns F_β *in the limit*) [9].

A useful notion that captures the idea of the learner converging on a given text is that of a *locking sequence*, or more generally that of a *stabilising sequence*. A sequence $\sigma \in (\mathbb{N} \cup \{\#\})^*$ is called a *stabilising sequence* [8] for a learner M on some set L if $\mathrm{content}(\sigma) \subseteq L$ and for all $\tau \in (L \cup \{\#\})^*$, $M(\sigma) = M(\sigma \circ \tau)$. A sequence $\sigma \in (\mathbb{N} \cup \{\#\})^*$ is called a *locking sequence* [4] for a learner M on some set L if σ is a stabilising sequence for M on L and $W_{M(\sigma)} = L$.

The following proposition due to Blum and Blum [4] will be occasionally useful.

▶ **Proposition 20.** *[4] If a learner M explanatorily learns some set L, then there exists a locking sequence for M on L. Furthermore, all stabilising sequences for M on L are also locking sequences for M on L.*

Clearly, also a **Bc**-version of Proposition 20 holds.

It is not clear in the first place whether or not every finitely generated subgroup of a randomly generated subgroup of $(\mathbb{Q}, +)$ can even be represented as an r.e. set. This will be clarified in the next series of results. We recall that a *finitely generated subgroup* F of G_R is any subgroup of G_R that has some *finite generating set* S, which means that every element of F can be written as a linear combination of finitely many elements of S and the inverses of elements of S. F is *trivial* if it is equal to $\{0\}$; otherwise it *non-trivial.* Furthermore, if G_R is a subgroup of $(\mathbb{Q}, +)$, then any finitely generated subgroup F of G_R is *cyclic*, that is, $F = \left\langle \frac{q}{m} \right\rangle$ for some $q \in \mathbb{N}$ and $m \in \mathbb{N}$ with $\gcd(q, m) = 1$ (see, for example, [26, Theorem 8.1]). The latter fact will be used freely throughout this paper. For any generating sequence β for G_R and any finitely generated subgroup F of G_R, the set of representations of elements of F with respect to β will be denoted by F_β.

▶ **Theorem 21.** *Let $R \leq_T K$ be Martin-Löf random. Then there is a generating sequence $(b_i)_{i<\omega}$ of G_R such that for every non-trivial finitely generated subgroup F of G_R the set F_β is r.e.*

▶ Remark 22. The statement of Theorem 21 excludes the trivial subgroup because for any generating sequence $\beta := (b_i)_{i<\omega}$ for G_R, $\langle 0 \rangle_\beta$ cannot be r.e. To see this, suppose, by way of contradiction, that $\langle 0 \rangle_\beta$ were r.e. Given any $\sigma, \sigma' \in \mathbb{Z}^{<\omega}$, set $\ell = \max(\{|\sigma| - 1, |\sigma'| - 1\})$, and for all $i \in \{0, \ldots, \ell\}$, $w_i = \sigma(i)$ if $i \leq |\sigma| - 1$ and 0 otherwise, and $v_i = \sigma'(i)$ if $i \leq |\sigma'| - 1$ and 0 otherwise. Then $\sigma \cdot \beta \restriction_{|\sigma|-1} = \sigma' \cdot \beta \restriction_{|\sigma'|-1} \Leftrightarrow \sigma \cdot \beta \restriction_{|\sigma|-1} - \sigma' \cdot \beta \restriction_{|\sigma'|-1} = 0 \Leftrightarrow \sum_{i=0}^{\ell} (w_i - v_i) b_i = 0 \Leftrightarrow (w_0 - v_0, w_1 - v_1, \ldots, w_\ell - v_\ell) \in \langle 0 \rangle_\beta$. Thus if $\langle 0 \rangle_\beta$ were r.e., then equality with respect to β would also be r.e., which, as was shown earlier, is impossible.

We note that there cannot be any generating sequence β for G_R such that there are finitely generated subgroups F, F' of G_R with F_β r.e. and F'_β co-r.e.

▶ **Theorem 23.** *Let $R \leq_T K$ be Martin-Löf random. Let β be any generating sequence for G_R. Then for any finitely generated subgroups F and F' of G_R, one of the following holds: (i) both F_β and F'_β are r.e., (ii) both F_β and F'_β are co-r.e., or (iii) at least one of F_β and F'_β is neither r.e. nor co-r.e.*

▶ **Theorem 24.** *Let $R \leq_T K$ be Martin-Löf random. Then there is a generating sequence β of G_R such that F_β is r.e. for every non-trivial finitely generated subgroup F of G_R and $\mathcal{F}_\beta$ is* **Bc***-learnable.*

The next result shows, in contrast to Theorem 24, that if $R \leq_T K$ is Martin-Löf random, then, given *any* generating sequence β for G_R such that F_β is r.e. for every non-trivial finitely generated subgroup F of G_R, the class $\mathcal{F}_\beta$ is not explanatorily learnable.

▶ **Theorem 25.** *Let $R \leq_T K$ be Martin-Löf random. Suppose $\beta := (b_i)_{i<\omega}$ is a generating sequence for G_R such that for any non-trivial finitely generated subgroup F of G_R, F_β is r.e. Then $\mathcal{F}_\beta$ is not* **Ex***-learnable.*

The next theorem considers the learnability of the set of representations of any finitely generated subgroup F of the quotient group $G_R/\mathbb{Z}$ with respect to the generating sequence for $G_R/\mathbb{Z}$ constructed in the proof of Theorem 15. Slightly abusing the notation defined in Notation 19, for any generating sequence β for $G_R/\mathbb{Z}$, F_β will denote the set of representations of any subgroup F of $G_R/\mathbb{Z}$ with respect to β, and $\mathcal{F}_\beta$ will denote $\{F_\beta \mid F \text{ is a finitely generated subgroup of } G_R/\mathbb{Z}\}$.

▶ **Theorem 26.** *Suppose $R \leq_T K$ is Martin-Löf random. Let $G_R/\mathbb{Z}$ be the quotient group of G_R by $\mathbb{Z}$. Then there is a generating sequence β for $G_R/\mathbb{Z}$ such that F_β is r.e. for all finitely generated subgroups of $G_R/\mathbb{Z}$ and $\mathcal{F}_\beta$ is* **Bc***-learnable.*

As in the case of the collection of non-trivial finitely generated subgroups of G_R, the class $\mathcal{F}_\beta$ is not explanatorily learnable with respect to any generating sequence β for $G_R/\mathbb{Z}$. The proof is entirely analogous to that of Theorem 25.

▶ **Theorem 27.** *Let $R \leq_T K$ be Martin-Löf random. Suppose $\beta := (b_i)_{i<\omega}$ is a generating sequence for $G_R/\mathbb{Z}$ such that for any finitely generated subgroup F of $G_R/\mathbb{Z}$, F_β is r.e. Then $\mathcal{F}_\beta$ is not* $\mathbf{Ex}$*-learnable.*

A natural question is whether the learnability or non-learnability of a class of representations for a collection of subgroups of G_R is independent of the choice of the generating sequence for G_R. We have seen in Theorem 25, for example, that the non explanatory learnability of the class of non-trivial finitely generated subgroups of G_R holds for *any* generating sequence for G_R such that F_β is r.e. whenever F is a finitely generated subgroup. The next theorem gives a positive learnability result that is to some extent independent of the choice of the generating sequence: for any generating sequence β for G_R such that equality with respect to β is K-recursive and F_β is r.e. whenever F is a finitely generated subgroup of G_R, the class $\mathcal{F}_\beta$ is explanatorily learnable relative to oracle K.

▶ **Theorem 28.** *Let $R \leq_T K$ be Martin-Löf random. Then for any generating sequence β for G_R such that equality with respect to β is K-recursive (in other words, the set $E_\beta := \{(\sigma, \sigma') \in \mathbb{Z}^{<\omega} \times \mathbb{Z}^{<\omega} \mid \sigma \cdot \beta_{|\sigma|-1} = \sigma' \cdot \beta_{|\sigma'|-1}\}$ is K-recursive) and F_β is r.e. for all finitely generated subgroups of G_R, $\mathcal{F}_\beta$ is* $\mathbf{Ex}[K]$*-learnable.*

We recall from Theorem 15 that there is a generating sequence $\beta := (b_i)_{i<\omega}$ for G_R such that equality modulo 1 with respect to β is r.e.; in other words, the set $\{(\sigma, \sigma') \in \mathbb{Z}^{<\omega} \times \mathbb{Z}^{<\omega} \mid \sigma \cdot \beta_{|\sigma|-1} \equiv \sigma' \cdot \beta_{|\sigma|'-1}\ (mod\ 1)\}$ is r.e. The next result considers the learnability of a class that is in some sense “orthogonal” to the class $\mathbb{Z}_\beta$: the class of all sets of representations of $\mathbb{Z}$ with respect to *any* generating sequence β' for G_R such that $\mathbb{Z}_{\beta'}$ is r.e. In the statement and proof of the next theorem, for any generating sequence β for G_R, let E_β denote the set $\{(\sigma, \sigma') \in \mathbb{Z}^{<\omega} \times \mathbb{Z}^{<\omega} \mid \sigma \cdot \beta_{|\sigma|-1} = \sigma' \cdot \beta_{|\sigma'|-1}\ (mod\ 1)\}$.

▶ **Theorem 29.** *Let $R \leq_T K$ be Martin-Löf random. Let $\mathcal{G}_0$ be the collection of all generating sequences β for G_R such that E_β is r.e., and define $\mathcal{E}_0 := \{E_\beta \mid \beta \in \mathcal{G}_0\}$. Then $\mathcal{E}_0$ is not* $\mathbf{Bc}$*-learnable.*

In contrast to Theorem 29, we present a positive learnability result for the collection of all E_β such that E_β is co-r.e. The learnability is with respect to a hypothesis space which uses co-r.e. indices. That is to say, given any text T, the learner will on T always output an r.e. index for sets of the form $(\mathbb{Z}^{<\omega} \times \mathbb{Z}^{<\omega}) \setminus E_{\beta'}$, where $E_{\beta'}$ is some co-r.e. set. In the statement and proof of the next theorem, given any generating sequence β for G_R such that equality with respect to β is co-r.e., E_β will denote the set $\{(\sigma, \sigma') \in \mathbb{Z}^{<\omega} \times \mathbb{Z}^{<\omega} \mid \sigma \cdot \beta_{|\sigma|-1} = \sigma' \cdot \beta_{|\sigma'|-1}\}$.

▶ **Theorem 30.** *Let $R \leq_T K$ be Martin-Löf random. Let $\mathcal{G}_1$ be the collection of all generating sequences β for G_R such that E_β is co-r.e., and define $\mathcal{E}_1 := \{E_\beta \mid \beta \in \mathcal{G}_1\}$. Then $\mathcal{E}_1$ is explanatorily learnable relative to oracle K using co-r.e. indices. That is to say, there is a K-recursive learner M such that for any $E_\beta \in \mathcal{E}_1$ and any text T for E_β, M on T will output an r.e. index for $(\mathbb{Z}^{<\omega} \times \mathbb{Z}^{<\omega}) \setminus E_\beta$ in the limit.*

5 Conclusion and Possible Future Research

This paper introduced a method of constructing random subgroups of rationals, whereby Martin-Löf random binary sequences are directly encoded into the generators of the group.

It was shown that if the Martin-Löf random sequence associated to a randomly generated subgroup G is limit-recursive, then one can build a generating sequence β for G such that the word problem for G is co-r.e. with respect to β, as well as another generating sequence β' such that the word problem for $G/\mathbb{Z}$ with respect to β' is r.e. We also showed that every non-trivial finitely generated subgroup of G has an r.e. representation with respect to a suitably chosen generating sequence for G; moreover, the class of all such r.e. representations is behaviourally correctly learnable but never explanatorily learnable. We did not, however, extend the definition of algorithmic randomness to *all* Abelian groups; we suspect that such a general definition might be out of reach of current methods due to the fact that the isomorphism types of even rank 2 groups (subgroups of $(\mathbb{Q}^2, +)$) are still unknown.

References

1 Reinhold Baer. Abelian groups without elements of finite order. *Duke Mathematical Journal*, 3(1):68–122, March 1937.

2 Janis Bārzdiņš. Two theorems on the limiting synthesis of functions. *Latv. Gos. Univ. Uch. Zapiski*, 210:82–88, 1974. (In Russian).

3 Ross A. Beaumont and Herbert S. Zuckerman. A characterization of the subgroups of the additive rationals. *Pacific Journal of Mathematics*, 1(2):169–177, 1951.

4 Lenore Blum and Manuel Blum. Toward a Mathematical Theory of Inductive Inference. *Information and Control*, 28:125–155, 1975.

5 John Case and Carl Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.

6 Rod Downey and Denis Hirschfeldt. *Algorithmic Randomness and Complexity*. Springer-Verlag, Berlin, Heidelberg, 2010.

7 Jerome A. Feldman. Some decidability results on grammatical inference and complexity. *Information and Control*, 20(3):244–262, 1972.

8 Mark A. Fulk. *A Study of Inductive Inference Machines*. PhD thesis, State University of New York at Buffalo, Buffalo, NY, USA, 1986.

9 E. M. Gold. Language Identification in the Limit. *Information and Control*, 10:447–474, 1967.

10 Mikhail Gromov. Random walk in random groups. *Geometric and Functional Analysis*, 13:73–146, 2003.

11 Nazif G. Khisamiev. Chapter 17:Constructive abelian groups. In Yu. L. Ershov, S. S. Goncharov, A. Nerode, J. B. Remmel, and V. W. Marek, editors, *Handbook of Recursive Mathematics*, volume 139 of *Studies in Logic and the Foundations of Mathematics*, pages 1177–1231. Elsevier, 1998.

12 Bakhadyr Khoussainov. A Quest for Algorithmically Random Infinite Structures. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, pages 56:1–56:9, 2014.

13 Bakhadyr Khoussainov. A Quest for Algorithmically Random Infinite Structures, II. In *Logical Foundations of Computer Science - International Symposium, LFCS 2016*, pages 159–173, 2016.

14 Ming Li and Paul Vitány. A New Approach to Formal Language Theory by Kolmogorov Complexity. *SIAM Journal on Computing*, 24(2):398–410, 1995.

15 Ming Li and Paul Vitány. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, New York, NY, USA, 3rd edition, 2008.

16 David Marker. *Model theory: an introduction*, volume 217 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, NY, USA, 2002.

17 Eric Martin and Daniel N. Osherson. *Elements of scientific inquiry*. MIT Press, Cambridge, Massachusetts, 1998.

18 Per Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602–619, 1966.

19 André Nies. *Computability and Randomness*. Oxford University Press, Inc., New York, NY, USA, 2009.

20 André Nies and Volkher Scholz. Martin-Löf random quantum states. *arXiv preprint arXiv:1709.08422*, 2017.

21 Piergiorgio Odifredd. *Classical Recursion Theory*. North-Holland, Amsterdam, 1989.

22 Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. MIT Press, Cambridge, MA, USA, 1987.

23 Claus-Peter Schnorr. A Unified Approach to the Definition of a Random Sequence. *Mathematical Systems Theory*, 5(3):246–258, 1971.

24 Robert I. Soare. *Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets*. Perspectives in Mathematical Logic. Springer Berlin Heidelberg, 1999.

25 Frank Stephan and Yuri Ventsov. Learning algebraic structures from text. *Theoretical Computer Science*, 268(2):221–273, 2001.

26 Sándor Szabó and Arthur D. Sands. *Factoring groups into subsets*. Lecture Notes in Pure and Applied Mathematics. Chapman and Hall/CRC Press, 6000 Broken Sound Parkway NW, Suite 300, 2009.

27 Wanda Szmielew. Elementary properties of Abelian groups. *Fundamenta Mathematicae*, 41(2):203–271, 1955.

28 Todor Tsankov. The additive group of the rationals does not have an automatic presentation. *Journal of Symbolic Logic*, 76(4):1341–1351, 2011.

29 Volodya Vovk, Alexander Gammerman, and Craig Saunders. Machine-Learning Applications of Algorithmic Randomness. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999)*, pages 444–453, 1999.

Counting Induced Subgraphs: An Algebraic Approach to #W[1]-hardness[*]

Julian Dörfler
Saarbrücken Graduate School of Computer Science, Saarland Informatics Campus (SIC), Germany
s8judoer@stud.uni-saarland.de

Marc Roth
Cluster of Excellence (MMCI), Saarland Informatics Campus (SIC), Saarbrücken, Germany
https://www.roth-marc.com
mroth@mmci.uni-saarland.de

Johannes Schmitt
ETH Zürich, Switzerland
https://people.math.ethz.ch/~schmittj/
johannes.schmitt@math.ethz.ch

Philip Wellnitz
Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), Saarbrücken, Germany
https://people.mpi-inf.mpg.de/~wellnitz/
wellnitz@mpi-inf.mpg.de

Abstract

We study the problem #IndSub(Φ) of counting all induced subgraphs of size k in a graph G that satisfy the property Φ. This problem was introduced by Jerrum and Meeks and shown to be #W[1]-hard when parameterized by k for some families of properties Φ including, among others, connectivity [JCSS 15] and even- or oddness of the number of edges [Combinatorica 17]. Very recently [IPEC 18], two of the authors introduced a novel technique for the complexity analysis of #IndSub(Φ), inspired by the "topological approach to evasiveness" of Kahn, Saks and Sturtevant [FOCS 83] and the framework of graph motif parameters due to Curticapean, Dell and Marx [STOC 17], allowing them to prove hardness of a wide range of properties Φ. In this work, we refine this technique for graph properties that are non-trivial on edge-transitive graphs with a prime power number of edges. In particular, we fully classify the case of monotone bipartite graph properties: It is shown that, given *any* graph property Φ that is closed under the removal of vertices and edges, and that is non-trivial for bipartite graphs, the problem #IndSub(Φ) is #W[1]-hard and cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f, unless the Exponential Time Hypothesis fails. This holds true even if the input graph is restricted to be bipartite and counting is done modulo a fixed prime. A similar result is shown for properties that are closed under the removal of edges only.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Problems, reductions and completeness; Mathematics of computing → Combinatorics; Mathematics of computing → Graph theory

Keywords and phrases counting complexity, edge-transitive graphs, graph homomorphisms, induced subgraphs, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.26

Related Version A full version of the paper is available at https://arxiv.org/abs/1904.10479.

Funding *Johannes Schmitt*: The third author has received funding from the European Research Council (ERC) under the EU's Horizon 2020 research and innovation programme (No 786580).

Acknowledgements We are very grateful to Radu Curticapean and Holger Dell for fruitful discussions and valuable feedback on early drafts of this work.

[*] All authors of this paper are students.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 26; pp. 26:1–26:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The study of the computational complexity of counting problems was initiated by Valiant's seminal work about the complexity of computing the permanent [22]. In contrast to a decision problem which requires to *verify* the existence of a solution, a counting problem asks to compute the *number* of solutions. Counting complexity theory is particularly interesting for problems whose decision versions are solvable efficiently but whose counting versions are intractable. One such example is the problem of finding/counting perfect matchings, whose decision version is solvable in polynomial time [7] and whose counting version is as least as hard as every problem in the Polynomial Hierarchy PH with respect to polynomial-time Turing reductions [22, 21]. In this work, we consider the following problem which was first introduced by Jerrum and Meeks [11]: Fix a graph property Φ, given a graph G and a positive integer k, compute the number of all induced subgraphs of G with k vertices that satisfy Φ. We denote this problem by #IndSub(Φ) and remark that, strictly speaking, #IndSub(Φ) is the *unlabeled* version of p-#InducedSubgraphWithProperty(Φ) as defined in [12, Section 1.3.1]. In particular, our properties only depend on the isomorphism type of a graph and not on any labeling of the vertices.

We study the *parameterized complexity* of #IndSub(Φ) depending on the property Φ. The underlying framework, known as *parameterized counting complexity theory*, was introduced independently by Flum and Grohe [8] and McCartin [16], and constitutes a hybrid of (classical) computational counting and parameterized complexity theory. Here, the method of parameterization allows us to perform a multivariate analysis of the complexity of #IndSub(Φ): Instead of the distinction between polynomial-time solvable and NP-hard cases, we search for properties Φ for which the problem is solvable in time $f(k) \cdot n^{O(1)}$, where n is the number of vertices of the graph and f can be any computable function. If this is the case, the problem is called *fixed-parameter tractable.* Unfortunately, the only known cases of Φ for which #IndSub(Φ) is fixed-parameter tractable are trivial in the sense that there are only finitely many k such that Φ is neither true nor false on the set of all graphs with k vertices. On the contrary, it is easy to see that #IndSub(Φ) is most likely not fixed-parameter tractable if Φ encodes a problem whose decision version is already known to be hard. An example of the latter is the property of being a complete graph. In this case, the problem #IndSub(Φ) is identical to the problem of counting cliques of size k, for which even the decision version, that is, *finding* a clique of size k in a graph with n vertices, cannot be done in time $f(k) \cdot n^{o(k)}$, unless the Exponential Time Hypothesis fails [3, 4].

The first non-trivial hardness result of #IndSub(Φ) was given by Jerrum and Meeks for Φ the property of being connected [11]. Note that, in this case, the decision version of the problem can be solved efficiently as, on input G and k, one only has to decide whether there exists a connected component of G of size at least k. This result initiated a line of research in which Jerrum and Meeks proved fixed-parameter tractability of #IndSub(Φ) to be unlikely for the property of having an even (or odd) number of edges [12], for properties that induce low edge densities [10] and for properties that are closed under the addition of edges and whose (edge-)minimal elements have large treewidth [17]. More precisely, all of those results established hardness for the parameterized complexity class #W[1], which can be seen as the parameterized counting equivalent of NP. In a recent breakthrough result [5], Curticapean, Dell and Marx have shown, that for every graph property Φ, the problem #IndSub(Φ) is either fixed-parameter tractable or hard for #W[1], that is, there are no cases of intermediate difficulty. On the downside, they did not provide an explicit criterion for #W[1]-hardness that allows to pin down the complexity of #IndSub(Φ), given a concrete property Φ.

However, combining the framework of [5] with tools from the "topological approach to evasiveness" by Kahn, Saks and Sturtevant [13], two of the authors of the current paper established #W[1]-hardness for a wide range of properties, including, for example, all non-trivial properties that are closed under the removal of edges and false on odd cycles [20]. Taken together, the above results suggest the following conjecture.

▶ **Conjecture 1.** *Let Φ be a computable graph property satisfying that there are infinitely many positive integers k such that Φ is neither true nor false on all graphs with k vertices. Then $\#\textsc{IndSub}(\Phi)$ is $\#\mathsf{W}[1]$-hard.*

Unfortunately, a proof of this conjecture seems to be a long way off. In this work however, building up on [5, 20], we introduce an algebraic approach that allows us to resolve the above conjecture in case of *all* non-trivial monotone properties on bipartite graphs. In particular, we obtain a matching lower bound under the Exponential Time Hypothesis.

Results and techniques

We call a graph property *monotone* if it is closed under the removal of vertices and edges and *edge-monotone* if it is closed under the removal of edges only. Furthermore, we write IS_k for the graph consisting of k isolated vertices and $K_{t,t}$ for the complete bipartite graph with t vertices on each side. Our main theorems read as follows.

▶ **Theorem 2.** *Let Φ be a computable graph property and let $\mathcal{K}$ be the set of all prime powers t such that $\Phi(\mathsf{IS}_{2t}) \neq \Phi(K_{t,t})$. If $\mathcal{K}$ is infinite then $\#\textsc{IndSub}(\Phi)$ is $\#\mathsf{W}[1]$ hard. If additionally $\mathcal{K}$ is dense then it cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f unless ETH fails. This holds true even if the input graphs to $\#\textsc{IndSub}(\Phi)$ are restricted to be bipartite.*

In the previous theorem, a set $\mathcal{K}$ is *dense* if there exists a constant c such that for every $m \in \mathbb{N}$, there exists a $k \in \mathcal{K}$ such that $m \leq k \leq cm$. While the hypotheses of Theorem 2 sound technical, the theorem applies in many situations. In particular, it is applicable to properties that are neither (edge-) monotone nor the complement thereof: Let Φ be the property of being Eulerian. The graph $K_{t,t}$ contains an Eulerian cycle if $t = 2^s$ for $s \geq 1$. Hence we can apply Theorem 2 with $\mathcal{K} = \{2^s \mid s \geq 1\}$, which is infinite and dense.

▶ **Corollary 3.** *Let Φ be the property of being Eulerian. Then $\#\textsc{IndSub}(\Phi)$ is $\#\mathsf{W}[1]$-hard and cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f unless the ETH fails. This holds true even if the input graphs to $\#\textsc{IndSub}(\Phi)$ are restricted to be bipartite.*

In case Φ is edge-monotone, the condition $\Phi(\mathsf{IS}_{2t}) \neq \Phi(K_{t,t})$ is equivalent to non-triviality and if Φ is monotone, we obtain the following, more concise statement of the hardness result.

▶ **Theorem 4.** *Let Φ be a computable monotone graph property such that Φ and $\neg\Phi$ hold on infinitely many bipartite graphs. Then $\#\textsc{IndSub}(\Phi)$ is $\#\mathsf{W}[1]$-hard and cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f unless the Exponential Time Hypothesis fails. This holds true even if the input graphs to $\#\textsc{IndSub}(\Phi)$ are restricted to be bipartite.*

Let us illustrate further consequences of the previous theorems with respect to (edge-) monotone properties. First of all, most of the prior hardness results ([11, 10, 17, 12, 20]) are shown to hold in the restricted case of bipartite graphs. We provide three examples:

▶ **Corollary 5.** *The problem $\#\textsc{IndSub}(\Phi)$, restricted to* bipartite input graphs, *is $\#\mathsf{W}[1]$-hard and cannot be solved in time $f(k) \cdot |V(G)|^{o(k)}$ for any computable function f unless ETH fails, if Φ is one of the properties of being disconnected, planar or non-hamiltonian.*

One example of a monotone property Φ for which the complexity of $\#\textsc{IndSub}(\Phi)$ was unknown, even for general graphs, is given by the following corollary of Theorem 4.

▶ **Corollary 6.** *Let F be a fixed bipartite graph with at least one edge and define $\Phi(G) = 1$ if G does not contain a subgraph isomorphic to F. Then $\#\textsc{IndSub}(\Phi)$ is $\#\mathsf{W}[1]$-hard and cannot be solved in time $f(k) \cdot |V(G)|^{o(k)}$ for any computable function f unless ETH fails. This holds true even if the input graphs of $\#\textsc{IndSub}(\Phi)$ are restricted to be bipartite.*

As the number of induced subgraphs of size k that satisfy Φ equals $\binom{|V(G)|}{k}$ minus the number of induced subgraphs of size k that satisfy $\neg\Phi$, *all of the previous result remain true for the complementary properties* $\neg\Phi$.

In proving the previous theorems we build up on the approach in [5, 20], where it was shown that, given a graph property Φ and a positive integer k, the number of induced subgraphs of size k in a graph G that satisfy Φ can equivalently be expressed as the following sum over all (isomorphism types of) graphs H:

$$\sum_H a_\Phi(H) \cdot \#\mathsf{Hom}(H \to G)\,, \tag{1}$$

where a_Φ is a function from graphs to integers with finite support and $\#\mathsf{Hom}(H \to G)$ is the number of graph homomorphisms from H to G. It is known that computing a linear combination of homomorphism numbers, as in the above expression, is *precisely as hard as* computing its hardest term with a non-zero coefficient ([5], also implicitly proved in [2]). We refer to this property as *complexity monotonicity.* In [20] two of the authors of the current paper used a topological approach to analyze the coefficient $a_\Phi(K_k)$ of the complete graph on k vertices. If this coefficient is non-zero then complexity monotonicity implies that computing the number of induced subgraphs of size k in a graph G that satisfy Φ is at least as hard as computing the number $\#\mathsf{Hom}(K_k \to G)$. This, in turn, is equivalent to computing the number of cliques of size k in G, the canonical $\#\mathsf{W}[1]$-complete problem [8]. While this approach led to hardness proofs for a wide range of properties Φ, it seems that resolving Conjecture 1, even restricted to monotone properties, requires a significant amount of new ideas. Without going too much into the details[1] of [20], our analysis of $a_\Phi(K_k)$ is complicated by the fact that the number of edges of the complete graph on $k \geq 4$ vertices is not a prime power. In this work, we hence focus on the coefficient of $a_\Phi(H)$ for graphs H that have a prime power number of edges and for which computing $\#\mathsf{Hom}(H \to G)$ is hard. One example of such graphs is the biclique $K_{t,t}$ for some prime power t. Here a biclique $K_{t,t}$, also called a complete bipartite graph, has t vertices on each side and contains every edge from a vertex on the left side to a vertex to the right side. Hence the number of edges is t^2 which is a prime power if t is.

In analyzing the coefficient $a_\Phi(K_{t,t})$ of the complete bipartite graph, we invoke the results of Rivest and Vuillemin [19] who considered transitive boolean functions over a domain of prime power cardinality to resolve the asymptotic version of what is known as *Karp's evasiveness conjecture* (we recommend Miller's survey [18] for an excellent overview).

[1] Readers familiar with [20] might recall that fixed points of group actions have been used to derive a simpler formula to compute the number $a_\Phi(K_t)$ modulo a prime p for positive powers t of p. This formula would simplify greatly if the group had a p-power number of elements *and* acted transitively on the edges of K_t. Unfortunately, this can never happen for $t \geq 4$, since the number of edges of K_t is not itself a p-power.

Given a property Φ and a graph H, the *alternating enumerator* of Φ and H is defined to be

$$\hat{\chi}(\Phi, H) := \sum_{S \subseteq E(H)} \Phi(H[S]) \cdot (-1)^{\#S},$$

where $H[S]$ is the graph with vertices $V(H)$ and edges S. Roughly speaking, it will turn out that the value of $a_\Phi(H)$ is closely related to $\hat{\chi}(\Phi, H)$. We furthermore point out that, in case Φ is closed under the removal of edges, the alternating enumerator $\hat{\chi}(\Phi, H)$ equals what is called the reduced Euler characteristic of the simplicial complex on $E(H)$ associated to Φ [18, 20]. In Section 3 we study the alternating enumerator in case of edge-transitive graphs, that is, graphs whose automorphism groups act transitively on the set of edges. We give a self-contained proof of the following fact, which implicitly follows from [19].

▶ **Lemma 7.** *Let Φ be a graph property and let H be an edge-transitive graph with p^k edges such that p is a prime and $\Phi(H[\emptyset]) \neq \Phi(H)$. Then it holds that $\hat{\chi}(\Phi, H) = (\pm 1) \mod p$.*

Now, intuitively, Lemma 7 induces a strategy towards proving hardness of #IndSub(Φ): Assume a family of edge-transitive graphs $\mathcal{H}$ can be found such that $\#E(H)$ is a prime power and $\Phi(H[\emptyset]) \neq \Phi(H)$ for every $H \in \mathcal{H}$. Then #IndSub(Φ) is at least as hard as counting homomorphisms from graphs in $\mathcal{H}$, the latter of which is fully understood [6]. This observation gives a strong motivation for the study of edge-transitive graphs with a prime power number of edges. In the second part of Section 3, we fully classify those graphs as subgraphs of bipartite graphs or vertex-transitive subgraphs of wreath graphs; consult Section 3 for the formal definitions. The proof of the following theorem, which might be of independent interest, relies on a non-trivial application of Sylow's theorems.

▶ **Theorem 8.** *Let G be a connected edge-transitive graph with p^t edges for some prime p and positive integer t. Then either G is bipartite or G is vertex-transitive and can be obtained from the wreath graph W_{p^k} for $k \geq 1$ by removing edges (or both).*

With the analysis of $\hat{\chi}$ and edge-transitive graphs completed, we turn to the reduction from counting homomorphisms in Section 4. More precisely, given a class $\mathcal{H}$ of edge-transitive graphs with a prime power number of edges and a graph property Φ such that for every $H \in \mathcal{H}$ we have that $\Phi(H[\emptyset]) \neq \Phi(H)$, we construct a parameterized Turing reduction from #Hom($\mathcal{H}$) to #IndSub(Φ). Here, the problem #Hom($\mathcal{H}$) is defined as follows: Given as input a graph $H \in \mathcal{H}$ and a graph G, compute the number of homomorphisms from H to G. For technical reasons, we cannot immediately transform the number of induced subgraphs that satisfy Φ to a linear combination of homomorphism numbers as in Equation (1). We solve this technical issue by introducing color-prescribed variants of those problems in an intermediate step. In this context we consider H-colored graphs. Recall that a graph G is H-colored if it comes with a homomorphism c from G to H. A homomorphism from H to G is then called color-prescribed if it maps every vertex v of H to a vertex u of G satisfying that $c(u) = v$. We demonstrate that, given an H-colored graph G and oracle access to #IndSub(Φ), the following linear combination can be computed in time $f(|V(H)|) \cdot |V(G)|^{O(1)}$.

$$\sum_{S \subseteq E(H)} \hat{a}_\Phi(S) \cdot \#\mathsf{cp\text{-}Hom}(H[S] \to G). \tag{2}$$

Here $\mathsf{cp\text{-}Hom}(H[S] \to G)$ denotes the set of color-prescribed homomorphisms from $H[S]$ to G and $\hat{a}_\Phi$ is a function of finite support only depending in Φ. In particular, $\hat{a}_\Phi(E(H))$ and $\hat{\chi}(\Phi, H)$ are proved to agree up to a factor of -1. Finally, we establish complexity monotonicity for linear combinations of color-prescribed homomorphisms as in Equation (2), which in combination with Lemma 7 yields the desired reduction.

Combining the previous results, we invoke the reduction on graph properties that are non-trivial on bipartite graphs and prove Theorem 2 and Theorem 4, in Section 5. Furthermore, we illustrate that our algebraic approach readily extends to modular counting by proving that both, Theorem 2 and Theorem 4 remain true in case counting is done modulo a fixed prime. Due to space constraints, the formal statement and proof of the modular counting version, as well as some proofs of Sections 3 and 4, are deferred to the full version.

2 Preliminaries

Given a positive integer k, we write $[k]$ for the set $\{1,\dots,k\}$ and given a set A we write $\binom{A}{k}$ for the set of all subsets of size k of A. Furthermore, assuming that A is finite, we write $\#A$ or $|A|$ for its cardinality. Given a function $g : A \times B \to C$ and an element $a \in A$, we write $g(a,\star)$ for the function which maps $b \in B$ to $g(a,b)$. Some of our proofs rely on (elementary) group theory; due to the space constraints we refrain from an introduction and refer the reader to e.g. Chapter 1 in the standard textbook of Lang [14].

2.1 Graph theory

Graphs in this work are considered simple, undirected and without self-loops. More precisely, a graph G is a pair of a finite set $V(G)$ of vertices and a symmetric and irreflexive relation $E(G) \subseteq V(G)^2$. If a graph H is obtained from G by deleting a set of edges and a set of vertices of G, including incident edges, then H is called a *subgraph* of G. Given a subset $\hat{V}$ of $V(G)$ we write $G[\hat{V}]$ for the graph with vertices $\hat{V}$ and edges $E \cap \hat{V}^2$. The resulting graph is called an *induced subgraph* of G. An *edge-subgraph* of a graph H is a graph obtained from H by deleting edges. Given a set $S \subseteq E(H)$ we write $H[S]$ for the edge-subgraph $(V(H), S)$ of H.

Homomorphisms and embeddings

A *homomorphism* from a graph H to a graph G is a mapping $h : V(H) \to V(G)$ that preserves adjacencies. In other words, for every edge $\{u,v\} \in E(H)$ it holds that $\{h(u),h(v)\} \in E(G)$. We write $\mathsf{Hom}(H \to G)$ for the set of all homomorphisms from H to G. A homomorphism inducing a bijection of vertices and satisfying $\{u,v\} \in E(H)$ if and only if $\{f(u),f(v)\} \in E(G)$ is called an *isomorphism* and we say that two graphs H and $\hat{H}$ are *isomorphic* if there exists an isomorphism from H to $\hat{H}$. We write $\mathsf{Sub}(H \to G)$ and $\mathsf{IndSub}(H \to G)$ for the sets of all subgraphs and induced subgraphs of G, respectively, that are isomorphic to H.

An isomorphism from a graph to itself is called an *automorphism.* The set of automorphisms of a graph, together with the operation of functional composition constitutes a group, called the *automorphism group* of a graph. Slightly abusing notation, we will write $\mathsf{Aut}(H)$ for both the set of automorphisms of a graph H as well as for the automorphism group of H.

An *embedding* is an injective homomorphism and we write $\mathsf{Emb}(H \to G)$ for the set of embeddings from H to G. If an embedding h from H to G additionally satisfies that $\{h(u),h(v)\} \in E(G)$ implies $\{u,v\} \in E(H)$, we call it a *strong embedding.* We write $\mathsf{StrEmb}(H \to G)$ for the set of strong embeddings from H to G. Observe that the images of embeddings and strong embeddings from H to G are precisely the subgraphs and induced subgraphs of G that are isomorphic to H.

Colored variants

Given graphs G and H, we say that G is *H-colored* if G comes with a homomorphism c from G to H, called an *H-coloring*. Note that, in particular, every edge-subgraph of H can be H-colored by the identity function on $V(H)$, which is assumed to be the given coloring whenever we consider H-colored edge-subgraphs of H in this paper. Given an edge-subgraph F of H and a homomorphism h from F to a H-colored graph G, we say that h is *color-prescribed* if for all $v \in V(F) = V(H)$ it holds that $c(h(v)) = v$. We write $\mathsf{cp\text{-}Hom}(F \to G)$ for the set of all color-prescribed homomorphisms from F to G. $\mathsf{cp\text{-}StrEmb}(F \to G)$ is defined similarly for color-prescribed strong embeddings. We point out that a definition of $\mathsf{cp\text{-}Emb}$ is obsolete as every color-prescribed homomorphism is injective by definition and hence an embedding. Furthermore, we write $\mathsf{cp\text{-}Sub}(F \to G)$ and $\mathsf{cp\text{-}IndSub}(F \to G)$ for the sets of images of color-prescribed embeddings and strong embeddings from F to G, respectively. Elements of $\mathsf{cp\text{-}Sub}(F \to G)$ and $\mathsf{cp\text{-}IndSub}(F \to G)$ are referred to as color-prescribed subgraphs and induced subgraphs.[2]

Graph properties and the alternating enumerator

A *graph property* is a function Φ from graphs to $\{0, 1\}$ such that for any pair of isomorphic graphs H and $\hat{H}$ we have that $\Phi(H) = \Phi(\hat{H})$. Adapting the notation of Rivest and Vuillemin [19], we define the *alternating enumerator* of a property Φ and a graph H to be the function

$$\hat{\chi}(\Phi, H) := \sum_{S \subseteq E(H)} \Phi(H[S]) \cdot (-1)^{\# S} .$$

A graph property Φ is called *edge-monotone* if it is closed under the removal of edges. It is called *monotone* if it is closed under the removal of edges *and* vertices.[3] Given a graph property Φ, a positive integer k and a graph G, we write $\mathsf{IndSub}(\Phi, k \to G)$ for the set of all induced subgraphs of size k of G that satisfy Φ. Furthermore, given a graph property Φ and an H-colored graph G, we write $\mathsf{cp\text{-}IndSub}(\Phi \to G)$ for the set of all color-prescribed induced subgraphs of size $|V(H)|$ in G that satisfy Φ.

2.2 Parameterized counting complexity

The field of parameterized counting was introduced independently by McCartin [16] and Flum and Grohe [8] and constitutes a hybrid of classical computational counting and parameterized complexity theory. A *parameterized counting problem* is a pair of a function $P : \Sigma^* \to \mathbb{N}$ and a computable parameterization $\kappa : \Sigma^* \to \mathbb{N}$. It is called *fixed-parameter tractable* (FPT) if there exists a computable function f and a deterministic algorithm that computes $P(x)$ in time $f(\kappa(x)) \cdot |x|^{O(1)}$ for every $x \in \Sigma^*$. A *parameterized Turing reduction* from (P, κ) to $(\hat{P}, \hat{\kappa})$ is a deterministic FPT algorithm with respect to κ that is given oracle access to $\hat{P}$ and that on input x computes $P(x)$ with the additional restriction that there exists a computable function g such that for any oracle query y it holds that $\hat{\kappa}(y) \leq g(\kappa(x))$. We write $(P, \kappa) \leq^{\mathrm{fpt}}_{\mathrm{T}} (\hat{P}, \hat{\kappa})$ if a parameterized Turing reduction exists.

[2] The observant reader might have noticed that the sets $\mathsf{cp\text{-}Sub}(F \to G)$ and $\mathsf{cp\text{-}Hom}(F \to G)$ as well as $\mathsf{cp\text{-}IndSub}(F \to G)$ and $\mathsf{cp\text{-}StrEmb}(F \to G)$ are essentially the same as a color-prescribed homomorphism is uniquely identified by its image. However, we decided to distinguish those notions in order to make the combinatorial arguments in Section 4 more accessible.

[3] To avoid confusion, we remark that in some literature, e.g. in [17] a property is called monotone if it is closed under *addition* of vertices and edges.

Given a graph G and a positive integer k, the parameterized counting problem #Clique asks to compute the number of complete subgraphs of size k in G and is parameterized by k, that is $\kappa(G,k) := k$. It is complete for the class #W[1], which can be seen as a parameterized counting equivalent of NP [8]. Evidence for the fixed-parameter intractability of #W[1]-hard problems is given by the *Exponential Time Hypothesis* (ETH), which asserts that 3-SAT cannot be solved[4] in time $\mathsf{exp}(o(m))$ where m is the number of clauses of the input formula. Assuming ETH, #Clique cannot be solved in time $f(k) \cdot n^{o(k)}$ for any function f [3, 4] and hence #W[1]-hard problems are not fixed-parameter tractable.

Given a recursively enumerable class of graphs $\mathcal{H}$, the problem #Hom($\mathcal{H}$) asks, given a graph $H \in \mathcal{H}$ and an arbitrary graph G, to compute $\#\mathsf{Hom}(H \to G)$. Its parameterization is given by $\kappa(H,G) := |V(H)|$. The problems #cp-Hom($\mathcal{H}$) and #cp-IndSub($\mathcal{H}$) are defined similarly. Note that the inputs of the latter two problems are of the form (H,G) where $H \in \mathcal{H}$ and G comes with an explicitly given H-coloring.

Given a computable graph property Φ, the problem #IndSub(Φ) asks, given a graph G and a positive integer k, to compute $\#\mathsf{IndSub}(\Phi, k \to G)$ and the parameterization is given by $\kappa(G,k) := k$. Furthermore, we define #cp-IndSub(Φ) to be the problem of, given a graph G that is H-colored for some graph H, computing $\#\mathsf{cp}\text{-}\mathsf{IndSub}(\Phi \to G)$ and parameterize it by $\kappa(G) := |V(H)|$. We emphasize that, similarly to #cp-Hom($\mathcal{H}$), the input graph G comes with an explicitly given H-coloring, from which H can be constructed and thus the parameterization is well-defined.

3 Alternating enumerators and p-edge-transitive graphs

In this part of the paper we will provide a rough exposition of the work of Rivest and Vuillemin [19] who studied transitive boolean functions to resolve the asymptotic version of Karp's evasiveness conjecture. We will then apply their result to graphs H that are both edge-transitive and have p^ℓ many edges for some prime p. This will enable us to conclude that the alternating enumerator of Φ and H is (± 1) modulo p whenever $\Phi(H[\emptyset]) \neq \Phi(H)$. We start by introducing some required notions from algebraic graph theory.
The automorphism group of a graph H induces a group action on the edges of H, given by $h\{u,v\} := \{h(u), h(v)\}$. A group action is *transitive* if there exists only one orbit and a graph H is called *edge-transitive* if the group action on the edges is transitive, that is, if for every pair of edges $\{u,v\}$ and $\{\hat{u},\hat{v}\}$ there exists an automorphism $h \in \mathsf{Aut}(H)$ such that $h\{u,v\} = \{\hat{u},\hat{v}\}$. If additionally the number of edges of an edge-transitive graph is a prime power p^ℓ we call the graph *p-edge-transitive*.

▶ **Lemma 9** (Lemma 7 restated). *Let Φ be a graph property and let H be a p-edge-transitive graph such that $\Phi(H[\emptyset]) \neq \Phi(H)$. Then it holds that $\hat{\chi}(\Phi, H) = (\pm 1) \mod p$.*

Lemma 9 is implicitly proven in [19, Theorem 4.3], but for completeness we will include a short and self-contained proof, demonstrating a first application of the machinery of Sylow subgroups that we will need later.

For the proofs in this section, let us recall some key results from group theory. Given a prime number p, a finite group Γ' is called a *p-group* if the order $\#\Gamma'$ is a power of p. The following is a well-known and central result from the theory of finite groups.

[4] We point out that this includes deterministic and randomized algorithms.

▶ **Theorem 10** (Sylow theorems). *Let Γ be a finite group of order $\#\Gamma = p^k m$ for a prime p and an integer $m \geq 1$ coprime to p. Then Γ contains a subgroup Γ' of order p^k. Moreover, every other subgroup Γ'' of Γ of order p^k is conjugate to Γ', that is there exists $g \in \Gamma$ with $\Gamma'' = g\Gamma' g^{-1}$. In particular, the groups Γ', Γ'' are isomorphic (via the conjugation by g).*

Finally, every subgroup $\tilde{\Gamma} \subseteq \Gamma$ which is a p-group is actually contained in some conjugate $g\Gamma' g^{-1}$ of the group Γ'.

A subgroup $\Gamma' \subseteq \Gamma$ as above is called a *p-Sylow subgroup* of Γ.

The following result is a first important application of the Sylow theorems. It can be found as Exercise (E28) in [1]; for completeness we include a proof in the full version.

▶ **Lemma 11.** *Let Γ be a finite group acting transitively on a set T such that $\#T = p^l$ for some $l \geq 0$. Then the induced action of any p-Sylow subgroup $\Gamma' \subseteq \Gamma$ on T is still transitive.*

This result allows us to give a short proof of Lemma 9 above. We sketch the proof here and provide the details in the full version of the paper.

Proof sketch of Lemma 9. Let Γ' be a p-Sylow subgroup of $\mathsf{Aut}(H)$, then by Lemma 11 it acts transitively on $E(H)$. This action on the edges of H induces an action on the set of subsets $S \subseteq E(H)$ and by the Orbit-Stabilizer Theorem, for any S which is not invariant under Γ', the size of its orbit by Γ' is a positive power of p. Then in the sum

$$\hat{\chi}(\Phi, H) = \sum_{S \subseteq E(H)} \Phi(H[S]) \cdot (-1)^{\#S},$$

we group together summands belonging to S in the same Γ'-orbit. The contribution of any orbit of positive size is divisible by p and can be left out modulo p. Since Γ' acts transitively on $E(H)$, the only invariant sets S are $S = \emptyset$ and $S = E(H)$, so we have

$$\hat{\chi}(\Phi, H) = \Phi(H[\emptyset]) + \Phi(H[E(H)]) \cdot (-1)^{\#E(H)} = \Phi(H[\emptyset]) - \Phi(H) \mod p.$$

Note that we use the fact that for $p > 2$ we have that $\#E(H)$ is odd since it is a prime power and for $p = 2$ we have $-1 = 1$ modulo p. Now, the condition $\Phi(H[\emptyset]) \neq \Phi(H)$ exactly gives us $\Phi(H[\emptyset]) - \Phi(H) = \pm 1 \mod p$. ◀

There are two main examples for p-edge-transitive graphs. The first example is the class of the complete, bipartite graphs K_{p^l,p^m} with $l, m \geq 0$. The graph K_{p^l,p^m} has p^{l+m} edges and the automorphism group clearly acts transitively on the edges of that graph. The second example is the class of wreath graphs W_{p^k} for $k \geq 1$. The graph W_{p^k} has p^k vertices that can be decomposed in disjoint sets $V_0, \ldots, V_{p-1}$ of order p^{k-1} each, and edges $\{v_i, v_{i+1}\}$ for each $i = 0, \ldots, p-1$ and vertices $v_i \in V_i, v_{i+1} \in V_{i+1}$ (where it is understood that $V_p = V_0$). Thus in total, W_{p^k} has p^{2k-1} edges, except for $p = 2$ where it has 2^{2k-2} edges. The graph W_{p^k} can be seen as the lexicographical product of a p-cycle with a graph consisting of p^{k-1} disjoint vertices. For $k = 1$ we exactly obtain the p-cycle. To see that W_{p^k} is edge-transitive, we observe that on the one hand, for fixed i we can apply an arbitrary permutation on V_i leaving the graph invariant. On the other hand, there exists a “rotational action” sending V_j to V_{j+1} for $j = 0, \ldots, p-1$, which also leaves the graph invariant. Using these two types of automorphisms, we can map every edge to every other edge.

The following result tells us that in a certain sense the graphs K_{p^l,p^m} and W_{p^k} are the maximal p-edge-transitive graphs. A graph G is called *vertex-transitive* if its automorphism group $\mathsf{Aut}(G)$ acts transitively on its set of vertices $V(G)$.

▶ **Theorem 12** (Theorem 8 restated). *Let G be a connected p-edge-transitive graph. Then either G is bipartite (and thus a subgraph of a graph of the form K_{p^l,p^m} for some $l,m \geq 0$) or G is vertex-transitive and an edge-subgraph of W_{p^k} for $k \geq 1$ (or both).*

Due to the space constraints the proof is deferred to the full version of the paper.

4 The main reduction: From homomorphisms to induced subgraphs

In what follows we will construct a sequence of reductions, starting from $\#\textsc{Hom}(\mathcal{H})$ and ending in $\#\textsc{IndSub}(\Phi)$. Here, $\mathcal{H}$ is a recursively enumerable set of p-edge-transitive graphs and Φ is a graph property such that for every graph $H \in \mathcal{H}$ we have that $\Phi(H[\emptyset]) \neq \Phi(H)$. More precisely, we will prove that

$$\#\textsc{Hom}(\mathcal{H}) \leq^{\mathrm{fpt}}_{\mathrm{T}} \#\textsc{cp-Hom}(\mathcal{H}) \overset{\text{Lemma 17}}{\leq^{\mathrm{fpt}}_{\mathrm{T}}} \#\textsc{cp-IndSub}(\Phi) \leq^{\mathrm{fpt}}_{\mathrm{T}} \#\textsc{IndSub}(\Phi) \qquad (3)$$

In particular, all of those reductions will be tight in the sense that conditional lower bounds on the fine-grained complexity of $\#\textsc{Hom}(\mathcal{H})$ immediately transfer to $\#\textsc{IndSub}(\Phi)$. For the hardness results we rely on a result of Dalmau and Jonsson [6] stating that the problem $\#\textsc{Hom}(\mathcal{H})$ is known to be #W[1]-hard whenever $\mathcal{H}$ is recursively enumerable and of unbounded treewidth.[5] Here a class of graphs is said to have unbounded treewidth if for every $b \in \mathbb{N}$ there exists a graph in the class with treewidth at least b. Due to space constraints, the remainder of this section is concerned with proving Lemma 17, that is, the second step of the reduction sequence; the first and the third step are deferred to the full version.

Reducing color-prescribed homomorphisms to color-prescribed induced subgraphs

The reduction from color-prescribed homomorphisms to color-prescribed induced subgraphs requires the introduction of an H-colored variant of the framework of graph motif parameters, which was explicitly introduced in [5] and implicitly used in [2]. More precisely, given an H-colored graph G and a property Φ, we will express $\#\mathsf{cp\text{-}IndSub}(\Phi \to G)$ as a linear combination of color-prescribed homomorphisms, that is, terms of the form $\#\mathsf{cp\text{-}Hom}(H[S] \to G)$. In a first step, we show complexity monotonicity for linear combinations of color-prescribed homomorphisms. While this property allows a quite simple proof, a second step, in which we study the coefficient of $\#\mathsf{cp\text{-}Hom}(H \to G)$ requires a thorough understanding of the alternating enumerator of Φ and H. In case of p-edge-transitive graphs, the latter is provided by Lemma 9.

We start by introducing a colored variant of the tensor product of graphs (see e.g. Chapter 5.4.2 in [15]). Given two H-colored graphs G and $\hat{G}$ with colorings c and $\hat{c}$ we define their *color-prescribed* tensor product $G \times_H \hat{G}$ as the graph with vertices $V = \{(v,\hat{v}) \in V(G) \times V(\hat{G}) \mid c(v) = \hat{c}(\hat{v})\}$ and edges between two vertices $(v,\hat{v})$ and $(u,\hat{u})$ if and only if $\{v,u\} \in E(G)$ and $\{\hat{v},\hat{u}\} \in E(\hat{G})$. The next lemma states that $\#\mathsf{cp\text{-}Hom}$ is linear with respect to $\times_H$, a short proof of which can be found in the full version of the paper.

▶ **Lemma 13.** *Let H be a graph, let F be an edge-subgraph of H and let G and $\hat{G}$ be H-colored. Then we have that*

$$\#\mathsf{cp\text{-}Hom}(F \to G \times_H \hat{G}) = \#\mathsf{cp\text{-}Hom}(F \to G) \cdot \#\mathsf{cp\text{-}Hom}(F \to \hat{G})\,.$$

[5] We remark that the graph parameter of treewidth is not used explicitly in this work. Hence we omit the definition and refer the interested reader e.g. to Chapter 11 in [9].

We are now prepared to prove the color-prescribed variant of complexity monotonicity.

▶ **Lemma 14** (Complexity monotonicity). *Let H be a graph and let a be a function from edge-subgraphs of H to rationals. There exists an algorithm $\mathbb{A}$ that is given an H-colored graph G as input and has oracle access to the function*

$$\sum_{S\subseteq E(H)} a(H[S])\cdot \#\textsf{cp-Hom}(H[S]\to \star)\,,$$

and computes $\#\textsf{cp-Hom}(H[S]\to G)$ for all S such that $a(H[S])\neq 0$ in time $f(|H|)\cdot|V(G)|$ where f is a computable function. Furthermore, every oracle query $\hat{G}$ satisfies $|V(\hat{G})|\leq f(|H|)\cdot|V(G)|$.

Proof. Using Lemma 13 we have that for every H-colored graph F it holds that

$$\sum_{S\subseteq E(H)} a(H[S])\cdot \#\textsf{cp-Hom}(H[S]\to (G\times_H F)) \tag{4}$$

$$= \sum_{S\subseteq E(H)} a(H[S])\cdot \#\textsf{cp-Hom}(H[S]\to G)\cdot \#\textsf{cp-Hom}(H[S]\to F)\,, \tag{5}$$

which we can evaluate for $F = H[\emptyset],\dots,H[E(H)]$. This induces a system of linear equations which can easily be shown to have a unique solution; the proof that the corresponding matrix is non-singular can be found in the full version of the paper. Consequently, the numbers $a(H[S])\cdot\#\textsf{cp-Hom}(H[S]\to G)$ are uniquely determined and can be computed by solving the system using Gaussian elimination. Finally, we obtain the numbers $\#\textsf{cp-Hom}(H[S]\to G)$ by multiplying with $a(H[S])^{-1}$ whenever $a(H[S])\neq 0$. ◀

It remains to express the number of color-prescribed induced subgraphs that satisfy a property Φ as a linear combination of color-prescribed homomorphisms. We only sketch the proof of the following lemma and defer the details to the full version of the paper.

▶ **Lemma 15.** *Let H be a graph, let Φ be a graph property and let G be an H-colored graph. Then it holds that*

$$\#\textsf{cp-IndSub}(\Phi\to G) = \sum_{S\subseteq E(H)} \Phi(H[S]) \sum_{J\subseteq E(H)\setminus S} (-1)^{\#J}\cdot \#\textsf{cp-Hom}([H[S\cup J]\to G)\,.$$

Moreover, the absolute values of the coefficient of $\#\textsf{cp-Hom}(H\to G)$ and $\hat{\chi}(\Phi,H)$ are equal.

Proof sketch. We rely on the following claim which follows by inclusion-exclusion.

▷ Claim 16. Let H be graph, let $S\subseteq E(H)$ and let G be an H-colored graph. Then we have that

$$\#\textsf{cp-IndSub}(H[S]\to G) = \sum_{J\subseteq E(H)\setminus S} (-1)^{\#J}\cdot \#\textsf{cp-Sub}(H[S\cup J]\to G)\,.$$

Now summing up over all S for which $\Phi(H[S])=1$ and applying Claim 16 yields

$$\#\textsf{cp-IndSub}(\Phi\to G) = \sum_{S\subseteq E(H)} \Phi(H[S]) \sum_{J\subseteq E(H)\setminus S} (-1)^{\#J}\cdot \#\textsf{cp-Hom}(H[S\cup J]\to G)\,. \tag{6}$$

Finally, we collect for the coefficient of $\#\textsf{cp-Hom}(H\to G)$ and obtain

$$\sum_{S\subseteq E(H)} \Phi(H[S])\cdot(-1)^{\#E(H)-\#S} = (-1)^{\#E(H)}\cdot\hat{\chi}(\Phi,H)\,. \tag{7}$$

◀

The application of the complexity monotonicity property for color-prescribed homomorphisms (Lemma 14) requires non-zero coefficients. However, this can be guaranteed for the coefficient of interest in case of p-edge-transitive graphs as shown in Section 3. Formally, the reduction is constructed as follows.

▶ **Lemma 17.** *Let Φ be a graph property and let H be a p-edge-transitive graph such that $\Phi(H[\emptyset]) \neq \Phi(H)$. There exists an algorithm $\mathbb{A}$ that is given an H-colored graph G as input and has oracle access to the function $\#\textsf{cp-IndSub}(\Phi \to \star)$ and computes $\#\textsf{cp-Hom}(H \to G)$ in time $f(|H|) \cdot |V(G)|$ where f is a computable function. Furthermore, every oracle query $\hat{G}$ is H-colored as well and satisfies $|V(\hat{G})| \leq f(|H|) \cdot |V(G)|$.*

Proof. Using Lemma 15 we can express $\#\textsf{cp-IndSub}(\Phi \to \star)$ as a linear combination of color-prescribed homomorphisms. In particular, the coefficient of $\#\textsf{cp-Hom}(H \to \star)$ is $(\pm 1) \cdot \hat{\chi}(\Phi, H)$ and by Lemma 9 we have that this number is non-zero whenever H is p-edge-transitive and $\Phi(H[\emptyset]) \neq \Phi(H)$. Hence we can use the algorithm from Lemma 14 to compute $\#\textsf{cp-Hom}(H \to G)$ in the desired running time. ◀

5 Non-trivial monotone properties on bipartite graphs

In the last part of the paper, we apply the algebraic approach which was laid out in the preceding sections to bipartite graph properties. This will allow us to prove our main result. To this end, we say that a set $\mathcal{K} \subseteq \mathbb{N}$ is *dense* if there exists a constant c such that for every $k' \in \mathbb{N}$ there exists $k \in \mathcal{K}$ such that $k' \leq k \leq ck'$. Furthermore, we write $\textsf{IS}_k$ for the graph with k isolated vertices. The following theorem is obtained by invoking the reduction sequence (3) to complete bipartite graphs $K_{t,t}$ for prime powers $t = p^k$, which are p-edge-transitive (see Section 3). Due to the space constraints, the details, as well as the case of modular counting, are deferred to the full version of the paper.

▶ **Theorem 18** (Theorem 2 restated). *Let Φ be a computable graph property and let $\mathcal{K}$ be the set of all prime powers t such that $\Phi(\textsf{IS}_{2t}) \neq \Phi(K_{t,t})$. If $\mathcal{K}$ is infinite then $\#\textsc{IndSub}(\Phi)$ is $\#\mathsf{W}[1]$ hard. If additionally $\mathcal{K}$ is dense then it cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f unless ETH fails. This holds true even if the input graphs to $\#\textsc{IndSub}(\Phi)$ are restricted to be bipartite.*

Note that, in case Φ or its complement is edge-monotone, we only have to find infinitely many prime powers t for which Φ is neither true nor false on the set of all edge-subgraphs of $K_{t,t}$, which is the case for all sensible, non-trivial properties that do not rely on the number of vertices in some way. If Φ (or its complement) is monotone, that is, not only closed under the removal of edges, but also under the removal of vertices, then such artificial properties do not exist and we can state the result more clearly as follows.

▶ **Corollary 19** (Theorem 4 restated). *Let Φ be a computable monotone graph property such that Φ and $\neg\Phi$ hold on infinitely many bipartite graphs. Then $\#\textsc{IndSub}(\Phi)$ is $\#\mathsf{W}[1]$-hard and cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f unless ETH fails. This holds true even if the input graphs to $\#\textsc{IndSub}(\Phi)$ are restricted to be bipartite.*

Proof. If Φ is monotone and Φ and $\neg\Phi$ hold on infinitely many bipartite graphs, then $\Phi(\textsf{IS}_k) = 1$ for all positive integers k and $\Phi(K_{t,t}) = 0$ for all but finitely many t. Hence we can apply Theorem 18 and, in particular, the set $\mathcal{K}$ will contain all but finitely many prime powers and is therefore dense. ◀

Conclusion

We have established hardness for #IndSub(Φ) for *any* (edge-)monotone property Φ that is non-trivial on bipartite graphs. In particular, this holds true even if we count modulo a prime and restrict the input graphs to be bipartite as well. Hence, we did not only significantly extend the set of graph properties Φ for which the (parameterized) complexity of #IndSub(Φ) is understood, but we also generalized many of the prior results, such as [11], [17] and parts of [20] to the cases of bipartite input graphs and modular counting.

As a next step towards a proof of Conjecture 1, we suggest the study of properties that are defined by forbidden induced subgraphs, for which the complexity of #IndSub(Φ) is only partially resolved at this point.

References

1 Shreeram S. Abhyankar. *Lectures on algebra. Vol. I.* World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, 2006. doi:10.1142/9789812773449.

2 Hubie Chen and Stefan Mengel. Counting Answers to Existential Positive Queries: A Complexity Classification. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 315–326, 2016. doi:10.1145/2902251.2902279.

3 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Inf. Comput.*, 201(2):216–231, 2005. doi:10.1016/j.ic.2005.05.001.

4 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.

5 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223, 2017. doi:10.1145/3055399.3055502.

6 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.

7 Jack Edmonds. Paths, Trees, and Flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

8 Jörg Flum and Martin Grohe. The Parameterized Complexity of Counting Problems. *SIAM J. Comput.*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.

9 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

10 Mark Jerrum and Kitty Meeks. Some Hard Families of Parameterized Counting Problems. *TOCT*, 7(3):11:1–11:18, 2015. doi:10.1145/2786017.

11 Mark Jerrum and Kitty Meeks. The parameterised complexity of counting connected subgraphs and graph motifs. *J. Comput. Syst. Sci.*, 81(4):702–716, 2015. doi:10.1016/j.jcss.2014.11.015.

12 Mark Jerrum and Kitty Meeks. The parameterised complexity of counting even and odd induced subgraphs. *Combinatorica*, 37(5):965–990, 2017. doi:10.1007/s00493-016-3338-5.

13 Jeff Kahn, Michael E. Saks, and Dean Sturtevant. A topological approach to evasiveness. *Combinatorica*, 4(4):297–306, 1984. doi:10.1007/BF02579140.

14 Serge Lang. *Algebra*. Graduate Texts in Mathematics. Springer New York, 2005. doi:10.1007/978-1-4613-0041-0.

15 László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012. URL: http://www.ams.org/bookstore-getitem/item=COLL-60.

16 Catherine McCartin. Parameterized counting problems. *Ann. Pure Appl. Logic*, 138(1-3):147–182, 2006. doi:10.1016/j.apal.2005.06.010.

17 Kitty Meeks. The challenges of unbounded treewidth in parameterised subgraph counting problems. *Discrete Applied Mathematics*, 198:170–194, 2016. doi:10.1016/j.dam.2015.06.019.

18 Carl A. Miller. Evasiveness of Graph Properties and Topological Fixed-Point Theorems. *Foundations and Trends in Theoretical Computer Science*, 7(4):337–415, 2013. doi:10.1561/0400000055.

19 Ronald L. Rivest and Jean Vuillemin. On recognizing graph properties from adjacency matrices. *Theoret. Comput. Sci.*, 3(3):371–384, 1976/77. doi:10.1016/0304-3975(76)90053-0.

20 Marc Roth and Johannes Schmitt. Counting Induced Subgraphs: A Topological Approach to #W[1]-hardness. In *13th International Symposium on Parameterized and Exact Computation, IPEC 2018, August 20-24, 2018, Helsinki, Finland*, pages 24:1–24:14, 2018. doi:10.4230/LIPIcs.IPEC.2018.24.

21 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.

22 Leslie G. Valiant. The Complexity of Computing the Permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

Packing Arc-Disjoint Cycles in Tournaments

Stéphane Bessy
Université de Montpellier, LIRMM, CNRS, Montpellier, France
bessy@lirmm.fr

Marin Bougeret
Université de Montpellier, LIRMM, CNRS, Montpellier, France
bougeret@lirmm.fr

R. Krithika
Indian Institute of Technology Palakkad, India
krithika@iitpkd.ac.in

Abhishek Sahu
The Institute of Mathematical Sciences, HBNI, Chennai, India
asahu@imsc.res.in

Saket Saurabh
The Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Bergen, Norway
saket@imsc.res.in

Jocelyn Thiebaut
Université de Montpellier, LIRMM, CNRS, Montpellier, France
thiebaut@lirmm.fr

Meirav Zehavi
Ben-Gurion University, Beersheba, Israel
meiravze@bgu.ac.il

Abstract

A tournament is a directed graph in which there is a single arc between every pair of distinct vertices. Given a tournament T on n vertices, we explore the classical and parameterized complexity of the problems of determining if T has a cycle packing (a set of pairwise arc-disjoint cycles) of size k and a triangle packing (a set of pairwise arc-disjoint triangles) of size k. We refer to these problems as Arc-disjoint Cycles in Tournaments (ACT) and Arc-disjoint Triangles in Tournaments (ATT), respectively. Although the maximization version of ACT can be seen as the linear programming dual of the well-studied problem of finding a minimum feedback arc set (a set of arcs whose deletion results in an acyclic graph) in tournaments, surprisingly no algorithmic results seem to exist for ACT. We first show that ACT and ATT are both NP-complete. Then, we show that the problem of determining if a tournament has a cycle packing and a feedback arc set of the same size is NP-complete. Next, we prove that ACT and ATT are fixed-parameter tractable, they can be solved in $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ time and $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time respectively. Moreover, they both admit a kernel with $\mathcal{O}(k)$ vertices. We also prove that ACT and ATT cannot be solved in $2^{o(\sqrt{k})} n^{\mathcal{O}(1)}$ time under the Exponential-Time Hypothesis.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Complexity classes; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Design and analysis of algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases arc-disjoint cycle packing, tournaments, parameterized algorithms, kernelization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.27

Related Version This paper is based on the two independent manuscripts arXiv:abs/1802.06669 and arXiv:abs/1802.07090.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 27; pp. 27:1–27:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Given a (directed or undirected) graph G and a positive integer k, the Disjoint Cycle Packing problem is to determine whether G has k (vertex or arc/edge) disjoint (directed or undirected) cycles. Packing disjoint cycles is a fundamental problem in Graph Theory and Algorithm Design with applications in several areas. Since the publication of the classic Erdős-Pósa theorem in 1965 [21], this problem has received significant scientific attention in various algorithmic realms. In particular, Vertex-Disjoint Cycle Packing in undirected graphs is one of the first problems studied in the framework of parameterized complexity. In this framework, each problem instance is associated with a non-negative integer k called *parameter*, and a problem is said to be *fixed-parameter tractable* (FPT) if it can be solved in $f(k)n^{\mathcal{O}(1)}$ time for some computable function f, where n is the input size. For convenience, the running time $f(k)n^{\mathcal{O}(1)}$ is denoted as $\mathcal{O}^\star(f(k))$. A *kernelization algorithm* is a polynomial-time algorithm that transforms an arbitrary instance of the problem to an equivalent instance of the same problem whose size is bounded by some computable function g of the parameter of the original instance. The resulting instance is called a *kernel* and if g is a polynomial function, then it is called a *polynomial kernel.* A decidable parameterized problem is FPT if and only if it has a kernel (not necessarily of polynomial size). Kernelization typically involves applying a set *reduction rules* to the given instance to produce another instance. A reduction rule is said to be *safe* if it is sound and complete, i.e., applying it to the given instance produces an equivalent instance. In order to classify parameterized problems as being FPT or not, the W-hierarchy is defined: FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots \subseteq$ XP. It is believed that the subset relations in this sequence are all strict, and a parameterized problem that is hard for some complexity class above FPT in this hierarchy is said to be fixed-parameter intractable. Further details on parameterized algorithms we refer to recent books [16, 19, 24, 26].

Vertex-Disjoint Cycle Packing in undirected graphs is FPT with respect to the solution size k [10, 36] but has no polynomial kernel unless NP $\subseteq$ coNP/poly [11]. In contrast, Edge-Disjoint Cycle Packing in undirected graphs admits a kernel with $\mathcal{O}(k \log k)$ vertices (and is therefore FPT) [11]. On directed graphs, these problems have many practical applications (for example in biology [12, 18]) and they have been extensively studied [7, 34]. It turns out that Vertex-Disjoint Cycle Packing and Arc-Disjoint Cycle Packing are equivalent and are W[1]-hard [33, 42]. Therefore, studying these problems on a subclass of directed graphs is a natural direction of research. Tournaments form a mathematically rich subclass of directed graphs with interesting structural and algorithmic properties [6, 38]. Tournaments have several applications in modeling round-robin tournaments and in the study of voting systems and social choice theory [29, 31].

Feedback Vertex Set and Feedback Arc Set are two well-explored algorithmic problems on tournaments. A *feedback vertex (arc) set* is a set of vertices (arcs) whose deletion results in an acyclic graph. Given a tournament, MinFAST and MinFVST are the problems of obtaining a feedback arc set and feedback vertex set of minimum size, respectively. We refer to the corresponding decision version of the problems as FAST and FVST. The optimization problems MinFAST and MinFVST have numerous practical applications in the areas of voting theory [17], machine learning [15], search engine ranking [20] and have been intensively studied in various algorithmic areas. MinFAST and MinFVST are NP-hard [3, 13] while FAST and FVST are FPT when parameterized by the solution size k [4, 23, 25, 31]. Further, FAST has a kernel with $\mathcal{O}(k)$ vertices and FVST has a kernel with $\mathcal{O}(k^{1.5})$ vertices [9, 35]. Surprisingly, the duals (in the linear programming sense) of MinFAST and MinFVST have not been considered in the literature until recently. Any tournament that has a cycle also has a triangle [7]. Therefore, if a tournament has k vertex-disjoint cycles, then it also has k vertex-disjoint triangles. Thus, Vertex-Disjoint Cycle Packing in tournaments is just packing vertex-disjoint triangles. This problem is NP-hard [8]. A straightforward application of the *colour coding* technique [5] shows that this problem is FPT and a kernel with $\mathcal{O}(k^2)$ vertices is an immediate consequence of the quadratic element kernel known for 3-Set Packing [1]. Recently, a kernel with $\mathcal{O}(k^{1.5})$ vertices was shown for this problem using interesting variants and generalizations of the popular *expansion lemma* [35].

A tournament that has k arc-disjoint cycles need not necessarily have k arc-disjoint triangles. This observation hints that packing arc-disjoint cycles could be significantly harder than packing vertex-disjoint cycles. It also hints that packing arc-disjoint cycles and arc-disjoint triangles in tournaments could be problems of different complexities. This is the starting point of our study. Subsequently, we refer to a set of pairwise arc-disjoint cycles as a *cycle packing* and a set of pairwise arc-disjoint triangles as a *triangle packing*. Given a tournament, MaxACT and MaxATT are the problems of obtaining a maximum set of arc-disjoint cycles and triangles, respectively. We refer to the corresponding decision version of the problems as ACT and ATT. Formally, given a tournament T and a positive integer k, ACT (resp. ATT) is the task of determining if T has k arc-disjoint cycles (resp. triangles). From a structural point of view, the problem of partitioning the arc set of a directed graph into a collection of triangles has been studied for regular tournaments [44], almost regular tournaments [2] and complete digraphs [28]. In this work, we study the classical complexity of MaxACT and MaxATT and the parameterized complexity of ACT and ATT with respect to the solution size (i.e. the number k of cycles/triangles) as parameter.

Our main contributions.

- We prove that MaxATT and MaxACT are NP-hard (Theorems 5 and 7). As a consequence, we also show that ACT and ATT do not admit algorithms with $\mathcal{O}^\star(2^{o(\sqrt{k})})$ running time under the Exponential-Time Hypothesis (Theorem 10). Moreover, deciding if a tournament has a cycle packing and a feedback arc set of the same size is NP-complete (Theorem 9).
- A tournament T has k arc-disjoint cycles if and only if T has k arc-disjoint cycles each of length at most $2k + 1$ (Theorem 11).
- ACT can be solved in $\mathcal{O}^\star(2^{\mathcal{O}(k \log k)})$ time (Theorem 17) and admits a kernel with $\mathcal{O}(k)$ vertices (Theorem 16).
- ATT can be solved in $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ time and admits a kernel with $\mathcal{O}(k)$ vertices (Theorem 18).

2 Preliminaries

We denote the set $\{1, 2, \ldots, n\}$ of consecutive integers from 1 to n by $[n]$.

Directed Graphs. A *directed graph* D (or *digraph*) is a pair consisting of a finite set $V(D)$ of *vertices* of D and a set $A(D)$ of *arcs* of D, which are ordered pairs of elements of $V(D)$. For a vertex $v \in V(D)$, its *out-neighbourhood*, denoted by $N^+(v)$, is the set $\{u \in V(D) \colon vu \in A(D)\}$ and its *out-degree*, denoted by $d^+(x)$, is $|N^+(v)|$. For a set F of arcs, $V(F)$ denotes the union of the sets of endpoints of arcs in F. Given a digraph D and a subset X of vertices, we denote by $D[X]$ the digraph induced by the vertices in X. Moreover, we denote by $D \setminus X$ the digraph $D[V(D) \setminus X]$ and say that this digraph is obtained by *deleting* X *from* D.

Paths and Cycles. A *path* P in a digraph D is a sequence $(v_1, \dots, v_k)$ of distinct vertices such that for each $i \in [k-1]$, $v_i v_{i+1} \in A(D)$. The set $\{v_1, \dots, v_k\}$ is denoted by $V(P)$ and the set $\{v_i v_{i+1} \colon i \in [k-1]\}$ is denoted by $A(P)$. A *cycle* C in D is a sequence $(v_1, \dots, v_k)$ of distinct vertices such that $(v_1, \dots, v_k)$ is a path and $v_k v_1 \in A(D)$. The length of a path or cycle X is the number of vertices in it. A cycle on three vertices is called a *triangle*. A digraph is called a *directed acyclic graph* if it has no cycles. A *feedback arc set* (FAS) is a set of arcs whose deletion results in an acyclic graph. For a digraph D, let $\mathrm{minfas}(D)$ denote the size of a minimum FAS of D. Any directed acyclic graph D has an ordering $\sigma(D) = (v_1, \dots, v_n)$ called *topological ordering* of its vertices such that for each $v_i v_j \in A(D)$, $i < j$ holds. Given an ordering σ and two vertices u and v, we write $u <_\sigma v$ if u is before v in σ.

Tournaments. A *tournament* T is a digraph in which for every pair u, v of distinct vertices either $uv \in A(T)$ or $vu \in A(T)$ but not both. In other words, a tournament T on n vertices is an orientation of the complete graph K_n. A tournament T can alternatively be defined by an ordering $\sigma(T) = (v_1, \dots, v_n)$ of its vertices and a set of *backward arcs* $\overleftarrow{A}_\sigma(T)$ (which will be denoted $\overleftarrow{A}(T)$ as the considered ordering is not ambiguous), where each arc $a \in \overleftarrow{A}(T)$ is of the form $v_{i_1} v_{i_2}$ with $i_2 < i_1$. Indeed, given $\sigma(T)$ and $\overleftarrow{A}(T)$, we define $V(T) = \{v_i : i \in [n]\}$ and $A(T) = \overleftarrow{A}(T) \cup \overrightarrow{A}(T)$ where $\overrightarrow{A}(T) = \{v_{i_1} v_{i_2} : (i_1 < i_2) \text{ and } v_{i_2} v_{i_1} \notin \overleftarrow{A}(T)\}$ is the set of *forward arcs* of T in the given ordering $\sigma(T)$. The pair $(\sigma(T), \overleftarrow{A}(T))$ is called a *linear representation* of the tournament T. A tournament is called *transitive* if it is a directed acyclic graph and a transitive tournament has a unique topological ordering. Given two tournaments T_1, T_2 defined by $\sigma(T_l)$ and $\overleftarrow{A}(T_l)$ with $l \in \{1, 2\}$, we denote by $T = T_1 T_2$ the tournament called the *concatenation of* T_1 *and* T_2, where $V(T) = V(T_2) \cup V(T_2)$, $\sigma(T) = \sigma(T_1)\sigma(T_2)$ is the concatenation of the two sequences, and $\overleftarrow{A}(T) = \overleftarrow{A}(T_1) \cup \overleftarrow{A}(T_2)$.

3 NP-hardness of MaxACT and MaxATT

This section contains our main results. We prove the NP-hardness of MaxATT using a reduction from 3-SAT(3). Here, 3-SAT(3) denotes the specific case of 3-SAT where each clause has at most three literals, and each literal appears at most two times positively and exactly one time negatively. In the following, denote by F the input formula of an instance of 3-SAT(3). Let n be the number of its variables and m be the number of its clauses. We may suppose that $n \equiv 3 \pmod 6$. If it is not the case, we can add up to 5 unused variables x with the trivial clause $x \vee \overline{x}$. This operation guarantees us we keep the hypotheses of 3-SAT(3). We can also assume that $m + 1 \equiv 3 \pmod 6$. Indeed, if it not the case, we add 6 new unused variables $x_1, \dots, x_6$ with the 6 trivial clauses $x_i \vee \overline{x_i}$, and the clause $x_1 \vee x_2$. This padding process keep both the 3-SAT(3) structure and $n \equiv 3 \pmod 6$. From F we construct a tournament T which is the concatenation of two tournaments T_v and T_c defined below.

In the following, let f be the reduction that maps an instance F of 3-SAT(3) to a tournament T we describe now.

The variable tournament T_v. For each variable v_i of F, we define a tournament V_i of order 6 as follows: $\sigma_i(V_i) = (r_i, \bar{x}_i, x_i^1, s_i, x_i^2, t_i)$ and $\overleftarrow{A}_\sigma(V_i) = \{s_i r_i, t_i x_i^1\}$. Figure 1 is a representation of one variable gadget V_i. One can notice that the minimum FAS of V_i corresponds exactly to the set of its backward arcs. We now define $V(T_v)$ be the union of the vertex sets of the V_is and we equip T_v with the order $\sigma_1 \sigma_2 \dots \sigma_n$. Thus, T_v has $6n$

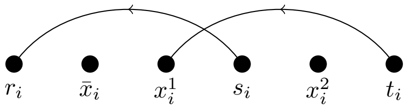


Figure 1 The variable gadget V_i. Only backward arcs are depicted, so all the remaining arcs are forward arcs.

vertices. We also add the following backward arcs to T_v. Since $n \equiv 3 \pmod 6$, there is an edge-disjoint (undirected) triangle packing of K_n covering all its edges with triangles that can be computed in polynomial time [32]. Let $\{u_1, \ldots, u_n\}$ be an arbitrary enumeration of the vertices of K_n. Using a perfect triangle packing Δ_{K_n} of K_n, we create a tournament T_{K_n} such that $\sigma'(T_{K_n}) = (u_1, \ldots, u_n)$ and $\overleftarrow{A}_{\sigma'}(T_{K_n}) = \{u_k u_i : (u_i, u_j, u_k) \text{ is a triangle of } \Delta_{K_n} \text{ with } i < j < k\}$. Now we set $\overleftarrow{A}_\sigma(T_v) = \{xy : x \in V(V_i), y \in V(V_j) \text{ for } i \neq j \text{ and } u_j u_i \in \overleftarrow{A}_{\sigma'}(T_{K_n})\} \cup \bigcup_{i=1}^{n} \overleftarrow{A}_\sigma(V_i)$. In some way, we "blew up" every vertex u_i of T_{K_n} into our variable gadget V_i.

The clause tournament T_c. For each of the m clauses c_j of F, we define a tournament C_j of order 3 as follows: $\sigma(C_j) = (c_j^1, c_j^2, c_j^3)$ and $\overleftarrow{A}_\sigma(C_j) = \emptyset$. In addition, we have a $(m+1)^{th}$ tournament denoted by C_{m+1} and defined by $\sigma(C_{m+1}) = (c_{m+1}^1, c_{m+1}^2, c_{m+1}^3)$ and $\overleftarrow{A}_\sigma(C_{m+1}) = \{c_{m+1}^3 c_{m+1}^1\}$, that is C_{m+1} is a triangle. We call this triangle the *dummy triangle* , and its vertices the *dummy vertices*. We now define T_c such that $\sigma(T_c)$ is the concatenation of each ordering $\sigma(C_j)$ in the natural order, that is $\sigma(T_c) = (c_1^1, c_1^2, c_1^3, \ldots, c_m^1, c_m^2, c_m^3, c_{m+1}^1, c_{m+1}^2, c_{m+1}^3)$. So T_c has $3(m+1)$ vertices. Since $m + 1 \equiv 3 \pmod 6$, we use the same trick as above to add arcs to $\overleftarrow{A}_\sigma(T_c)$ coming from a perfect packing of undirected triangles of K_{m+1}. Once again, we "blew up" every vertex u_j of $T_{K_{m+1}}$ into our clause gadget C_j.

The tournament T. To define our final tournament T let us begin with its ordering σ defined by $\sigma(T) = \sigma(T_v)\sigma(T_c)$. Then we construct $\overleftarrow{A}^{vc}(T)$ the backward arcs between T_c and T_v. For any $j \in [m]$, if the clause c_j in F has three literals, that is $c_j = \ell_1 \vee \ell_2 \vee l_3$, then we add to $\overleftarrow{A}^{vc}(T)$ the three backward arcs $c_j^3 z_u$ where $u \in [3]$ and such that $z_u = \bar{x}_{i_u}$ when $\ell_u = \bar{v}_{i_u}$, and $z_u \in \{x_{i_u}^1, x_{i_u}^2\}$ when $\ell_u = v_{i_u}$ in such a way that for any $i \in [n]$, there exists a unique arc $a \in \overleftarrow{A}^{vc}(T)$ with $h(a) = x_i^1$. Informally, in the previous definition, if $x_{i_u}^1$ is already "used" by another clause, we chose $z_u = x_{i_u}^2$. Such an orientation will always be possible since each variable occurs at most two times positively and once negatively in F. If the clause c_j in F has only two literals, that is $c_j = \ell_1 \vee \ell_2$, then we add in $\overleftarrow{A}^{vc}(T)$ the two backward arcs $c_j^2 z_u$ where $u \in [2]$ and such that $z_u = \bar{x}_{i_u}$ when $\ell_u = \bar{v}_{i_u}$ and $z_u \in \{x_{i_u}^1, x_{i_u}^2\}$ when $\ell_u = v_{i_u}$ in such a way that for any $i \in [n]$, there exists a unique arc $a \in \overleftarrow{A}^{vc}(T)$ with $h(a) = x_i^1$. Finally, we add in $\overleftarrow{A}^{vc}(T)$ the backward arcs $c_{m+1}^u \bar{x}_i$ for any $u \in [3]$ and $i \in [n]$. These arcs are called *dummy arcs*. We set $\overleftarrow{A}_\sigma(T) = \overleftarrow{A}_\sigma(T_v) \cup \overleftarrow{A}_\sigma(T_c) \cup \overleftarrow{A}^{vc}(T)$. Notice that each $\bar{x}_i$ has exactly four arcs $a \in \overleftarrow{A}_\sigma(T)$ such that $h(a) = \bar{x}_i$ and $t(a)$ is a vertex of T_c. To finish the construction, notice also that T has $6n+3(m+1)$ vertices and can be computed in polynomial time. Figure 2 is an example of the tournament obtained from a trivial 3-SAT(3) instance.

Now, we move on to proving the correctness of the reduction. First of all, observe that in each variable gadget V_i, there are only four triangles: let δ_i^1, δ_i^2, δ_i^3 and δ_i^4 be the triangles $(r_i, \bar{x}_i, s_i)$, (r_i, x_i^1, s_i), (x_i^1, s_i, t_i) and (x_i^1, x_i^2, t_i), respectively. Moreover, notice that there are only three maximal triangle packings of V_i which are $\{\delta_i^1, \delta_i^3\}$, $\{\delta_i^1, \delta_i^4\}$ and $\{\delta_i^2, \delta_i^4\}$. We call these packings $\Delta_i^\top$, $\Delta_i^{\top'}$ and $\Delta_i^\perp$, respectively.

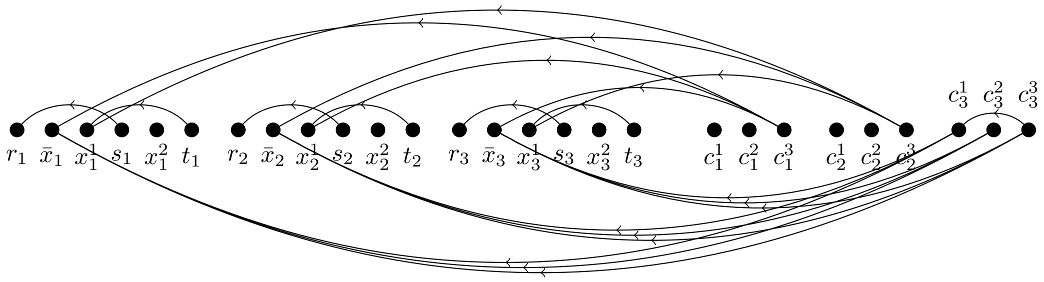


Figure 2 Example of reduction obtained when $F = \{c_1, c_2\}$ where $c_1 = \bar{v}_1 \vee v_2 \vee \bar{v}_3$ and $c_2 = v_1 \vee \bar{v}_2 \vee v_3$. Forward arcs are not depicted. In addition to the depicted backward arcs, we have the 36 backward arcs from V_3 to V_1, and the 9 backward arcs from C_3 to C_1.

Given a triangle packing Δ of T and a subset X of vertices, we define for any $x \in X$ the Δ-local out-degree of the vertex x, denoted $d^+_{X\setminus\Delta}(x)$, as the remaining out-degree of x in $T[X]$ when we remove the arcs of the triangles of Δ. More formally, we set: $d^+_{X\setminus\Delta}(x) = |\{xa\colon a \in X, xa \in A[X], xa \notin A(\Delta)\}|$.

▶ Remark 1. Given a variable gadget V_i, we have:

(i) $d^+_{V_i\setminus\Delta_i^\top}(x_i^1) = d^+_{V_i\setminus\Delta_i^\top}(x_i^2) = 1$ and $d^+_{V_i\setminus\Delta_i^\top}(\bar{x}_i) = 3$,

(ii) $d^+_{V_i\setminus\Delta_i^{\top\prime}}(x_i^1) = 1$, $d^+_{V_i\setminus\Delta_i^{\top\prime}}(x_i^2) = 0$ and $d^+_{V_i\setminus\Delta_i^{\top\prime}}(\bar{x}_i) = 3$,

(iii) $d^+_{V_i\setminus\Delta_i^\perp}(x_i^1) = d^+_{V_i\setminus\Delta_i^\perp}(x_i^2) = 0$ and $d^+_{V_i\setminus\Delta_i^\perp}(\bar{x}_i) = 4$,

(iv) none of $\bar{x}_i x_i^1$, $\bar{x}_i x_i^2$, $\bar{x}_i t_i$ belongs to $\Delta_i^\top$ or $\Delta_i^\perp$.

Informally, we want to set the variable x_i to true (resp. false) when one of the locally-optimal $\Delta_i^{\top\prime}$ or $\Delta_i^\top$ (resp. $\Delta_i^\perp$) is taken in the variable gadget V_i in the global solution. Now given a triangle packing Δ of T, we partition Δ into the following sets:

- $\Delta_{V,V,V} = \{(a,b,c) \in \Delta : a \in V_i,\ b \in V_j,\ c \in V_k$ with $i < j < k\}$,
- $\Delta_{V,V,C} = \{(a,b,c) \in \Delta : a \in V_i,\ b \in V_j,\ c \in C_k$ with $i < j\}$,
- $\Delta_{V,C,C} = \{(a,b,c) \in \Delta : a \in V_i,\ b \in C_j,\ c \in C_k$ with $j < k\}$,
- $\Delta_{C,C,C} = \{(a,b,c) \in \Delta : a \in C_i,\ b \in C_j,\ c \in C_k$ with $i < j < k\}$,
- $\Delta_{2V,C} = \{(a,b,c) \in \Delta : a,\ b \in V_i,\ c \in C_j\}$,
- $\Delta_{V,2C} = \{(a,b,c) \in \Delta : a \in V_i, b,\ c \in C_j\}$,
- $\Delta_{3V} = \{(a,b,c) \in \Delta : a,\ b,\ c \in V_i\}$,
- $\Delta_{3C} = \{(a,b,c) \in \Delta : a,\ b,\ c \in C_i\}$.

Notice that in T, there is no triangle with two vertices in a variable gadget V_i and its third vertex in a variable gadget V_j with $i \neq j$ since all the arcs between two variable gadgets are oriented in the same direction. We have the same observation for clauses.
In the two next lemmas, we prove some properties concerning the solution Δ, which imply the result of Lemma 4.

▶ **Lemma 2.** *There exists a triangle packing Δ^v (resp. Δ^c) which uses exactly the arcs between distinct variable gadgets (resp. clause gadgets). Therefore, we have $|\Delta_{V,V,V}| \leq 6n(n-1)$ and $|\Delta_{C,C,C}| \leq 3m(m+1)/2$ and these bounds are tight.*

Proof. First recall that the tournament T_v is constructed from a tournament T_{K_n} which admits a perfect packing of $n(n-1)/6$ triangles. Then we replaced each vertex u_i in T_{K_n} by the variable gadget V_i and kept all the arcs between two variable gadgets V_i

and V_j in the same orientation as between u_i and u_j. Let $u_iu_ju_k$ be a triangle of the perfect packing of T_{K_n}. We temporally relabel the vertices of V_i, V_j and V_k respectively by $\{f_i, i \in [6]\}$, $\{g_i, i \in [6]\}$ and $\{h_i, i \in [6]\}$ and consider the tripartite tournament $K_{6,6,6}$ given by $V(K_{6,6,6}) = \{f_i, g_i, h_i, i \in [6]\}$ and $A(K_{6,6,6}) = \{f_ig_j, g_ih_j, h_if_j : i, j \in [6]\}$. Then it is easy to check that $\{(f_i, g_j, h_{i+j \pmod 6}) : i, j \in [6]\}$ is a perfect triangle packing of $K_{6,6,6}$. Since every triangle of T_{K_n} becomes a $K_{6,6,6}$ in T_v, we can find a triangle packing Δ^v which use all the arcs between disjoint variable gadgets. We use the same reasoning to prove that there exists a triangle packing Δ^c which use all the arcs available in T_c between two distinct clause gadget. ◀

▶ **Lemma 3.** *For any triangle packing Δ of the tournament T, we have:*
(i) $|\Delta_{V,V,V}| + |\Delta_{C,C,C}| \leq 6n(n-1) + 3m(m+1)/2,$
(ii) $|\Delta_{2V,C}| + |\Delta_{V,2C}| + |\Delta_{V,C,C}| + |\Delta_{V,V,C}| \leq |\overleftarrow{A}^{vc}(T)|,$
(iii) $|\Delta_{3V}| \leq 2n,$
(iv) $|\Delta_{3C}| \leq 1.$
Therefore in total we have $|\Delta| \leq 6n(n-1) + 3m(m+1)/2 + 2n + |\overleftarrow{A}^{vc}(T)| + 1.$

Proof. Let Δ be a triangle packing of T. Recall that we have: $|\Delta| = |\Delta_{V,V,V}| + |\Delta_{V,V,C}| + |\Delta_{V,C,C}| + |\Delta_{C,C,C}| + |\Delta_{2V,C}| + |\Delta_{V,2C}| + |\Delta_{3V}| + |\Delta_{3C}|$. First, inequality (i) comes from Lemma 2. Then, we have $|\Delta_{2V,C}| + |\Delta_{V,2C}| + |\Delta_{V,C,C}| + |\Delta_{V,V,C}| \leq |\overleftarrow{A}^{vc}(T)|$ since every triangle of these sets consumes one backward arc from T_c to T_v. We have $|\Delta_{3V}| \leq 2n$ since we have at most 2 disjoint triangles in each variable gadget. Finally we also have $|\Delta_{3C}| \leq 1$ since the dummy triangle is the only triangle lying in a clause gadget. ◀

▶ **Lemma 4.** *F is satisfiable if and only if there exists a triangle packing Δ of size $6n(n-1) + 3m(m+1)/2 + 2n + |\overleftarrow{A}^{vc}(T)| + 1$ in the tournament T.*

As 3-SAT(3) is NP-hard [39, 43], this implies the following theorem.

▶ **Theorem 5.** MaxATT *is NP-hard.*

As mentioned in the introduction, packing arc-disjoint cycles is not necessarily equivalent to packing arc-disjoint triangles. Thus, we need to establish the following lemma to transfer the previous NP-hardness result to MaxACT.

▶ **Lemma 6.** *Given a* 3-SAT(3) *instance F, and T the tournament constructed from F with the reduction f, we have a triangle packing Δ of T of size* $6n(n-1) + 3m(m+1)/2 + 2n + |\overleftarrow{A}^{vc}(T)| + 1$ *if and only if there is a cycle packing O of the same size.*

The previous lemma and Theorem 5 imply the following theorem.

▶ **Theorem 7.** MaxACT *is NP-hard.*

Let us now define two special cases Tight-ATT (resp. Tight-ACT) where, given a tournament T and a linear ordering σ with k backward arcs, where $k = \text{minfas}(T)$, the goal is to decide if there is a triangle (resp. cycle) packing of size k. We call these special cases the "tight" versions of the classical packing problems because as the input admits an FAS of size k, any triangle (or cycle) packing has size at most k. We have the following result, directly implying the NP-hardness of Tight-ATT and Tight-ACT.

▶ **Lemma 8.** *Let T be a tournament constructed by the reduction f, and k be the threshold value defined in Lemma 4. Then, we have $k = \text{minfas}(T)$ and we can construct (in polynomial time) an ordering of T with k backward arcs.*

▶ **Theorem 9.** Tight-ATT *and* Tight-ACT *are NP-hard.*

Finally, the size s of the required packing in Lemma 4 satisfies $s = \mathcal{O}((n+m)^2)$. Under the Exponential-time Hypothesis, the problem 3-SAT cannot be solved in $2^{o(n+m)}$ [16, 30]. Then, using the linear reduction from 3-SAT to 3-SAT(3) [43], we also get the following result.

▶ **Theorem 10.** *Under the Exponential-time Hypothesis,* ATT *and* ACT *cannot be solved in* $\mathcal{O}^\star(2^{o(\sqrt{k})})$ *time.*

In the framework of parameterizing above guaranteed values [37], the above results imply that ACT parameterized below the guaranteed value of the size of a minimal feedback arc set is fixed-parameter intractable.

4 Parameterized Complexity of ACT

The classical Erdős-Pósa theorem for cycles in undirected graphs states that for each non-negative integer k, every undirected graph either contains k vertex-disjoint cycles or has a feedback vertex set consisting of $f(k) = \mathcal{O}(k \log k)$ vertices [21]. An interesting consequence of this theorem is that it leads to an FPT algorithm for Vertex-Disjoint Cycle Packing (see [36] for more details).

Analogous to these results, we prove an Erdős-Pósa type theorem for tournaments and show that it leads to an $\mathcal{O}^\star(2^{\mathcal{O}(k \log k)})$ time algorithm and a linear vertex kernel for ACT. First we obtain the following result.

▶ **Theorem 11.** *Let k and r be positive integers such that $r \leq k$. A tournament T contains a set of r arc-disjoint cycles if and only if T contains a set of r arc-disjoint cycles each of length at most $2k+1$.*

Proof. The reverse direction of the claim holds trivially. Let us now prove the forward direction. Let $\mathcal{C}$ be a set of r arc-disjoint cycles in T that minimizes $\sum_{C \in \mathcal{C}} |C|$. If every cycle in $\mathcal{C}$ is a triangle, then the claim trivially holds. Otherwise, let C be a longest cycle in $\mathcal{C}$ and let ℓ denote its length. Let v_i, v_j be a pair of non-consecutive vertices in C. Then, either $v_iv_j \in A(T)$ or $v_jv_i \in A(T)$. In any case, the arc e between v_i and v_j along with $A(C)$ forms a cycle C' of length less than ℓ with $A(C') \setminus \{e\} \subset A(C)$. By our choice of $\mathcal{C}$, this implies that e is an arc in some other cycle $\widehat{C} \in \mathcal{C}$. This property is true for the arc between any pair of non-consecutive vertices in C. Therefore, we have $\binom{\ell}{2} - \ell \leq \ell(k-1)$ leading to $\ell \leq 2k+1$. ◀

This result essentially shows that it suffices to determine the existence of k arc-disjoint cycles in T each of length at most $2k+1$ in order to determine if (T,k) is an yes-instance of ACT. This immediately leads to a quadratic Erdős-Pósa bound. That is, for every non-negative integer k, every tournament T either contains k arc-disjoint cycles or has an FAS of size $\mathcal{O}(k^2)$. Next, we strengthen this result to arrive at a linear bound.

We will use the following lemma known from [14] in order to prove Theorem 13[1]. For a digraph D, let $\Lambda(D)$ denote the number of non-adjacent pairs of vertices in D. That is, $\Lambda(D)$ is the number of pairs u, v of vertices of D such that neither $uv \in A(D)$ nor $vu \in A(D)$.

[1] The authors would like to thank F. Havet for pointing out that Lemma 12 was a consequence of a result by Chudnovsky et al. [14], as well for an improvement of the constant in Theorem 13.

▶ **Lemma 12.** [14] *Let D be a triangle-free digraph in which for every pair u, v of distinct vertices, at most one of uv or vu is in $A(D)$. Then, we can compute an FAS of size at most $\Lambda(D)$ in polynomial time.*

▶ **Theorem 13.** *For every non-negative integer k, every tournament T either contains k arc-disjoint triangles or has an FAS of size at most $5(k-1)$ that can be obtained in polynomial time.*

Proof. Let $\mathcal{C}$ be a maximal set of arc-disjoint triangles in T (that can be obtained greedily in polynomial time). If $|\mathcal{C}| \geq k$, then we have the required set of triangles. Otherwise, let D denote the digraph obtained from T by deleting the arcs that are in some triangle in $\mathcal{C}$. Clearly, D has no triangle and $\Lambda(D) \leq 3(k-1)$. Let F be an FAS of D obtained in polynomial time using Lemma 12. Then, we have $|F| \leq 3(k-1)$. Next, consider a topological ordering σ of $D - F$. Each triangle of $\mathcal{C}$ contains at most 2 arcs which are backward in this ordering. If we denote by F' the set of all the arcs of the triangles of $\mathcal{C}$ which are backward in σ, then we have $|F'| \leq 2(k-1)$ and $(D-F)-F'$ is acyclic. Thus $F^* = F \cup F'$ is an FAS of T satisfying $|F^*| \leq 5(k-1)$. ◀

Next, we show how to obtain a linear kernel for ACT. This kernel is inspired by the linear kernelization described in [9] for FAST and uses Theorem 13. Let T be a tournament on n vertices. First, we apply the following reduction rule.

▶ **Reduction Rule 4.1.** *If a vertex v is in no cycle, then delete v from T.*

This rule is clearly safe as our goal is to find k cycles and v cannot be in any of them. To describe our next rule, we need to state a result by Bessy et al. [9]. An *interval* is a consecutive set of vertices in a linear representation $(\sigma(T), \overleftarrow{A}(T))$ of a tournament T.

▶ **Lemma 14** ([9])**.** *Let $T = (\sigma(T), \overleftarrow{A}(T))$ be a tournament on which Reduction Rule 4.1 is not applicable. If $|V(T)| \geq 2|\overleftarrow{A}(T)| + 1$, then there exists a partition $\mathcal{J}$ of $V(T)$ into intervals (that can be computed in polynomial time) such that there are $|\overleftarrow{A}(T) \cap E| > 0$ arc-disjoint cycles using only arcs in E where E denotes the set of arcs in T with endpoints in different intervals.*

Our reduction rule that is based on this lemma is as follows.

▶ **Reduction Rule 4.2.** *Let $T = (\sigma(T), \overleftarrow{A}(T))$ be a tournament on which Reduction Rule 4.1 is not applicable. Let $\mathcal{J}$ be a partition of $V(T)$ into intervals satisfying the properties specified in Lemma 14. Reverse all arcs in $\overleftarrow{A}(T) \cap E$ and decrease k by $|\overleftarrow{A}(T) \cap E|$ where E denotes the set of arcs in T with endpoints in different intervals.*

▶ **Lemma 15.** *Reduction Rule 4.2 is safe.*

Proof. Let T' be the tournament obtained from T by reversing all arcs in $\overleftarrow{A}(T) \cap E$. Suppose T' has $k - |\overleftarrow{A}(T) \cap E|$ arc-disjoint cycles. Then, it is guaranteed that each such cycle is completely contained in an interval. This is due to the fact that T' has no backward arc with endpoints in different intervals. Indeed, if a cycle in T' uses a forward (backward) arc with endpoints in different intervals, then it also uses a back (forward) arc with endpoints in different intervals. It follows that for each arc $uv \in E$, neither uv nor vu is used in these $k - |\overleftarrow{A}(T) \cap E|$ cycles. Hence, these $k - |\overleftarrow{A}(T) \cap E|$ cycles in T' are also cycles in T. Then, we can add a set of $|\overleftarrow{A}(T) \cap E|$ cycles obtained from the second property of Lemma 14 to these $k - |\overleftarrow{A}(T) \cap E|$ cycles to get k cycles in T. Conversely, consider a set of k cycles in

T. As argued earlier, we know that the number of cycles that have an arc that is in E is at most $|\overleftarrow{A}(T) \cap E|$. The remaining cycles (at least $k - |\overleftarrow{A}(T) \cap E|$ of them) do not contain any arc that is in E, in particular, they do not contain any arc from $\overleftarrow{A}(T) \cap E$. Therefore, these cycles are also cycles in T'. ◀

Thus, we have the following result.

▶ **Theorem 16.** ACT *admits a kernel with* $\mathcal{O}(k)$ *vertices.*

Proof. Let (T, k) denote the instance obtained from the input instance by applying Reduction Rule 4.1 exhaustively. From Lemma 13, we know that either T has k arc-disjoint triangles or has an FAS of size at most $5(k-1)$ that can be obtained in polynomial time. In the first case, we return a trivial yes-instance of constant size as the kernel. In the second case, let F be the FAS of size at most $5(k-1)$ of T. Let $(\sigma(T), \overleftarrow{A}(T))$ be the linear representation of T where $\sigma(T)$ is a topological ordering of the vertices of the directed acyclic graph $T - F$. As $V(T-F) = V(T)$, $|\overleftarrow{A}(T)| \leq 5(k-1)$. If $|V(T)| \geq 10k - 9$, then from Lemma 14, there is a partition of $V(T)$ into intervals with the specified properties. Therefore, Reduction Rule 4.2 is applicable (and the parameter drops by at least 1). When we obtain an instance where neither of the Reduction Rules 4.1 and 4.2 is applicable, it follows that the tournament in that instance has at most $10k$ vertices. ◀

Finally, we show that ACT can be solved in $\mathcal{O}^\star(2^{\mathcal{O}(k \log k)})$ time. The idea is to reduce the problem to the following Arc-Disjoint Paths problem in directed acyclic graphs: given a digraph D on n vertices and k ordered pairs $(s_1, t_1), \ldots, (s_k, t_k)$ of vertices of D, do there exist arc-disjoint paths $P_1, \ldots, P_k$ in D such that P_i is a path from s_i to t_i for each $i \in [k]$? On directed acyclic graphs, Arc-Disjoint Paths is known to be NP-complete [22], W[1]-hard [42] with respect to k as parameter and solvable in $n^{\mathcal{O}(k)}$ time [27]. Despite its fixed-parameter intractability, we will show that we can use the $n^{\mathcal{O}(k)}$ algorithm and Theorems 13 and 16 to describe an FPT algorithm for ACT.

▶ **Theorem 17.** ACT *can be solved in* $\mathcal{O}^\star(2^{\mathcal{O}(k \log k)})$ *time.*

Proof. Consider an instance (T, k) of ACT. Using Theorem 16, we obtain a kernel $\mathcal{I} = (\widehat{T}, \widehat{k})$ such that $\widehat{T}$ has $\mathcal{O}(k)$ vertices. Further, $\widehat{k} \leq k$. By definition, (T, k) is an yes-instance if and only if $(\widehat{T}, \widehat{k})$ is an yes-instance. Using Theorem 13, we know that $\widehat{T}$ either contains $\widehat{k}$ arc-disjoint triangles or has an FAS of size at most $5(\widehat{k} - 1)$ that can be obtained in polynomial time. If Theorem 13 returns a set of $\widehat{k}$ arc-disjoint triangles in $\widehat{T}$, then we declare that (T, k) is an yes-instance.

Otherwise, let $\widehat{F}$ be the FAS of size at most $5(\widehat{k} - 1)$ returned by Theorem 13. Let D denote the (acyclic) digraph obtained from $\widehat{T}$ by deleting $\widehat{F}$. Observe that D has $\mathcal{O}(k)$ vertices. Suppose $\widehat{T}$ has a set $\mathcal{C} = \{C_1, \ldots, C_{\widehat{k}}\}$ of $\widehat{k}$ arc-disjoint cycles. For each $C \in \mathcal{C}$, we know that $A(C) \cap \widehat{F} \neq \emptyset$ as $\widehat{F}$ is an FAS of $\widehat{T}$. We can guess that subset F of $\widehat{F}$ such that $F = \widehat{F} \cap A(\mathcal{C})$. Then, for each cycle $C_i \in \mathcal{C}$, we can guess the arcs F_i from F that it contains and also the order π_i in which they appear. This information is captured as a partition $\mathcal{F}$ of F into $\widehat{k}$ sets, F_1 to $F_{\widehat{k}}$ and the set $\{\pi_1, \ldots, \pi_{\widehat{k}}\}$ of permutations where π_i is a permutation of F_i for each $i \in [\widehat{k}]$. Any cycle C_i that has $F_i \subseteq F$ contains a (v, x)-path between every pair (u, v), (x, y) of consecutive arcs of F_i with arcs from $A(D)$. That is, there is a path from $\mathrm{h}(\pi_i^{-1}(j))$ and $\mathrm{t}(\pi_i^{-1}((j+1) \mod |F_i|))$ with arcs from D for each $j \in [|F_i|]$. The total number of such paths in these $\widehat{k}$ cycles is $\mathcal{O}(|F|)$ and the arcs of these paths are contained in D which is a (simple) directed acyclic graph.

The number of choices for F is $2^{|\widehat{F}|}$ and the number of choices for a partition $\mathcal{F} = \{F_1, \ldots, F_{\widehat{k}}\}$ of F and a set $X = \{\pi_1, \ldots, \pi_{\widehat{k}}\}$ of permutations is $2^{\mathcal{O}(|\widehat{F}| \log |\widehat{F}|)}$. Once such a choice is made, the problem of finding $\widehat{k}$ arc-disjoint cycles in $\widehat{T}$ reduces to the problem of finding $\widehat{k}$ arc-disjoint cycles $\mathcal{C} = \{C_1, \ldots, C_{\widehat{k}}\}$ in $\widehat{T}$ such that for each $1 \leq i \leq \widehat{k}$ and for each $1 \leq j \leq |F_i|$, C_i has a path P_{ij} between $\mathrm{h}(\pi_i^{-1}(j))$ and $\mathrm{t}(\pi_i^{-1}((j+1) \mod |F_i|))$ with arcs from $D = \widehat{T} - \widehat{F}$. This problem is essentially finding $r = \mathcal{O}(|\widehat{F}|)$ arc-disjoint paths in D and can be solved in $|V(D)|^{\mathcal{O}(r)}$ time using the algorithm in [27]. Therefore, the overall running time of the algorithm is $\mathcal{O}^\star(2^{\mathcal{O}(k \log k)})$ as $|V(D)| = \mathcal{O}(k)$ and $r = \mathcal{O}(k)$. ◀

5 Parameterized Complexity of ATT

It is easy to obtain an $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ time algorithm using the classical colour coding technique [5] for packing subgraphs of bounded size, and in particular for ATT. Moreover, using matching techniques, we also provide a kernel with a linear number of vertices.

In this section, we provide an FPT algorithm and a linear vertex kernel for ATT. First, it is easy to obtain an $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ time algorithm using the classical colour coding technique [5] for packing subgraphs of bounded size.

▶ **Theorem 18.** ATT *can be solved in* $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ *time.*

Proof. Consider an instance $\mathcal{I} = (T, k)$ of ATT. Let n denote $|V(T)|$ and m denote $|A(T)|$. Let $\mathcal{F}$ denote the family of colouring functions $c : A(T) \to [3k]$ of size $2^{\mathcal{O}(k)} \log^2 m$ that can be computed in $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ time using $3k$-perfect family of hash functions [41]. For each colouring function c in $\mathcal{F}$, we colour $A(T)$ according to c and find a triangle packing of size k whose arcs use different colours. We use a standard dynamic programming routine to finding such a triangle packing. Clearly, if $\mathcal{I}$ is an yes-instance and $\mathcal{C}$ is a set of k arc-disjoint triangles in T, there is a colouring function in $\mathcal{F}$ that colours the $3k$ arcs in these triangles with distinct colours and our algorithm will find the required triangle packing. Given a colouring $c \in \mathcal{F}$, we first compute for every set of 3 colours $\{a, b, c\}$ whether the arcs coloured with a, b or c induce a triangle using 3 different colours or not. Then, for every set S of $3(p+1)$ colours with $p \in [k-1]$, we recursively test if the arcs coloured with the colours in S induce $p+1$ arc-disjoint triangles whose arcs use all the colours of S. This is achieved by iterating over every subset $\{a, b, c\}$ of S and checking if there is a triangle using colours a, b and c and a collection of p arc-disjoint triangles whose arcs use all the colours of $S \setminus \{a, b, c\}$. For a given S, we can find this collection of triangles in $\mathcal{O}(p^3) = \mathcal{O}(k^3)$ time. Therefore, the overall running time of the algorithm is $\mathcal{O}^\star(2^{\mathcal{O}(k)})$. ◀

Next, we show that ATT has a linear vertex kernel.

▶ **Theorem 19.** ATT *admits a kernel with* $\mathcal{O}(k)$ *vertices.*

Proof. Let $\mathcal{X}$ be a maximal collection of arc-disjoint triangles of a tournament T obtained greedily. Let $V_\mathcal{X}$ denote the vertices of the triangles in $\mathcal{X}$ and $A_\mathcal{X}$ denote the arcs of $V_\mathcal{X}$. Let U be the remaining vertices of $V(T)$, i.e., $U = V(T) \setminus V_\mathcal{X}$. If $|\mathcal{X}| \geq k$, then (T, k) is an yes-instance of ATT. Otherwise, $|\mathcal{X}| < k$ and $|V_\mathcal{X}| < 3k$. Moreover, notice that $T[U]$ is acyclic and T does not contain a triangle with one vertex in $V_\mathcal{X}$ and two in vertices in U (otherwise $\mathcal{X}$ would not be maximal).

Let B be the (undirected) bipartite graph defined by $V(B) = A_\mathcal{X} \cup U$ and $E(B) = \{au\colon a \in A_\mathcal{X}, u \in U$ such that $(t(a), h(a), u)$ forms a triangle in $T\}$. Let M be a maximum matching of B and A' (resp. U') denote the vertices of $A_\mathcal{X}$ (resp. U) covered by M. Define $\overline{A'} = A_\mathcal{X} \setminus A'$ and $\overline{U'} = U \setminus U'$.

We now prove that $(V_{\mathcal{X}} \cup U', k)$ is a linear kernel of (T, k). Let $\mathcal{C}$ be a maximum sized triangle packing that minimizes the number of vertices of $\overline{U'}$ belonging to a triangle of $\mathcal{C}$. By previous remarks, we can partition $\mathcal{C}$ into $C_{\mathcal{X}} \cup F$ where $C_{\mathcal{X}}$ are the triangles of $\mathcal{C}$ included in $T[V_{\mathcal{X}}]$ and F are the triangles of $\mathcal{C}$ containing one vertex of U and two vertices of $V_{\mathcal{X}}$. It is clear that F corresponds to a union of vertex-disjoint stars of B with centres in U. Denote by $U[F]$ the vertices of U clause gadget g to a triangle of F. If $U[F] \subseteq U'$ then $(V_{\mathcal{X}} \cup U', k)$ is immediately a kernel. Suppose there exists a vertex x_0 such that $x_0 \in U[F] \cap \overline{U'}$.

We will build a tree rooted in x_0 with edges alternating between F and M. For this let $H_0 = \{x_0\}$ and construct recursively the sets H_{i+1} such that

$$H_{i+1} = \begin{cases} N_F(H_i) \text{ if } i \text{ is even,} \\ N_M(H_i) \text{ if } i \text{ is odd,} \end{cases}$$

where, given a subset $S \subseteq U$, $N_F(S) = \{a \in A_{\mathcal{X}} : \exists s \in S \text{ s.t. } (t(a), h(a), s) \in F \text{ and } as \notin M\}$ and given a subset $S \subseteq A_{\mathcal{X}}$, $N_M(S) = \{u \in U : \exists a \in A_{\mathcal{X}} \text{ s.t. } au \in M\}$. Notice that $H_i \subseteq U$ when i is even and that $H_i \subseteq A_{\mathcal{X}}$ when i is odd, and that all the H_i are distinct as F is a union of disjoint stars and M a matching in B. Moreover, for $i \geq 1$ we call T_i the set of edges between H_i and H_{i-1}. Now we define the tree T such that $V(T) = \bigcup_i H_i$ and $E(T) = \bigcup_i T_i$. As T_i is a matching (if i is even) or a union of vertex-disjoint stars with centres in H_{i-1} (if i is odd), it is clear that T is a tree.

For i being odd, every vertex of H_i is incident to an edge of M otherwise B would contain an augmenting path for M, a contradiction. So every leaf of T is in U and incident to an edge of M in T and T contains as many edges of M than edges of F. Now for every arc $a \in A_{\mathcal{X}} \cap V(T)$ we replace the triangle of $\mathcal{C}$ containing a and corresponding to an edge of F by the triangle $(t(a), h(a), u)$ where $au \in M$ (and au is an edge of T). This operation leads to another collection of arc-disjoint triangles with the same size as $\mathcal{C}$ but containing a strictly smaller number of vertices in $\overline{U'}$, yielding a contradiction.

Finally $V_{\mathcal{X}} \cup U'$ can be computed in polynomial time and we have $|V_{\mathcal{X}} \cup U'| \leq |V_{\mathcal{X}}| + |M| \leq 2|V_{\mathcal{X}}| \leq 6k$, which proves that the kernel has $\mathcal{O}(k)$ vertices. ◀

6 Concluding Remarks

In this work, we studied the classical and parameterized complexity of packing arc-disjoint cycles and triangles in tournaments. We showed NP-hardness, fixed-parameter tractability and linear kernelization results. An interesting problem could be to find subclasses of tournaments where these problems are polynomial-time solvable. For instance, we show in the full version of the paper that it is the case for sparse tournaments, that is for tournaments which admit an FAS that is a matching. This class of tournaments is worthy of attention for these packing problems as packing vertex-disjoint triangles (and hence cycles) in sparse tournaments is NP-complete [8]. To conclude, observe that very few problems on tournaments are known to admit an $\mathcal{O}^\star(2^{\sqrt{k}})$-time algorithm when parameterized by the standard parameter k [40] - FAST is one of them [4, 23]. To the best of our knowledge, outside bidimensionality theory, there are no packing problems that are known to admit such subexponential algorithms. In light of the $2^{o(\sqrt{k})}$ lower bound shown for ACT and ATT, it would be interesting to explore if these problems admit $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ algorithms.

References

1 F. N. Abu-Khzam. An Improved Kernelization Algorithm for r-Set Packing. *Inf. Process. Lett.*, 110(16):621–624, 2010.

2 I. Akaria and R. Yuster. Packing Edge-Disjoint Triangles in Regular and Almost Regular Tournaments. *Discrete Math.*, 338(2):217–228, 2015.

3 N. Alon. Ranking Tournaments. *SIAM J. Discrete Math.*, 20(1):137–142, 2006.

4 N. Alon, D. Lokshtanov, and S. Saurabh. Fast FAST. In *36th International Colloquium on Automata, Languages, and Programming (ICALP) Part I*, pages 49–58, 2009.

5 N. Alon, R. Yuster, and U. Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995.

6 J. Bang-Jensen and G. Gutin. Paths, Trees and Cycles in Tournaments. *Congressus Numerantium*, 115:131–170, 1996.

7 J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag London, 2009.

8 S. Bessy, M. Bougeret, and J. Thiebaut. Triangle Packing in (Sparse) Tournaments: Approximation and Kernelization. In *25th Annual European Symp. on Algorithms (ESA 2017)*, volume 87, pages 14:1–14:13, 2017.

9 S. Bessy, F. V. Fomin, S. Gaspers, C. Paul, A. Perez, S. Saurabh, and S. Thomassé. Kernels for Feedback Arc Set in Tournaments. *J. Comput. Syst. Sci*, 77(6):1071–1078, 2011.

10 H. L. Bodlaender. On Disjoint Cycles. *Int. J. Found. Comput. S.*, 5(1):59–68, 1994.

11 H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel Bounds for Disjoint Cycles and Disjoint Paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011.

12 A. Caprara, A. Panconesi, and R. Rizzi. Packing Cycles in Undirected Graphs. *J. Algorithms*, 48(1):239–256, 2003.

13 P. Charbit, S. Thomassé, and A. Yeo. The Minimum Feedback Arc Set Problem is NP-hard for Tournaments. *Comb Probab Comput.*, 16(1):1–4, 2007.

14 M. Chudnovsky, P. Seymour, and B. Sullivan. Cycles in Dense Digraphs. *Combinatorica*, 28(1):1–18, 2008.

15 W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to Order Things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

16 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

17 Jean-Charles de Borda. Mémoire sur les élections au scrutin. *Histoire de l'Académie Royale des Sciences*, 1781.

18 D. Dorninger. Hamiltonian Circuits Determining the Order of Chromosomes. *Discrete Appl. Math.*, 50(2):159–168, 1994.

19 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer-Verlag London, 2013.

20 C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank Aggregation Methods for the Web. In *10th International World Wide Web Conference*, pages 613–622, 2001.

21 P. Erdős and L. Pósa. On Independent Circuits Contained in a Graph. *Canadian J. Math.*, 17:347–352, 1965.

22 S. Even, A. Itai, and A. Shamir. On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

23 U. Feige. Faster FAST(Feedback Arc Set in Tournaments), 2009. `arXiv:0911.5094`.

24 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

25 F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Fast Local Search Algorithm for Weighted Feedback Arc Set in Tournaments. In *24th AAAI Conf. on Artificial Intelligence*, pages 65–70, 2010.

26 F. V. Fomin, D. Lokshtanov, S. Saurabh, and M. Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

27 S. Fortune, J. Hopcroft, and J. Wyllie. The Directed Subgraph Homeomorphism Problem. *Theor. Comput. Sci.*, 10(2):111–121, 1980.

28 R. B. Gardner. Optimal Packings and Coverings of the Complete Directed Graph with 3-Circuits and with Transitive Triples. In *28th Southeastern International Conference on Combinatorics, Graph Theory and Computing*, volume 127, pages 161–170, 1997.

29 E. Hemaspaandra, H. Spakowski, and J. Vogel. The Complexity of Kemeny Elections. *Theor. Comput. Sci.*, 349(3):382–391, 2005.

30 R. Impagliazzo, R. Paturi, and F. Zane. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci*, 63(4):512–530, 2001.

31 M. Karpinski and W. Schudy. Faster Algorithms for Feedback Arc Set Tournament, Kemeny Rank Aggregation and Betweenness Tournament. In *21st International Symp. on Algorithms and Computation (ISAAC)*, pages 3–14, 2010.

32 T. P. Kirkman. On a Problem in Combinations. *Cambridge and Dublin Mathematical Journal*, 2:191–204, 1847.

33 M. Krivelevich, Z. Nutov, M. R. Salavatipour, J. V. Yuster, and R. Yuster. Approximation Algorithms and Hardness Results for Cycle Packing Problems. *ACM Transactions on Algorithms*, 3(4), 2007.

34 M. Krivelevich, Z. Nutov, and R. Yuster. Approximation Algorithms for Cycle Packing Problems. In *16th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 556–561, 2005.

35 T. Le, D. Lokshtanov, S. Saurabh, S. Thomassé, and M. Zehavi. Subquadratic Kernels for Implicit 3-Hitting Set and 3-Set Packing Problems. In *29th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 331–342, 2018.

36 D. Lokshtanov, A. Mouawad, S. Saurabh, and M. Zehavi. Packing Cycles Faster Than Erdős-Pósa. In *44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 71:1–71:15, 2017.

37 M. Mahajan and V. Raman. Parameterizing Above Guaranteed Values: MaxSat and MaxCut. *J. Algorithms*, 31(2):335–354, 1999.

38 J.W. Moon. *Topics on Tournaments.* Holt, Rinehart and Winston, New York, 1968.

39 C. H. Papadimitriou. *Computational Complexity.* John Wiley and Sons Ltd., 2003.

40 M. Pilipczuk. *Tournaments and Optimality: New Results in Parameterized Complexity.* PhD thesis, The University of Bergen, 2013.

41 J. P. Schmidt and A. Siegel. The Spatial Complexity of Oblivious k-Probe Hash Functions. *SIAM J. Comput.*, 19(5):775–786, 1990.

42 A. Slivkins. Parameterized Tractability of Edge-Disjoint Paths on Directed Acyclic Graphs. *SIAM J. Discrete Math.*, 24(1):146–157, 2010.

43 C. A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Appl. Math.*, 8(1):85–89, 1984.

44 R. Yuster. Packing Triangles in Regular Tournaments. *J. of Graph Theory*, 74(1):58–66, 2013.

A Sub-Exponential FPT Algorithm and a Polynomial Kernel for Minimum Directed Bisection on Semicomplete Digraphs

Jayakrishnan Madathil
The Institute of Mathematical Sciences, HBNI, Chennai, India
jayakrishnanm@imsc.res.in

Roohani Sharma
The Institute of Mathematical Sciences, HBNI, Chennai, India
roohani@imsc.res.in

Meirav Zehavi
Ben-Gurion University, Beer-Sheva, Israel
meiravze@bgu.ac.il

Abstract

Given an n-vertex digraph D and a non-negative integer k, the Minimum Directed Bisection problem asks if the vertices of D can be partitioned into two parts, say L and R, such that $|L|$ and $|R|$ differ by at most 1 and the number of arcs from R to L is at most k. This problem, in general, is W-hard as it is known to be NP-hard even when $k = 0$. We investigate the parameterized complexity of this problem on semicomplete digraphs. We show that Minimum Directed Bisection on semicomplete digraphs is one of a handful of problems that admit sub-exponential time fixed-parameter tractable algorithms. That is, we show that the problem admits a $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$ time algorithm on semicomplete digraphs. We also show that Minimum Directed Bisection admits a polynomial kernel on semicomplete digraphs. To design the kernel, we use (n, k, k^2)-splitters. To the best of our knowledge, this is the first time such pseudorandom objects have been used in the design of kernels. We believe that the framework of designing kernels using splitters could be applied to more problems that admit sub-exponential time algorithms via chromatic coding. To complement the above mentioned results, we prove that Minimum Directed Bisection is NP-hard on semicomplete digraphs, but polynomial time solvable on tournaments.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases bisection, semicomplete digraph, tournament, fpt algorithm, chromatic coding, polynomial kernel, splitters

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.28

Funding *Meirav Zehavi*: Supported by ISF grant no. 1176/18.

Acknowledgements We thank Daniel Lokshtanov and Saket Saurabh for insightful discussions on bisection in semicomplete digraphs.

1 Introduction

A bisection of a graph is a partition of its vertex set into two (almost) equal parts. In the Minimum Bisection problem, given an undirected graph G and a non-negative integer k, the task to check whether the vertex set of G can be partitioned into two parts, say A and B, such that $||A| - |B|| \leq 1$ and the number of edges with one endpoint in A and the other endpoint in B is at most k. This problem has been studied extensively, resulting in a large volume of literature [7, 11, 12, 19, 20, 27, 28]. In particular, it has been shown that Minimum Bisection is NP-hard [19], but admits a fixed-parameter tractable algorithm (with k as the parameter) [11]. We study the directed counterpart of this problem in the

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 28; pp. 28:1–28:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

framework of parameterized complexity. In the Minimum Directed Bisection problem, the input consists of a digraph D and an integer k, and the question is to determine whether the vertices of D can be partitioned into two parts, say L and R, such that $||L| - |R|| \leq 1$, and there are at most k arcs with their tails in R and heads in L (i.e. arcs directed from R to L). The problem is formally defined below.

Minimum Directed Bisection **Parameter:** k
Input: A digraph D and a non-negative integer k.
Question: Can the vertex set of D be partitioned into two parts, say L and R, such that $||L| - |R|| \leq 1$ and there are at most k arcs (u, v) with $u \in R$ and $v \in L$?

Feige and Yahalom [16] showed that this problem is NP-hard even for $k = 0$, which implies that the Minimum Directed Bisection problem has no polynomial time approximation unless P = NP. Their result also implies that Minimum Directed Bisection is para-NP-hard on general digraphs and hence, it is W-hard when parameterized by k. We study the complexity of this problem on restricted classes of digraphs such as tournaments and semicomplete digraphs.

Our results. Our contribution is threefold.

1. We show that Minimum Directed Bisection is polynomial time solvable on tournaments.
2. On semicomplete digraphs, Minimum Directed Bisection is NP-hard, but admits an algorithm running in time $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$.
3. We also show that Minimum Directed Bisection on semicomplete digraphs admits a polynomial kernel.

A *tournament* is a digraph D such that for every distinct pair of vertices u and v of D, *exactly one* of the arcs (u, v) or (v, u) is present in D. A *semicomplete digraph* is a digraph D such that for every pair of distinct vertices u and v of D, *at least one* of the arcs (u, v) or (v, u) is present in D. Note that both the arcs (u, v) and (v, u) may be present in a semicomplete digraph. Tournaments and semicomplete digraphs form two interesting and well-studied classes of digraphs. See, for example, the monograph by Bang-Jensen and Gutin [3] for an overview. In recent years, tournaments and semicomplete digraphs received a great deal of attention, in part, due to the many structural results for these classes that are similar to the theory of minors for undirected graphs. In particular, it has been shown that tournaments are well-quasi ordered under strong minors [23] and strong immersions [9]. Similarly, semicomplete digraphs have been shown to be well-quasi ordered under strong minors, butterfly immersions [22] and strong immersions [5].

Due to the extremely rigid structure of tournaments, a number of problems that are NP-hard on general digraphs become polynomial time solvable on tournaments. Examples include classic problems such as Hamiltonian path [29] and Hamiltonian Cycle [8]. Although semicomplete digraphs are a slight generalization of tournaments, the flexibility in the definition allowing for the presence of anti-parallel arcs makes them distinctly dissimilar to tournaments when it comes to tractability of algorithmic problems. This contrast in behavior is perhaps best illustrated by problems such as Cutwidth and Optimal Linear Arrangement (OLA), which are polynomial time solvable on tournaments, but NP-hard on semicomplete digraphs [4]. Our work reinforces this pattern. We show that while Minimum Directed Bisection is polynomial time solvable on tournaments (Lemma 4), it is NP-hard on semicomplete digraphs (Lemma 16). The polynomial time solvability of Minimum Directed Bisection on tournaments follows from a polynomial time algorithm due to Fradkin [18] for computing the cutwidth of a tournament. Barbero et al. [4] showed that Fradkin's approach in [18] yields a polynomial time algorithm for the Optimal Linear Arrangement problem on tournaments as well. We rely on the analysis of Barbero et al. in [4] to conclude that Fradkin's approach works for Minimum Directed Bisection on tournaments as well. We establish the NP-hardness of Minimum Directed Bisection on semicomplete digraphs by a reduction from the Maximum Bisection problem on directed acyclic graphs.

Tournaments and semicomplete digraphs found favor with the parameterized complexity community mainly because of the possibility that several problems might admit sub-exponential time parameterized algorithms on these classes of digraphs. In the field of parameterized complexity, problems that admit sub-exponential time algorithms (except those on planar graphs, using the bidimensionality approach) are something of a rarity. In fact, the first such sub-exponential time algorithm was designed by Alon, Lokshtanov and Saurabh [1] for the Feedback Arc Set problem on tournaments (FAST). They developed the technique of chromatic coding to design a $2^{\mathcal{O}(\sqrt{k}\log^2 k)}$ algorithm for this problem. This running time has since been improved, independently by Feige [15], Karpinski and Schudy [21], and Fomin and Pilipczuk [17], to $2^{\mathcal{O}(\sqrt{k})}$. Fomin and Pilipczuk's [17] algorithm for FAST was in fact a by-product of their work on the Cutwidth and Optimal Linear Arrangement (OLA) problems on semicomplete digraphs. They showed that both Cutwidth and OLA admit sub-exponential FPT algorithms on semicomplete digraphs. Later, Barbero et al. [4] showed that these problems are indeed NP-hard on semicomplete digraphs. We show that Minimum Directed Bisection on semicomplete digraphs admits a sub-exponential time algorithm that runs in time $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$ (Theorem 21).

Both our algorithm and kernel for Minimum Directed Bisection build on a crucial observation that in any bisection (L, R) of a semicomplete digraph, every vertex v has a "preferred position" (either L or R), which we call the "canonical position" of v (denoted by $\mathtt{can}(v)$). In particular, if (L, R) is a bisection of size at most k, then we show that in (L, R), no more than $2k$ vertices can deviate from their canonical positions. Note that with just this information one can easily design an $n^{\mathcal{O}(k)}$ algorithm for the problem. In order to achieve sub-exponential FPT running time, we use the technique of chromatic coding to identify a small set of vertices that contain these $2k$ vertices that can deviate from their canonical positions. More precisely, we color the vertex set with $\mathcal{O}(\sqrt{k})$ colors such that all the k arcs in a k-sized bisection (if it exists) are properly colored. Then, to identify the desired set of vertices, we exploit the fact that no arc within a color class can be directed from R to L. The crucial point in chromatic coding is that it is sufficient to try out $2^{\mathcal{O}(\sqrt{k}\log k)}\log n$ different coloring functions so as to ensure that we find a coloring with the required property.

The kernelization complexity of problems on tournaments and semicomplete digraphs has proved to be even more diverse, requiring a wide array of techniques specific to individual problems. For instance, it is now textbook knowledge that the Feedback Arc Set problem on tournaments admits a simple kernel with $\mathcal{O}(k^2)$ vertices [10, 14]. There has been a line of work on this front that improved the size of the kernel, the current best being a linear kernel by Bessy et al. [6]. Barbero et al. [4] showed that while Cutwidth on semicomplete digraphs does not admit a polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP}/\text{poly}$, OLA does admit a kernel with $2k$ vertices. In addition, by using the submodularity property of directed cuts, Barbero et al. [4] showed that Cutwidth admits a Turing kernel on semicomplete digraphs.

We employ a family of hash functions called (n, k, k^2)-splitters, introduced by Naor et al. [26], to design a polynomial kernel for Minimum Directed Bisection on semicomplete digraphs (Theorem 22). Splitters are a well-known combinatorial tool often used to derandomize algorithms. To the best of our knowledge, this is the first occurrence of splitters' being used to design polynomial kernels. Our kernelization algorithm for Minimum Directed Bisection on semicomplete digraphs has two main steps. In the first step, we reduce the given instance (D, k) of the Minimum Directed Bisection problem to $k^{\mathcal{O}(1)}$ many instances of a weighted variant of the problem, with the guarantee that the sizes of the reduced instances is $k^{\mathcal{O}(1)}$ and at least one of the reduced instances is equivalent to the original instance (D, k). Then, in the second step, we apply a polynomial time reduction from the reduced instances to Minimum Directed Bisection to obtain the desired kernel. To elaborate a little about the first step, like in the design of our algorithm, we want a coloring function that properly colors the k arcs of the bisection. But unlike in the algorithm where we have to use $\mathcal{O}(\sqrt{k})$ colors to ensure sub-exponential runtime, for the kernel, it is enough to have such a coloring using $\mathcal{O}(k^2)$ colors. This increase in the number of colors from $\mathcal{O}(\sqrt{k})$ to $\mathcal{O}(k^2)$ allows us to bring down the size of the coloring family that guarantees the existence of the good coloring that we want, to a polynomial in k. This is where we use (n, k, k^2)-splitters: we construct a reduced instance for each of the $k^{\mathcal{O}(1)}$ coloring functions. The one corresponding to the good coloring function is the one that is always a yes-instance if the input instance is a yes-instance.

Related work on Minimum Bisection. As mentioned earlier, the Minimum Bisection problem, (the undirected counterpart of our problem) is an extensively-studied problem in algorithmic graph theory. Being a natural variant of the well-known Min Cut problem, Minimum Bisection has already been known to be NP-hard since the 1970s [19]. However, the parameterized complexity of the problem had been open for a long time, until it was finally settled by Cygan et al. [11] in 2014, who showed that the problem is fixed-parameter tractable. Before that, Jansen et al. [20] had shown that the problem is fixed-parameter tractable when parameterized by the treewidth of the input graph. The best known approximation algorithm for Minimum Bisection, to the best of our knowledge, is due to Räcke [28], with an approximation factor of $\mathcal{O}(\log n)$. While Minimum Bisection on planar graphs admits a single-exponential parameterized algorithm [7], it remains open whether the problem is NP-hard. On the related (but incomparable) class of unit disk graphs, Minimum Bisection is known to be NP-hard [12], but was recently shown to admit a single-exponential parameterized algorithm [27]. The Minimum Vertex Bisection problem (the vertex-deletion variant of Minimum Bisection) asks the following question. Given an undirected graph G and a non-negative integer k, is it possible to partition the vertex set of G into three parts A, S and B such that $|S| \leq k$, $|A| = |B|$ and there are no edges between A and B? This problem is W[1]-hard: as observed in [11], the W[1]-hardness of Minimum Vertex Bisection follows from a more general result due to Marx [25].

2 Preliminaries

For a natural number $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. For $z \in \mathbb{N}$, $|z|$ denotes the number of bits in the binary representation of z.

Splitters. Consider two finite sets A and B and a function $f : A \to B$. For a subset $S \subseteq A$, we say that f *splits S evenly* if for every $b_1, b_2 \in B$, $|f^{-1}(b_1) \cap S|$ and $|f^{-1}(b_2) \cap S|$ differ by at most 1. Notice that if $|B| > |S|$, then f splits S evenly if and only if f is injective on S.

▶ **Definition 1** ([26]). *For positive integers n, k and ℓ, an (n, k, ℓ)-*splitter *is a family $\mathcal{F}$ of functions from $[n]$ to $[\ell]$ such that for every subset $S \subseteq [n]$ of size at most k, there exists a function $f \in \mathcal{F}$ that splits S evenly.*

▶ **Proposition 2** ([2, 26]). *There is an algorithm that, given positive integers n and k, runs in time $k^{\mathcal{O}(1)} n \log n$ and constructs an (n, k, k^2)-splitter of size $k^{\mathcal{O}(1)} \log n$.*

Digraphs. For a digraph D, we denote by $V(D)$ and $A(D)$ the vertex set and arc set of D, respectively. For sets $X, Y \subseteq V(D)$, $A(X, Y) = \{(x, y) \in A(D) \mid x \in X \text{ and } y \in Y\}$. (Whenever dealing with multiple digraphs on the same vertex set, we may write $A_D(X, Y)$ instead of $A(X, Y)$ to emphasize that we mean the set of arcs from X to Y *in* D.) For a vertex $v \in V(D)$, $A(v)$ denotes the set of arcs incident with v. For an arc $a \in A(D)$, $V(a)$ denotes the set of endpoints of a. That is, if $a = (u, v) \in A(D)$, then $V(a) = \{u, v\}$. More generally, for a set of arcs $A \subseteq A(D)$, $V(A) = \cup_{a \in A} V(a)$. The *complement* of D, denoted by $\overline{D}$, is the digraph defined as follows: $V(\overline{D}) = V(D)$ and $A(\overline{D}) = (V^2 \setminus \{(u, u) \mid u \in V(D)\}) \setminus A(D)$. (We assume that D has no self-loops.) Note that for vertices $u, v \in V(D)$, if $(u, v) \notin A(D)$, then $(u, v), (v, u) \in A(\overline{D})$. For a digraph D, $\texttt{rev}(D)$ denotes the digraph obtained by reversing all arcs of D, i.e. $V(\texttt{rev}(D)) = V(D)$ and $A(\texttt{rev}(D)) = \{(u, v) \mid (v, u) \in A(D)\}$.

Consider a digraph D. Throughout this paper, we use (L_D, R_D) to denote a bipartition of D. That is $V(D) = L_D \cup R_D$ and $L_D \cap R_D = \emptyset$. We may drop the subscript D and simply write (L, R) when D is clear from the context. A bipartition (L_D, R_D) is a *bisection* of D if $||L_D| - |R_D|| \leq 1$; we refer to the value $|A(R_D, L_D)|$ as the *size of the bipartition* (L_D, R_D). Moreover, we say that (L_D, R_D) is a *one way partition* if $A(R_D, L_D) = \emptyset$, i.e. if (L_D, R_D) is a bipartition of size 0.

3 Some Observations and Simple Lemmas

Throughout this paper, whenever dealing with an instance (D, k) of Minimum Directed Bisection, we assume that $|V(D)|$ is even. This assumption about $V(D)$ is made without loss of generality since if $|V(D)|$ is odd, then we may construct two instances (D_1, k) and (D_2, k) of Minimum Directed Bisection such that at least one of them will be equivalent to (D, k) and $|V(D_1)| = |V(D_2)| = |V(D)| + 1$. The following lemma shows that it is safe to make this assumption.

▶ **Lemma 3** ($\star$[1]). *Given an instance (D, k) of* Minimum Directed Bisection *where $|V(D)|$ is odd, it is possible to construct two instances (D_1, k) and (D_2, k) of* Minimum Directed Bisection *in polynomial time such that* (i) *$|V(D_1)|$ and $|V(D_2)|$ are even,* (ii) *D_1 and D_2 are semicomplete digraphs (tournaments) if D is a semicomplete digraph (tournament), and* (iii) *(D, k) is a yes-instance if and only if either (D_1, k) or (D_2, k) is a yes-instance.*

The following lemma deals with Minimum Directed Bisection on tournaments.

▶ **Lemma 4** ($\star$). Minimum Directed Bisection *is polynomial time solvable on tournaments.*

Proof Sketch. The algorithm for Minimum Directed Bisection on tournaments works as follows. Given a tournament T on n vertices, sort the vertices in non-decreasing order by their in-degrees. Take L_T to be the set of the first $n/2$ vertices in the sorted list, and $R_T = V(T) \setminus L_T$. Then, our claim is that (L_T, R_T) is a minimum bisection of T, i.e. $|A(R_T, L_T)| \leq |A(Y, X)|$ for every bisection (X, Y) of T. The correctness of this algorithm follows from Lemma 1 in [4]. ◀

[1] Due to paucity of space, proofs of statements marked with a $\star$ have been omitted.

We now state two observations about semicomplete digraphs. These observations are considered to be folklore, so we omit their proofs. Specifically, the first observation is immediate, and the second can be found in [13].

▶ **Observation 5.** *Let D be a semicomplete digraph on n vertices, where n is even. For every $v \in V(D)$, we have $d^+(v) \geq n/2$ or $d^-(v) \geq n/2$.*

▶ **Observation 6.** *Every semicomplete digraph contains a Hamiltonian path. Moreover, there is a polynomial time algorithm that, given a semicomplete digraph D as input, finds a Hamiltonian path in D.*

We now develop some preliminary results that will be used in the design and analysis of our algorithm and kernel.

Canonical Position. A crucial idea behind our algorithm and kernel is that every vertex has a "preferred position" in a bisection (either L or R). In particular, if a vertex v deviates from its preferred position, then at least one of the arcs incident with v would have to belong to $A(R, L)$. Therefore, in a bisection of size at most k, no more than $\mathcal{O}(k)$ vertices can deviate from their preferred positions. Next, we formalize this idea.

▶ **Definition 7.** *Let D be a semicomplete digraph on n vertices, where n is even. The partition (L^c_D, R^c_D) (not necessarily a bisection) of $V(D)$ defined as $L^c_D = \{v \in V(D) \mid d^+(v) \geq n/2\}$ and $R^c_D = V(D) \setminus L^c_D$ is called the* canonical partition *of D. For a vertex $v \in L^c_D$, we say that L^c_D is the* canonical position of v *and write* $\texttt{can}(v) = L^c_D$. *Similarly, for a vertex $v \in R^c_D$, we say that R^c_D is the* canonical position of v *and write* $\texttt{can}(v) = R^c_D$.

▶ **Definition 8.** *Let D be a semicomplete digraph with a bipartition (L, R). For a vertex $v \in V(D)$, we say that (L, R)* respects the canonical position of v *if $v \in L$ and* $\texttt{can}(v) = L^c_D$ *or if $v \in R$ and* $\texttt{can}(c) = R^c_D$. *Otherwise, we say that (L, R)* violates the canonical position of v.

The following lemma follows directly from the definition of the canonical position of a vertex. It shows that if a bisection (L, R) violates the canonical position of a vertex v, then at least one of the arcs incident with v must belong to $A(R, L)$.

▶ **Lemma 9** ($\star$). *Consider a semicomplete digraph D on n vertices, where n is even. Let (L, R) be a bisection of D. Let $v \in V(D)$ be such that (L, R) violates the canonical position of v. Then, $A(R, L) \cap A(v) \neq \emptyset$.*

An immediate consequence of Lemma 9 is that any bisection of size at most k of a semicomplete digraph can violate the canonical positions of at most $2k$ vertices.

▶ **Corollary 10.** *Let (D, k) be an yes-instance of* Minimum Directed Bisection, *and let (L, R) be a bisection of D of size at most k. Let $X = \{v \in V(D) \mid (L, R)$ violates the canonical position of $v\}$. Then, $|X| \leq 2k$.*

Proof. As Lemma 9 shows, for every $v \in X$, $A(R, L)$ contains an arc incident with v. Since each arc in $A(R, L)$ can be incident with at most two vertices in X, and $|A(R, L)| \leq k$, it follows that $|X| \leq 2k$. ◀

Consider a bisection (L, R) of D, and a subset of vertices $Z \subseteq V(D)$ such that no arc of $D[Z]$ belongs to $A(R, L)$. The following lemma identifies the vertices in Z whose canonical positions can possibly be violated by (L, R). Moreover, it shows that there can be at most $4k$ such vertices in Z, and that (L, R) must respect the canonical position of every other vertex.

▶ **Lemma 11.** *Let D be a semicomplete digraph with a bisection (L, R) of size at most k. Let $Z \subseteq V(D)$ be such that no arc of $D[Z]$ belongs to $A_D(R, L)$. That is, $(L \cap Z, R \cap Z)$ is a one way partition of Z. Let P be a Hamiltonian path in $D[Z]$ from a vertex α to a vertex β for some $\alpha, \beta \in Z$. Let $X = \{x \in Z \mid \texttt{can}(x) = R^c_D\}$ and $Y = \{y \in Z \mid \texttt{can}(y) = L^c_D\}$. For $1 \leq j \leq |X|$, let x_j be the j^{th} vertex of P that belongs to X as we traverse P from α to β, and for $1 \leq p \leq |Y|$, let y_p be the p^{th} vertex of P that belongs to Y as we traverse $\texttt{rev}(P^i)$ from β to α. Then, if $|X| > 2k$, then for every $x_j \in X$ with $j > 2k$, $x_j \in R$. Similarly, if $|Y| > 2k$, then for every $y_p \in Y$ with $p > 2k$, $y_p \in L$. (In other words, (L, R) respects the canonical positions of x_j and y_p for $j > 2k$ and $p > 2k$.)*

Proof. Note first that for any arc $(z, z') \in A(P)$, it cannot be the case that $z \in R$ and $z' \in L$, as $(L \cap Z, R \cap Z)$ is a one way partition of Z. More generally, this property holds for any two distinct vertices $z, z' \in Z$ such that z appears before z' as we traverse P from α to β. To see this, suppose by way of contradiction that $z \in R$ and $z' \in L$. Then, consider the subpath of P from z to z'. This subpath must have an arc from a vertex in R to a vertex in L, which we have already argued to be impossible.

Now, suppose that $|X| > 2k$ and consider a vertex $x_j \in X$ with $j > 2k$. Then, $\texttt{can}(x_j) = R^c_D$. We need to show that $x_j \in R$. Assume for contradiction that $x_j \in L$, i.e. (L, R) violates the canonical position of x_j. Then, for every $j' \leq j$, we must have $x_{j'} \in L$ as well, as otherwise, we obtain a pair $\{x_{j'}, x_j\}$ such that $x_{j'}$ appears before x_j as we traverse P from α to β while $x_{j'} \in R$ and $x_j \in L$, which by the preceding argument, is not possible. Having $x_{j'} \in L$ for every $j' \leq j$ means that (L, R) violates the canonical position of $x_{j'}$ for every $j' \leq j$. Thus, (L, R) violates the canonical positions of more than $2k$ vertices. However, as (L, R) is a bisection of D of size at most k, by Corollary 10, (L, R) can violate the canonical positions of at most $2k$ vertices.

Using symmetric arguments, we also derive that $y_p \in L$ for every $p > 2k$. ◀

▶ Remark 12. In the above proof, it is actually shown that for any $j \leq |X|$ such that (L, R) violates the canonical position of $x_j \in X$, (L, R) must violate the canonical position of $x_{j'}$ for every $j' < j$ as well. Similarly, for any $p \leq |Y|$ such that (L, R) violates the canonical position of $y_p \in Y$, (L, R) must violate the canonical position of $y_{p'}$ for every $p' < p$ as well.

Tools for Our Kernel. We now develop some tools that will be used to design our kernel. Consider a vertex-weighted digraph D, where the weights are given by a function $w : V(D) \to \mathbb{N}$. For a subset $X \subseteq V(D)$ of vertices, the weight of X is defined as $w(X) = \sum_{x \in X} w(x)$. We say that a partition (L, R) of $V(D)$ is a *w-bisection* of D (or simply a *bisection* when w is clear from the context) if $|w(L) - w(R)| \leq 1$, and $|A(R, L)|$ is called the size of the bisection. So, given (D, w, k), the Weighted Minimum Bisection (WMB) problem asks whether D has a w-bisection of size at most k. We now define a "composition" of this problem, where the input consists of multiple pairs (D, w), and at least one is required to have a bisection of size at most k.

At Least One Weighted Minimum Bisection (ALO-WMB)
Input: A collection $D_1, D_2, \ldots, D_r$ of vertex-weighted semicomplete digraphs, a weight function $w_i : V(D_i) \to \mathbb{N}$ for every $i \in [r]$, and a non-negative integer k.
Question: Does there exist $i \in [r]$ such that D_i has a w_i-bisection of size at most k?

It is not difficult to see that this problem belongs to the class NP: a polynomial-sized certificate for the problem would be a partition (L, R) of $V(D_i)$ for some i, and it can be verified whether (L, R) is indeed a bisection of size at most k in time polynomial in $\sum_{j \in [r]} (|V(D_j)| + \sum_{v \in V(D_j)} |w_j(v)|)$. For future reference, we record this observation below.

▶ **Observation 13.** ALO-WMB *belongs to the class NP.*

$\mathcal{P}$-Contraction of a Digraph. Consider a digraph D. Let $\mathcal{P} = \{V_1, V_2, \ldots, V_p\}$ be a partition of $V(D)$ (i.e. $V_i \cap V_j = \emptyset$ for every distinct $i, j \in [p]$ and $\bigcup_{i \in [p]} V_i = V(D)$). The *$\mathcal{P}$-contraction* of D is the digraph $\mathcal{P}(D)$ obtained by "contracting" every part V_i into a single vertex, formally defined as follows. The digraph $\mathcal{P}(D)$ has p vertices, one corresponding to every part $V_i \in \mathcal{P}$. For each $i \in [p]$, let z_i deonte the vertex of $\mathcal{P}(D)$ corresponding to the part V_i. For distinct $i, j \in [p]$, the arc (z_i, z_j) is present in $\mathcal{P}(D)$ if and only if there exist $v_i \in V_i$ and $v_j \in V_j$ such that $(v_i, v_j) \in A(D)$. (Note that $\mathcal{P}(D)$ has no self-loops.) For a vertex $z_i \in V(\mathcal{P}(D))$, we refer to the part V_i as the *parent-part of z_i* (or simply *part of z_i*), and write $\texttt{part}(z_i) = V_i$. Moreover, we define the *$\mathcal{P}$-weighted-contraction* of D to be the digraph $\mathcal{P}(D)$, where weights are assigned to its vertices by a weight function $w_{\mathcal{P}}$, which is given by $w_{\mathcal{P}}(z_i) = |V_i|$ for every vertex z_i of $\mathcal{P}(D)$.

▶ **Lemma 14** (⋆). *Let D be an unweighted (not necessarily semicomplete) digraph and (L, R) a bipartition (not necessarily a bisection) of D. Consider a pair of partitions $\mathcal{L} = \{V_1, V_2, \ldots, V_p\}$ and $\mathcal{R} = \{V_{p+1}, V_{p+2}, \ldots, V_q\}$ of L and R, respectively. Let $\mathcal{P}$ be the partition of $V(D)$ that is the union of $\mathcal{L}$ and $\mathcal{R}$, i.e. $\mathcal{P} = \mathcal{L} \cup \mathcal{R}$. Let $H = \mathcal{P}(D)$ be the $\mathcal{P}$-weighted-contraction of D. (Thus, for each $V_i \in \mathcal{P}$, H has a vertex z_i, with weight $w_{\mathcal{P}}(z_i) = |V_i|$.) Let $L_H = \{z_1, z_2, \ldots, z_p\}$, and $R_H = \{z_{p+1}, z_{p+2}, \ldots, z_q\}$. Then, (L, R) is a bisection of D if and only if (L_H, R_H) is a $w_{\mathcal{P}}$-bisection of H.*

4 NP-hardness of Minimum Directed Bisection on Semicomplete Digraphs

In this section, we show that Minimum Directed Bisection is NP-hard on semicomplete digraphs by a reduction from Maximum Directed Bisection on directed acyclic graphs (DAGs). (Recall that in the Maximum Directed Bisection problem, the input consists of a digraph D and an integer k, and the task is to determine whether D admits a bisection (L, R) such that $|A(R, L)| \geq k$.) The NP-hardness of Maximum Directed Bisection on DAGs easily follows from the fact that Directed Max-Cut is NP-hard on DAGs, shown by Lampis et al. [24] (Theorem 4 in [24]).

▶ **Observation 15** (⋆). Maximum Directed Bisection *is NP-complete on DAGs.*

▶ **Lemma 16** (⋆). Minimum Directed Bisection *is NP-complete on semicomplete digraphs.*

Proof Sketch. It is easy to see that Minimum Directed Bisection belongs to NP. Now, for the proof of hardness, we give a reduction from the Maximum Directed Bisection problem on DAGs. To this end, consider an instance (D, k) of Maximum Directed Bisection where D is a DAG. Let $|V(D)| = n$. Without loss of generality, assume that n is even. (Add an isolate vertex to D if n is odd.) We construct an instance (D', k') of Minimum Directed Bisection as follows. Take $D' = \overline{D}$ and $k' = n^2/4 - k$. Note first that $\overline{D}$ is a semicomplete digraph, and that D' can be constructed in polynomial time. It can be verified that (D, k) is a yes-instance if and only if (D', k') is a yes-instance. ◀

▶ Remark 17. By Observation 13, the problem At Least One Weighted Minimum Bisection belongs to the class NP. Moreover, we have just shown that Minimum Directed Bisection is NP-complete on semicomplete digraphs. Thus, we can conclude that there is a polynomial time reduction from ALO-WMB to Minimum Directed Bisection on semicomplete digraphs.

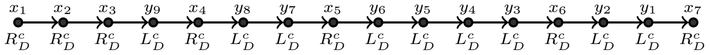


Figure 1 An example of a Hamiltonian path P_i in D_i, with the canonical positions of its vertices. We have $V(D_i) = X_i \cup Y_i$. The vertices of X_i are indexed from left to right, while the vertices of Y_i are indexed from right to left.

5 FPT Algorithm for Minimum Directed Bisection on Semicomplete Digraphs

In this section, we design an algorithm for Minimum Directed Bisection on semicomplete digraphs that runs in time $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$. Our algorithm is based on a technique called chromatic coding, introduced by Alon et al. [1]. The basic idea behind this technique is to color the vertices of the input digraph by $\mathcal{O}(\sqrt{k})$ colors so that all the k arcs in a k-sized bisection (if one exists) are properly colored, i.e. have their endpoints in different color classes. As stated below, it was shown that there is a coloring family of size $2^{\mathcal{O}(\sqrt{k}\log k)}\log n$ that does this job.

▶ **Proposition 18** (Chromatic coding [1]). *For positive integers n and k, there exists a family $\mathcal{F}$ of functions from $[n]$ to $[2\lceil\sqrt{k}\rceil]$ with the following property: for every graph G with $V(G) = [n]$ and with at most k edges, there exists a function $f \in \mathcal{F}$ that properly colors $E(G)$ (i.e. for every $uv \in E(G)$, $f(u) \neq f(v)$). Moreover, there is an algorithm that runs in time $2^{\mathcal{O}(\sqrt{k}\log k)}n\log n$ and constructs such a family $\mathcal{F}$ of size $2^{\mathcal{O}(\sqrt{k}\log k)}\log n$.*

The family $\mathcal{F}$ in Proposition 18 is called a *coloring family*, and the elements of $\mathcal{F}$ are called *coloring functions*. For a coloring function $f \in \mathcal{F}$, we refer to the set of all vertices that have been assigned the color i by f as the i^{th} color class.

Informally, our algorithm for Minimum Directed Bisection is as follows. Suppose that (D, k) is a yes-instance, and let (L, R) be a bisection of D of size at most k. We would like to color the vertices of D using $\mathcal{O}(\sqrt{k})$ colors so that all arcs of $A(R, L)$ are properly colored (of course, without knowing (L, R)). Proposition 18 guarantees such a coloring function f. For any color i, let D_i be the subgraph of D induced by vertices in the i^{th} color class. Because all arcs of $A(R, L)$ are properly colored, no arc of D_i can belong to $A(R, L)$. Thus, we "guess" the vertices of D_i whose canonical positions are violated by (L, R), and define a partition of $V(D_i)$ accordingly. Due to Lemma 11 and Remark 12, there are at most $4k$ vertices in D_i whose canonical positions can be violated by (L, R). Thus we would need to consider only $k^{\mathcal{O}(\sqrt{k})} = 2^{\mathcal{O}(\sqrt{k}\log k)}$ guesses, which gives rise to an algorithm with the claimed running time. We present a formal description of our algorithm in Algorithm 1.

In the next two lemmas, we establish the correctness of Algorithm 1, and analyse its runtime.

▶ **Lemma 19** (⋆). *Algorithm 1 is correct.*

Proof Sketch. Observe first that if Algorithm 1 returns yes, then D has a bisection of size at most k (that is (L^f_t, R^f_t) for some f and t). Therefore, in order to show that Algorithm 1 returns the correct answer, we shall prove the following. If (D, k) is a yes-instance with a k-sized bisection (L, R), then for some $f \in \mathcal{F}$ and for some tuple t defined in Step 2 of the algorithm, we will have $(L, R) = (L^f_t, R^f_t)$ where L^f_t and R^f_t are as defined in Step 2-(ii)-(a) of the algorithm. ◀

▶ **Lemma 20** (⋆). *Algorithm 1 runs in time $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$.*

Algorithm 1 Input: (D, k) where D is a semicomplete digraph on n vertices.

```
1. Run the algorithm in Proposition 18 and construct a coloring family $\mathcal{F}$.
2. For each $f \in \mathcal{F}$, perform the following steps:
      (i). For each $i \in [2\lceil\sqrt{k}\rceil]$, consider $D_i$, the subgraph of $D$ induced by $f^{-1}(i)$, and perform the following steps.
            (a). Find a Hamiltonian path $P_i$ in $D_i$. Then, $P_i$ is a path from $\alpha_i$ to $\beta_i$ for some $\alpha_i, \beta_i \in V(D_i)$.
            (b). Let $X_i = \{x \in V(D_i) \mid \texttt{can}(x) = R^{c}_{D}\}$ and $Y_i = \{y \in V(D_i) \mid \texttt{can}(y) = L^{c}_{D}\}$. For $1 \leq j \leq |X_i|$, let $x_j$ be the $j^{\text{th}}$ vertex of $P_i$ that belongs to $X_i$ as we traverse $P_i$ from $\alpha_i$ to $\beta_i$. For $1 \leq p \leq |Y_i|$, let $y_p$ be the $p$th vertex of $P_i$ that belongs to $Y_i$ as we traverse $\texttt{rev}(P_i)$ from $\beta_i$ to $\alpha_i$ (see Figure 1).
            (c). For each pair $(\ell_i, q_i)$, where $0 \leq \ell_i \leq \min\{|X_i|, 2k\}$ and $0 \leq q_i \leq \min\{|Y_i|, 2k\}$, construct a partition $(L_{\ell_i,q_i}, R_{\ell_i,q_i})$ of $V(D_i)$ as follows. For every $j \leq \ell_i$, assign $x_j$ to $L_{\ell_i,q_i}$, and for every $p \leq q_i$, assign $y_p$ to $R_{\ell_i,q_i}$. For every other vertex $z \in V(D_i)$, assign $z$ to $L_{\ell_i,q_i}$ if $\texttt{can}(z) = L^{c}_{D}$, and assign $z$ to $R_{\ell_i,q_i}$ otherwise.
      (ii). For each tuple $t = (t_1, t_2, \ldots, t_{2\lceil\sqrt{k}\rceil})$, $t_i = (\ell_i, q_i)$ for some $0 \leq \ell_i \leq \min\{|X_i|, 2k\}$ and $0 \leq q_i \leq \min\{|Y_i|, 2k\}$ such that $(L_{\ell_i,q_i} R_{\ell_i,q_i})$ is a one way partition of $V(D_i)$, perform the following step.
            (a). Construct a partition $(L^f_t, R^f_t)$ of $V(D)$ as follows: $L^f_t = \bigcup_i L_{\ell_i,q_i}$ and $R^f_t = \bigcup_i R_{\ell_i,q_i}$. If $(L^f_t, R^f_t)$ is a bisection of $D$ of size at most $k$, then return that $(D, k)$ is a yes-instance and terminate.
3. Return that $(D, k)$ is a no-instance.
```

We have thus proved the following theorem.

▶ **Theorem 21.** Minimum Directed Bisection *on semicomplete digraphs admits a* $2^{\mathcal{O}(\sqrt{k}\log k)} n^{\mathcal{O}(1)}$ *time algorithm.*

6 Polynomial Kernel for Minimum Directed Bisection on Semicomplete Digraphs

In this section, we design a polynomial kernel for the Minimum Directed Bisection problem on semicomplete digraphs. Specifically, we prove the following theorem.

▶ **Theorem 22.** *There is a polynomial time algorithm that, given an instance* (D, k) *of* Minimum Directed Bisection *where* D *is a semicomplete digraph, produces an equivalent instance* (D', k') *of* Minimum Directed Bisection *such that* $\max\{|V(D')|, k'\} \leq k^{\mathcal{O}(1)}$.

Throughout this section, D denotes a semicomplete digraph on n vertices and k a non-negative integer. Given an instance (D, k) of Minimum Directed Bisection, our kernel proceeds as follows. In the first step, we reduce the instance (D, k) to an equivalent instance of ALO-WMB (defined in Section 2), whose size is bounded by $k^{\mathcal{O}(1)}$. As ALO-WMB belongs to the class NP (see Remark 13), and Minimum Directed Bisection is NP-complete (see Lemma 16), we know that ALO-WMB admits a polynomial time reduction to Minimum Directed Bisection. In the second step, we apply this reduction to the $k^{\mathcal{O}(1)}$-sized ALO-WMB instance to obtain an equivalent instance of Minimum Directed Bisection of size $k^{\mathcal{O}(1)}$. And in order to reduce Minimum Directed Bisection to ALO-WMB in the first step, we use an $(n, 2k, 4k^2)$-splitter. Using such a splitter, we construct $k^{\mathcal{O}(1)}$ many instances of Weighted Minimum Bisection (WMB) such that each one of these instances has size $k^{\mathcal{O}(1)}$, and at least one of them is equivalent to our original instance (D, k).

▶ Remark 23 (Assumption about k). In this section, given an instance (D, k) of Minimum Directed Bisection, we assume that $\log n \leq k$. Otherwise, if $\log n > k$, then we have $n > 2^k > 2^{\sqrt{k}\log k}$. But this implies that Algorithm 1 runs in time polynomial in n. That is, if $\log n > k$, then we can solve Minimum Directed Bisection on semicomplete digraphs in polynomial time.

Terminology. In what follows, we use the following definitions and notations. Consider an instance (D, k) of Minimum Directed Bisection. Let (L, R) be a bisection of D. Let S be the set of endpoints of the arcs in $A(R, L)$, i.e. $S = V(A(R, L))$. For some positive integer $\ell \geq |S|$, let $f : V(D) \to [\ell]$ be a function that splits S evenly. For each $i \in [\ell]$ such that $f^{-1}(i) \neq \emptyset$, let $D_i, P_i, \alpha_i, \beta_i, X_i, Y_i, x_j$ and y_p be as defined in Algorithm 1. Since f splits S evenly, and $\ell \geq |S|$, the function f is injective on S. Therefore, no arc of D_i belongs to $A(R, L)$. Moreover, note that $V(D_i)$ is the disjoint union of X_i and Y_i.

Fix $i \in [\ell]$. We say that a vertex $z \in V(D_i)$ is *marked* if $z = x_j$ for some $x_j \in X_i$ with $j \leq 2k$, or if $z = y_p$ for some $y_p \in Y_i$ with $p \leq 2k$. Otherwise, we say that z is *unmarked*. That is, the first $2k$ of vertices of P_i that belong to X_i, and the last $2k$ vertices of P_i that belong to Y_i are marked. All other vertices are unmarked. (Here, first and last are defined with respect to the traversal of P_i from α_i to β_i.) Therefore, in light of Lemma 11, if (L, R) is of size at most k, and (L, R) violates the canonical position of a vertex $v \in V(D_i)$, then v is a marked vertex. In other words, every bisection of D of size at most k, if such a bisection exists, respects the canonical position of every unmarked vertex. Let M_i denote the set of all marked vertices in $V(D_i)$.

Let $n_i = |V(D_i)|$. For $r \in [n_i]$, let v_r be the r^{th} vertex of P_i as we traverse P_i from α_i to β_i. That is, $P_i = (\alpha_i =)v_1 v_2 \dots v_{n_i} (= \beta_i)$.

▶ **Observation 24.** *We have $|M_i| = |M_i \cap X_i| + |M_i \cap Y_i| \leq 2k + 2k = 4k$. If $v_r \in X_i \cap M_i$, then for every $v_s \in X_i$ with $s \leq r$, we have $v_s \in M_i$. Similarly, if $v_r \in Y_i \cap M_i$, then for every $v_s \in Y_i$ with $s \geq r$, we have $v_s \in M_i$.*

For $r, s \in [n_i], r \leq s$, let $P_i^{r,s}$ denote the subpath of P_i from vertex v_r to vertex v_s. Also, we say that the subpath $P_i^{r,s}$ is *monochromatic* if for every $r \leq j \leq s$, the vertex v_j is unmarked, and $\texttt{can}(v_j) = \texttt{can}(v_r)$. And we say that a monochromatic subpath $P_i^{r,s}$ is a *maximal monochromatic subpath* if there is no monochromatic subpath of P_i that strictly contains $P_i^{r,s}$. That is, a maximal monochromatic subpath is a maximal subpath of P_i such that all its vertices are unmarked and have the same canonical positions.

▶ **Lemma 25** ($\star$). *Assume that (D, k) is a yes-instance and that (L, R) is a bisection of D of size at most k. Then for every i, the path P_i has at most $4k + 2$ maximal monochromatic subpaths.*

The above lemma shows that $V(D_i) \setminus M_i$ can be partitioned into at most $4k+2$ maximal monochromatic subpaths. Additionally, $V(D_i)$ contains at most $4k$ marked vertices. Let $\mathcal{P}_{f,i}$ denote this partition of $V(D_i)$ into $4k + 2 + 4k = 8k + 2$ parts. That is, every part in $\mathcal{P}_{f,i}$ is either the set of vertices of a maximal monochromatic subpath of P_i or a singleton set consisting of a marked vertex. We call $\mathcal{P}_{f,i}$ the *monochromatic partition* of $V(D_i)$.

▶ **Lemma 26** ($\star$). *Assume that (D, k) is a yes-instance and that (L, R) is a bisection of D of size at most k. Then, for every part $Z \in \mathcal{P}_{f,i}$, either $Z \subseteq L$ or $Z \subseteq R$.*

Kernel. Our kernelization algorithm works as follows. Given (D,k), we begin by using Proposition 2 to construct an $(n,2k,4k^2)$-splitter $\mathcal{F}'$ in time $k^{\mathcal{O}(1)}n\log n$. By Proposition 2 and Remark 23, we get that $|\mathcal{F}'| = k^{\mathcal{O}(1)}$. For each function $f \in \mathcal{F}'$, we do as follows. For each $i \in [4k^2]$ such that $f^{-1}(i) \neq \emptyset$, we find a Hamiltonian path P_i in D_i. We say that $f \in \mathcal{F}$ is a *good function* if for every $i \in [4k^2]$ such that $f^{-1}(i) \neq \emptyset$, the path P_i has at most $4k+2$ maximal monochromatic subpaths. Let $\mathcal{F} \subseteq \mathcal{F}'$ be the collection of good functions in $\mathcal{F}$. We now compute the monochromatic partition $\mathcal{P}_{f,i}$ of $V(D_i)$. Note that $|\mathcal{P}_{f,i}| \leq 8k+2$. Let $\mathcal{P}_f$ be the partition of $V(D)$ obtained by taking the union of $\mathcal{P}_i$ for all i, that is, $\mathcal{P}_f = \bigcup_{i\in[4k^2]} \mathcal{P}_{f,i}$. We have $|\mathcal{P}_f| \leq 4k^2 \cdot (8k+2) = k^{\mathcal{O}(1)}$. This completes the first step our kernelizaition algorithm. Observe that this step can be executed in time $k^{\mathcal{O}(1)}n^{\mathcal{O}(1)}$.

Now, construct the $\mathcal{P}_f$-weighted contraction of D, (with weight function $w_{\mathcal{P}_f}$), and denote it by H_f. Note that $|V(H_f)| = |\mathcal{P}_f| = k^{\mathcal{O}(1)}$. As every vertex $v \in V(H_f)$ corresponds to a part of $\mathcal{P}_f$ of size at most n, we have $w_{\mathcal{P}_f}(v) = |\texttt{part}(v)| \leq n$ for every vertex $v \in V(H_f)$. Hence, $|w_{\mathcal{P}_f}(v)| \leq \log n \leq k$ for every $v \in V(H_f)$ (by Remark 23). Thus, H_f is a vertex-weighted semicomplete digraph with $k^{\mathcal{O}(1)}$ vertices and further, the weight of each vertex in $V(H_f)$ can be encoded using $k^{\mathcal{O}(1)}$ bits. This is the second step of our kernel. Observe that this step also runs in time polynomial in n.

Now, consider the instance $(\{(H_f, w_{\mathcal{P}_f}) \mid f \in \mathcal{F}\}, k)$ of ALO-WMB. Note that the size of the ALO-WMB instance $(\{(H_f, w_{\mathcal{P}_f}) \mid f \in \mathcal{F}\}, k)$ is bounded by $k^{\mathcal{O}(1)}$. The following lemma shows the equivalence of this instance to (D,k).

▶ **Lemma 27** ($\star$). *The instance (D,k) is a yes-instance of* Minimum Directed Bisection *if and only if $(\{(H_f, w_{\mathcal{P}_f}) \mid f \in \mathcal{F}\}, k)$ is a yes-instance of* ALO-WMB.

Note that the first two steps of the kernel – up to the construction of the ALO-WMB instance $(\{(H_f, w_{\mathcal{P}_f}) \mid f \in \mathcal{F}\}, k)$ – runs in polynomial time. Finally, we use the polynomial time reduction in Remark 17 to reduce the instance $(\{(H_f, w_{\mathcal{P}_f}) \mid f \in \mathcal{F}\}, k)$ of ALO-WMB to an equivalent instance (D', k') of Minimum Directed Bisection on semicomplete digraphs. As the size of the instance $(\{(H_f, w_{\mathcal{P}_f}) \mid f \in \mathcal{F}\}, k)$ is bounded by $k^{\mathcal{O}(1)}$, this reduction runs in time polynomial in k. We thus conclude that the size of the instance (D', k') is also bounded by $k^{\mathcal{O}(1)}$. This completes the proof of Theorem 22.

7 Conclusion

We studied the Minimum Directed Bisection problem from the parameterized complexity perspective. In particular, we gave an algorithm with running time $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$ and a polynomial kernel for Minimum Directed Bisection on semicomplete digraphs. Some natural questions that arise from this work are: can the $\log k$ dependence on the exponent in the running time be removed? How far can we reach in improving the size of the kernel? In particular, can one prove a lower bound/existence of a linear/quadratic kernel for Minimum Directed Bisection on semi-complete digraphs? We also believe that our technique of employing splitters can be generalized for the design of kernels for many problems, especially the ones that admit algorithms crucially using chromatic coding. At least, for such problems, the use of splitters can "simulate" the effects of chromatic coding that are sufficient for kernelization.

References

1 Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 49–58, 2009. doi:10.1007/978-3-642-02927-1_6.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

3 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs - theory, algorithms and applications*. Springer, 2002.

4 Florian Barbero, Christophe Paul, and Michal Pilipczuk. Exploring the Complexity of Layout Parameters in Tournaments and Semicomplete Digraphs. *ACM Trans. Algorithms*, 14(3):38:1–38:31, 2018. doi:10.1145/3196276.

5 Florian Barbero, Christophe Paul, and Michal Pilipczuk. Strong immersion is a well-quasi-ordering for semicomplete digraphs. *Journal of Graph Theory*, 90(4):484–496, 2019. doi:10.1002/jgt.22408.

6 Stéphane Bessy, Fedor V Fomin, Serge Gaspers, Christophe Paul, Anthony Perez, Saket Saurabh, and Stéphan Thomassé. Kernels for feedback arc set in tournaments. *Journal of Computer and System Sciences*, 77(6):1071–1078, 2011.

7 Thang Nguyen Bui and Andrew Peck. Partitioning Planar Graphs. *SIAM J. Comput.*, 21(2):203–215, 1992. doi:10.1137/0221016.

8 Paul Camion. Chemins et circuits hamiltoniens des graphes complets. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, 249(21):2151–2152, 1959.

9 Maria Chudnovsky and Paul D. Seymour. A well-quasi-order for tournaments. *J. Comb. Theory, Ser. B*, 101(1):47–53, 2011. doi:10.1016/j.jctb.2010.10.003.

10 M Cygan, F V Fomin, L Kowalik, D Lokshtanov, D Marx, M Pilipczuk, M Pilipczuk, and S Saurabh. *Parameterized algorithms*. Springer, 2015.

11 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed parameter tractable. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 323–332, 2014. doi:10.1145/2591796.2591852.

12 Josep Díaz and George B. Mertzios. Minimum Bisection Is NP-hard on Unit Disk Graphs. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 251–262, 2014. doi:10.1007/978-3-662-44465-8_22.

13 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

14 Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010. doi:10.1016/j.jda.2009.08.001.

15 Uriel Feige. Faster FAST(Feedback Arc Set in Tournaments). *CoRR*, abs/0911.5094, 2009. arXiv:0911.5094.

16 Uriel Feige and Orly Yahalom. On the complexity of finding balanced oneway cuts. *Inf. Process. Lett.*, 87(1):1–5, 2003. doi:10.1016/S0020-0190(03)00251-5.

17 Fedor V. Fomin and Michal Pilipczuk. Subexponential Parameterized Algorithm for Computing the Cutwidth of a Semi-complete Digraph. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 505–516, 2013. doi:10.1007/978-3-642-40450-4_43.

18 Alexandra Fradkin. *Forbidden structures and algorithms in graphs and digraphs*. PhD thesis, Princeton University, 2011.

19 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some Simplified NP-Complete Graph Problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

20 Klaus Jansen, Marek Karpinski, Andrzej Lingas, and Eike Seidel. Polynomial Time Approximation Schemes for MAX-BISECTION on Planar and Geometric Graphs. *SIAM J. Comput.*, 35(1):110–119, 2005. doi:10.1137/S009753970139567X.

21 Marek Karpinski and Warren Schudy. Faster Algorithms for Feedback Arc Set Tournament, Kemeny Rank Aggregation and Betweenness Tournament. In *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, pages 3–14, 2010. doi:10.1007/978-3-642-17517-6_3.

22 Ilhee Kim. *On containment relations in directed graphs.* PhD thesis, Princeton University, 2013.

23 Ilhee Kim and Paul D. Seymour. Tournament minors. *J. Comb. Theory, Ser. B*, 112:138–153, 2015. doi:10.1016/j.jctb.2014.12.005.

24 Michael Lampis, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. *Discrete Optimization*, 8(1):129–138, 2011. doi:10.1016/j.disopt.2010.03.010.

25 Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006. doi:10.1016/j.tcs.2005.10.007.

26 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and Near-Optimal Derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 182–191, 1995. doi:10.1109/SFCS.1995.492475.

27 Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Contraction Decomposition in Unit Disk Graphs and Algorithmic Applications in Parameterized Complexity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1035–1054, 2019. doi:10.1137/1.9781611975482.64.

28 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 255–264, 2008. doi:10.1145/1374376.1374415.

29 L Rédei. Ein kombinatorischer Satz. *Satz. Acta Litt. Szeged*, 7:39–43, 1934.

The Quantifier Alternation Hierarchy of Synchronous Relations

Diego Figueira
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR5800, F-33400 Talence, France
CNRS, ReLaX, UMI2000, Siruseri, India
diego.figueira@labri.fr

Varun Ramanathan
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR5800, F-33400 Talence, France
CNRS, ReLaX, UMI2000, Siruseri, India
varun.ramanathan@labri.fr

Pascal Weil
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR5800, F-33400 Talence, France
CNRS, ReLaX, UMI2000, Siruseri, India
pascal.weil@labri.fr

Abstract

The class of synchronous relations, also known as automatic or regular, is one of the most studied subclasses of rational relations. It enjoys many desirable closure properties and is known to be logically characterized: the synchronous relations are exactly those that are defined by a first-order formula on the structure of all finite words, with the prefix, equal-length and last-letter predicates. Here, we study the quantifier alternation hierarchy of this logic. We show that it collapses at level Σ_3 and that all levels below admit decidable characterizations. Our results reveal the connections between this hierarchy and the well-known hierarchy of first-order defined languages of finite words.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Transducers

Keywords and phrases synchronous relations, automatic relations, first-order logic, characterization, quantifier alternation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.29

Funding Work supported by ANR DeLTA, grant ANR-16-CE40-0007.

1 Introduction

We study classes of relations on finite words, within the class of rational relations. Synchronous relations [8] – also studied as *regular relations* [3] and *automatic relations* [5] – form a subclass of rational relations which is well-behaved from many standpoints. Contrary to rational relations, they enjoy crucial effective properties such as closure under intersection and complement. As a consequence, most paradigmatic problems are decidable for synchronous relations, in the same way as they are for regular languages. Further, they admit clean characterizations both in terms of automata and logic, providing yet more evidence of the connections between logic, formal languages and automata. Due to this good behavior, this class finds various applications in verification [6, 1], automatic structures [5], the theory of transducers and database theory [2].

Synchronous relations contain natural relations such as equality, prefix, or equal-length. In fact, any letter-to-letter transduction, alphabetic morphism or length-preserving rational relation lies within synchronous relations [4].

Synchronous relations are those that are accepted by multi-tape finite automata. A k-tape automaton over an alphabet $\mathbb{A}$ can be naturally seen as an NFA over the alphabet of

k-tuples $\hat{\mathbb{A}} = (\mathbb{A} \cup \{\perp\})^k$ that reads k input words $w_1, \ldots, w_k \in \mathbb{A}^*$ simultaneously, from left to right, the i-th transition reading the tuple from $\hat{\mathbb{A}}$ composed of the i-th letters of each word w_j (or $\perp$ if $i > |w_j|$). Synchronous relations can also be described as finite unions of the componentwise concatenation of a length-preserving rational relation with a recognizable relation – two other well-studied classes of relations [4].

On the other hand, relations can be defined by logical formulæ interpreted on words in $\mathbb{A}^*$: a formula φ with free variables $z_1, \ldots, z_k$ defines the k-ary relation of all tuples $(w_1, \ldots, w_k)$ such that φ holds with the interpretation $z_i \mapsto w_i$ $(1 \leq i \leq k)$. Eilenberg, Elgot and Shepherdson [9] showed that a relation is synchronous if, and only if, it can be defined in this way by a first-order formula using the prefix, equal-length and last letter predicates.

This characterization opens the possibility of exploring classes of synchronous relations specified by fragments of first-order logic. In the present work, we study the quantifier alternation hierarchy in this logic, that is, the classes of relations defined by formulæ with a bounded number of alternations of existential and universal quantifier blocks. This is a natural way of providing small, well-behaved classes (closed under boolean combinations) of synchronous relations. We show that the hierarchy collapses at level Σ_3 and we give clean combinatorial characterizations for its different layers, namely Σ_1, its boolean closure $\mathcal{B}\Sigma_1$, Σ_2 and $\mathcal{B}\Sigma_2$. These characterizations reveal strong links with the classical Σ_1- and Σ_2- fragments of the first-order theory on finite words with the order $<$ relation and letter predicates. Interestingly, the notion of *subwords*, which plays a central role in the characterization of $\Sigma_1[<]$ and $\mathcal{B}\Sigma_1[<]$, must be replaced here by the more subtle notion of *synchronized subwords*.

We also show that these characterizations are decidable: given a synchronous relation, one can decide whether it is defined by a formula in Σ_1 (resp. $\mathcal{B}\Sigma$, Σ_2, $\mathcal{B}\Sigma_2$). Our results provide therefore a complete decision procedure for the alternation hierarchy of synchronous relations.

Section 2 introduces technical preliminaries. Our main results are all stated in Section 3, and their proofs are given in the ensuing sections: Section 4 for the collapse of the hierarchy, Section 5 for what concerns the Σ_1- and $\mathcal{B}\Sigma_1$-fragments and Section 6 for the Σ_2- and $\mathcal{B}\Sigma_2$-fragments.

2 Preliminaries

For any set A and $\bar{a} \in A^k$, we denote by $\bar{a}(i)$ its i-th component, an element of A. If $w \in A^*$ is a word, we denote by $|w|$ its length and, for any $1 \leq i \leq j \leq |w|$, by $w[i]$ the letter of w in i-th position, and by $w[i..j]$ the factor $w[i] \cdots w[j]$ of w between positions i and j. To simplify notation, we let $w[i..j] = \varepsilon$ (the empty word) whenever $1 \leq i \leq j \leq |w|$ does not hold. If u, v are words, we let $u \sqcap v$ be the longest common prefix of u and v.

We will consider relations of a fixed arity $k \geq 2$, over a fixed alphabet $\mathbb{A}$ with at least two letters. Let $\perp$ be a symbol not in $\mathbb{A}$, and let $\mathbb{A}_\perp = \mathbb{A} \cup \{\perp\}$. We will often work with the alphabet $\mathbb{A}_\perp^k$, the direct product of k copies of $\mathbb{A}_\perp$.

Synchronous relations

Given $w_1, \ldots, w_k \in \mathbb{A}^*$, we define the *synchronized word* $\bar{w}$ of the tuple $(w_1, \ldots, w_k)$, written $\bar{w} = w_1 \otimes \cdots \otimes w_k$, to be the word in $(\mathbb{A}_\perp^k)^*$ such that:

- $|\bar{w}| = \max(|w_1|, \ldots, |w_k|)$; and
- for every $i \in \{1, \ldots, |\bar{w}|\}$ and $j \in \{1, \ldots, k\}$, we have $\bar{w}[i](j) = w_j[i]$ if $i \leq |w_j|$, and $\bar{w}[i](j) = \perp$ otherwise.

For example, $abba \otimes c \otimes de = (a, c, d)(b, \perp, e)(b, \perp, \perp)(a, \perp, \perp)$. We let SW_k be the set of all k-synchronized words, that is, $SW_k = \{w_1 \otimes \cdots \otimes w_k : w_1, \ldots, w_k \in \mathbb{A}^*\}$. For $S = \{s_1, \ldots, s_n\} \subseteq \{1, \ldots, k\}$ such that $s_1 < \cdots < s_n$, we define the *projection* $\pi_S \colon SW_k \to (\mathbb{A}^n_\perp)^*$ as $\pi_S(w_1 \otimes \cdots \otimes w_k) = w_{s_1} \otimes \cdots \otimes w_{s_n}$. In the case of a singleton $S = \{i\}$, note that $\pi_S : SW_k \to \mathbb{A}^*$, and we simply write π_i. If $R \subseteq (\mathbb{A}^*)^k$ is a k-ary relation, the *synchronized language of* R, denoted by L_R, is the language $\{w_1 \otimes \cdots \otimes w_k \colon (w_1, \ldots, w_k) \in R\} \subseteq (\mathbb{A}^k_\perp)^*$. The relation R is said to be *synchronous* if L_R is regular. The set of synchronous relations, of arbitrary arity, is denoted by **Sync**.

MSO over finite words

In the classical setting introduced by Büchi (see [21]), languages over an alphabet $\mathbb{A}$ are described by formulæ interpreted over the set of positions of a finite word, using the binary word ordering predicate $<$ and the unary letter predicates a ($a \in \mathbb{A}$) – where $a(i)$ holds if the word carries letter a in position i. Büchi's Theorem [7, 21] states that a language is regular if and only if it is definable by a closed monadic second order formula in this logic, written $\mathbf{MSO}[<, \{a\}_{a \in \mathbb{A}}]$, or $\mathbf{MSO}[<]$ if $\mathbb{A}$ is understood. If φ is a closed formula in $\mathbf{MSO}[<]$, we let $|\varphi|$ be the language in $\mathbb{A}^*$ it defines, and if $\mathcal{F}$ is a set of formulæ, $|\mathcal{F}|$ denotes the class $\{|\varphi| : \varphi \in \mathcal{F}\}$.

First-order formulæ in Büchi's logic define a strict subclass of regular languages, that of *star-free* languages (see [19, 12, 21, 22]). The quantifier alternation hierarchy within $\mathbf{FO}[<]$ forms a strict infinite hierarchy, and it has been the object of intense study (see [17, 18] for an overview). In the sequel, we will only use results regarding the $\mathbf{\Sigma}_1$, $\mathcal{B}\mathbf{\Sigma}_1$, $\mathbf{\Sigma}_2$ and $\mathcal{B}\mathbf{\Sigma}_2$ fragments of $\mathbf{FO}[<]$, possibly enriched with the constant predicate **max**, which stands for the last position in a word (see below in this section and Section 6). Recall that $\mathcal{B}$ designates the boolean closure; that $\mathbf{\Sigma}_1$ is the set of existential formulæ, of the form $\exists z_1 \cdots \exists z_n\ \varphi$ with φ quantifier-free; and that $\mathbf{\Sigma}_2$ consists in the formulæ of the form $\exists z_1 \cdots \exists z_n\ \varphi$ with φ in $\mathcal{B}\mathbf{\Sigma}_1$. The $\mathbf{\Pi}_i$ fragment consists of the negations of the formulæ in $\mathbf{\Sigma}_i$ (e.g., $\mathbf{\Pi}_1$ are all formulæ of the form $\forall z_1 \cdots \forall z_n\ \varphi$ with φ quantifier-free). We will sometimes write $\mathbf{FO}[<](\mathbb{A}^*)$ (or fragments thereof) when we want to make explicit the alphabet $\mathbb{A}$ we work with.

FO over the structure of all finite words

We now turn to the signature introduced by Eilenberg *et al.* [9] to discuss synchronous relations over $\mathbb{A}^*$, namely $\sigma = [\preceq, \mathsf{eq}, (\ell_a)_{a \in \mathbb{A}}]$. These predicates are interpreted as follows:

- $(w_1, w_2) \models x \preceq y$ if and only if w_1 is a *prefix* of w_2;
- $(w_1, w_2) \models \mathsf{eq}(x, y)$ if and only if w_1 and w_2 have *equal length*;
- $w \models \ell_a(x)$ if and only if the *last letter* of w is a.

Every formula φ with free variables $z_1, \ldots, z_k$ defines a k-ary relation written $\|\varphi\|$, namely:

$$\|\varphi\| = \{(w_1, \ldots, w_k) \in (\mathbb{A}^*)^k : (w_1, \ldots, w_k) \models \varphi\}.$$

Let $\mathbf{FO}[\sigma]$ denote the set of first order formulæ with signature σ, and for any $\mathcal{F} \subseteq \mathbf{FO}[\sigma]$ let $\|\mathcal{F}\|$ denote the set of relations definable by formulæ in $\mathcal{F}$. For convenience, we write $x \prec y$ for $(x \preceq y) \wedge \neg(y \preceq x)$, and L_φ for $L_{\|\varphi\|}$. For example, $\varphi(x_1, x_2, x_3) = (x_3 \preceq x_1) \wedge (x_3 \preceq x_2) \wedge \forall z\ (x_3 \prec z \to \neg(z \preceq x_1 \wedge z \preceq x_2))$ defines the set $\|\varphi\|$ of all triples (w_1, w_2, w_3) such that $w_3 = w_1 \sqcap w_2$.

Types and type sequences

For a letter $\bar{a} = (a_1, \ldots, a_k) \in \mathbb{A}_\perp^k$, the *type of* $\bar{a}$ is the subset of $\{1, \ldots, k\}^2$ $\mathsf{type}(\bar{a}) = \{(i,j) : a_i = a_j \neq \perp\}$. The *type* of a synchronized word $\bar{w} = \bar{a}_1 \cdots \bar{a}_n$ is given by $\mathsf{type}(\bar{w}) = \bigcap_{1 \leq i \leq n} \mathsf{type}(\bar{a}_i)$. For example, $\mathsf{type}((a, \perp, a, b)) = \{(1,3), (3,1), (1,1), (3,3), (4,4)\}$ and $\mathsf{type}((a, \perp, a, b)(\perp, \perp, b, b)) = \{(3,3), (4,4)\}$.

In particular, if $\bar{w} \in SW_k$, the successive values $T_1 \supsetneq T_2 \cdots \supsetneq T_n$ taken by the types of the prefixes of $\bar{w}$ form the *type sequence* of $\bar{w}$, written $\mathsf{type\text{-}seq}(\bar{w})$. In such a sequence, we say that T_i is an *end type* if either $i = n$, or $(j,j) \in T_i \setminus T_{i+1}$ for some $j \leq k$ – that is, if $\bar{w} = w_1 \otimes \cdots \otimes w_k$, T_i is an end type in $\mathsf{type\text{-}seq}(\bar{w})$ if the length of the longest prefix of $\bar{w}$ of type T_i is equal to $|w_j|$ for some j. If T is a type, we let $\mathbb{A}_T$ be the set of T-*compatible letters*, $\mathbb{A}_T = \{\bar{a} \in \mathbb{A}_\perp^k : T \subseteq \mathsf{type}(\bar{a}) \subseteq T^\star\}$, where $T^\star = \{(i,j) : (i,i), (j,j) \in T\}$; and let $\mathbb{A}_{-,T} = \{\bar{a} \in \mathbb{A}_\perp^k : T = \mathsf{type}(\bar{a})\}$. If T' is a type such that $T \subsetneq T'$, we also let $\mathbb{A}_{T',T} = \{\bar{a} \in \mathbb{A}_\perp^k : T = T' \cap \mathsf{type}(\bar{a})\}$. Hence, if $\bar{w}\bar{a} \in SW_k$ and $\bar{w}$ has type T (resp. T'), then $\mathsf{type}(\bar{w}\bar{a}) = T$ if and only if $\bar{a} \in \mathbb{A}_T$ (resp. $T \subsetneq T'$ and $\bar{a} \in \mathbb{A}_{T',T}$).

It follows that, if $\bar{T} = (T_1, \ldots, T_n)$ is a type sequence and $K(\bar{T})$ is the set of synchronized words $\bar{w}$ such that $\mathsf{type\text{-}seq}(\bar{w}) = \bar{T}$, then

$$K(\bar{T}) = \mathbb{A}_{-,T_1} \mathbb{A}_{T_1}^* \mathbb{A}_{T_1,T_2} \mathbb{A}_{T_2}^* \cdots \mathbb{A}_{T_{n-1},T_n} \mathbb{A}_{T_n}^*. \tag{1}$$

Note that this product of languages is *deterministic*, that is, given $\bar{w}$, we can determine $\mathsf{type\text{-}seq}(\bar{w})$ and its unique factorization in the product (1) by reading $\bar{w}$ from left to right: the first letter determines T_1, the next factor is the longest written in $\mathbb{A}_{T_1}$, the first letter not in $\mathbb{A}_{T_1}$ (together with T_1) determines T_2, etc.

If $\bar{w} = \bar{w}_1 \cdots \bar{w}_n$ is this factorization, with $\bar{w}_i \in \mathbb{A}_{T_{i-1},T_i} \mathbb{A}_{T_i}^*$ for each i ($\mathbb{A}_{T_-,T_i} \mathbb{A}_{T_i}^*$ for $i = 1$), we say that $\bar{w}_i$ is the *i-th type factor* of $\bar{w}$, written $\mathsf{type\text{-}factor}_i(\bar{w})$.

Synchronized subwords

We denote by $\sqsubseteq$ the (*scattered*) *subword relation* on $\mathbb{A}^*$ (sometimes called *subsequence*): if $u, v \in \mathbb{A}^*$, we have $u \sqsubseteq v$ if there exists a strictly increasing function $p \colon \{1, \ldots, |u|\} \to \{1, \ldots, |v|\}$, called the *witness function*, such that, for all $i \in \{1, \ldots, |u|\}$, $u[i] = v[p(i)]$.

Given $\bar{w} = w_1 \otimes \cdots \otimes w_k$ and $\bar{w}' = w'_1 \otimes \cdots \otimes w'_k$, we say that $\bar{w}$ is a *synchronized subword* of $\bar{w}'$, denoted by $\bar{w} \sqsubseteq_{\mathsf{s}} \bar{w}'$ if and only if $\bar{w} \sqsubseteq \bar{w}'$, with a witness function p which is

- *type preserving*: $\mathsf{type}(\bar{w}[1..i]) = \mathsf{type}(\bar{w}'[1..p(i)])$ for all $1 \leq i \leq |\bar{w}|$; and
- *end preserving*: $p(|w_j|) = |w'_j|$ for all $j \in \{1, \ldots, k\}$.

▶ **Lemma 1.** *For $\bar{u}, \bar{u}' \in SW_k$ with type sequences $\bar{T}$ and $\bar{T}'$, we have $\bar{u} \sqsubseteq_{\mathsf{s}} \bar{u}'$ if and only if $\bar{T}$ is a subsequence of $\bar{T}'$ with a witness function $t \colon \{1, \ldots, |\bar{T}|\} \to \{1, \ldots, |\bar{T}'|\}$ such that, for every i, the i-th type factor of $\bar{u}$ is a subword of the $t(i)$-th type factor of $\bar{u}'$, and they further have the same last letter if $\bar{T}_i$ is an end type of $\bar{T}$.*

Proof. Suppose first that $\bar{u} \sqsubseteq_{\mathsf{s}} \bar{u}'$ and let $p \colon \{1, \ldots, |\bar{u}|\} \to \{1, \ldots, |\bar{u}'|\}$ be a witness function. Let $\bar{T}$ and $\bar{T}'$ be the type sequences of $\bar{u}$ and $\bar{u}'$ and let $\bar{u} = \bar{u}_1 \cdots \bar{u}_n$, $\bar{u}' = \bar{u}'_1 \cdots \bar{u}'_m$ be the type factorizations of $\bar{u}$ and $\bar{u}'$. For each $1 \leq i \leq n$, if $\bar{v}_i = \bar{u}_1 \cdots \bar{u}_i$, then $T_i = \mathsf{type}(\bar{v}_i) = \mathsf{type}(\bar{u}'[1..p(|\bar{v}_i|)])$, so $\bar{T}$ is a subsequence of $\bar{T}'$. Let $t(i)$ be such that $T'_{t(i)} = T_i$. Since p is type-preserving, the factor $\bar{u}_i$ is a subword of $\bar{u}'_{t(i)}$. Both have the same last letter if T_i is an end type for $\bar{u}$, since p is end-preserving.

Conversely, suppose that $T_i = T'_{t(i)}$ and that $\bar{u}_i \sqsubseteq \bar{u}'_{t(i)}$, with witness function p_i (with domain $\{1, \ldots, |\bar{u}_i|\}$). Let p be the function on $\{1, \ldots, |\bar{u}|\}$ obtained by "concatenating" the p_i: $p(|\bar{u}_1 \cdots \bar{u}_{i-1}| + h) = |\bar{u}'_1 \cdots \bar{u}'_{t(i)-1}| + p_i(h)$. It is directly verified that p witnesses $\bar{u} \sqsubseteq_{\mathsf{s}} \bar{u}'$. ◀

Given a quasi-order $\preceq$ over a domain X, the $\preceq$-*upward closure* of an element $x \in X$ is the set $\uparrow_{\preceq} x = \{x' \in X : x \preceq x'\}$. If $S \subseteq X$, we also let $\uparrow_{\preceq} S = \bigcup_{x \in S} \uparrow_{\preceq} x$. Finally, S is $\preceq$-*upward closed* if $S = \uparrow_{\preceq} S$. Henceforward, we write $\uparrow w$ and $\uparrow S$ as short for $\uparrow_{\sqsubseteq} w$ and $\uparrow_{\sqsubseteq} S$; and we write $\uparrow_{\mathsf{s}} \bar{w}$ and $\uparrow_{\mathsf{s}} S$ as short for $\uparrow_{\sqsubseteq_{\mathsf{s}}} w$ and $\uparrow_{\sqsubseteq_{\mathsf{s}}} S$.

A *well-quasi-order* (*wqo*) is a quasi-order $(X, \preceq)$ such that for every infinite sequence $(x_i)_{i \in \mathbb{N}}$ of elements of X, there exist $i < j$ such that $x_i \preceq x_j$. A crucial observation is that, if $\preceq$ is a wqo, then any set has a finite number of $\preceq$-minimal elements.

It is a classical result (Higman's lemma [10], see also [11, chap. 6]) that the subword order on $\mathbb{A}^*$ is a wqo. Unsurprisingly, the same holds for the synchronized subword order.

▶ **Proposition 2.** *For every k, $(SW_k, \sqsubseteq_{\mathsf{s}})$ is a well-quasi-order.*

Proof. If $(\bar{w}_n)_n$ is an infinite sequence of elements of SW_k, we can extract an infinite subsequence of elements with the same type sequence $\bar{T} = (T_1, \ldots, T_n)$ (since there are only finitely many type sequences). Similarly, we can further extract an infinite subsequence where, for each end type T_i, all the i-th type factors end with the same letter.

On this subsequence, $\sqsubseteq_{\mathsf{s}}$ coincides with the intersection of the subword order applied to each of the n type factors. The result follows since the subword order is a wqo and wqo's are closed under intersection. ◀

Bounded subword and synchronized subword classes

It is well-known [21] that $|\mathbf{\Sigma}_1[<]|$ is the set of languages of the form $\uparrow \mathcal{S}$, where $\mathcal{S}$ is a set of words (which can be assumed to be finite by the wqo property). Similarly, $|\mathcal{B}\mathbf{\Sigma}_1[<]|$ is the set of finite unions of languages of the form $\uparrow \mathcal{S} \setminus \uparrow \mathcal{S}'$, where $\mathcal{S}, \mathcal{S}'$ are finite sets of words. For $h \in \mathbb{N}$, let $\sim_h$ be the equivalence relation on $\mathbb{A}^*$ defined by $w_1 \sim_h w_2$ if and only if w_1 and w_2 have the same subwords of length at most h. Then we also know [13, 21] that $|\mathcal{B}\mathbf{\Sigma}_1[<]|$ is the set of finite unions of $\sim_h$-classes, also known as the set of *piecewise testable* languages.

We introduce analogous definitions for synchronized subwords. If $h \in \mathbb{N}$, we let $\approx_h$ be the equivalence relation on synchronized words defined by $\bar{w}_1 \approx_h \bar{w}_2$ if $\bar{w}_1$ and $\bar{w}_2$ have the same set of synchronized subwords of length less than or equal to h. We let $\mathcal{V}_h$ be the set of equivalence classes of $\approx_h$ and $\bar{\mathcal{V}}_h$ be its Boolean closure. Finally, we let $\bar{\mathcal{V}} = \bigcup_{h \in \mathbb{N}} \bar{\mathcal{V}}_h$.

3 Summary of results

We start with an overview of our main results. Their proofs are discussed in the next sections. Theorem 3 refines the already mentioned 1969 result of Eilenberg *et al.* [9], which states that the relations definable in $\|\mathbf{FO}[\sigma]\|$ are exactly the synchronous relations.

▶ **Theorem 3.** *For any alphabet having at least two letters,*

$$\|\mathbf{\Sigma}_1[\sigma]\| \subsetneq \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\| \subsetneq \|\mathbf{\Sigma}_2[\sigma]\| \subsetneq \|\mathcal{B}\mathbf{\Sigma}_2[\sigma]\| \subsetneq \|\mathbf{\Sigma}_3[\sigma]\| = \|\mathbf{FO}[\sigma]\| = \mathbf{Sync}.$$

The characterizations of the $\mathbf{\Sigma}_1[\sigma]$- and the $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-fragments are in terms of synchronized subwords, rather than ordinary subwords as in the case of word languages.

▶ **Theorem 4.** *For any relation R, $R \in \|\mathbf{\Sigma}_1[\sigma]\|$ if and only if $L_R = \uparrow_{\mathsf{s}} L_R$.*

▶ **Theorem 5.** *For any relation R, $R \in \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$ if and only if $L_R \in \bar{\mathcal{V}}$.*

We note the following corollary of Theorem 4, which follows from the wqo property of the synchronized subword order (Proposition 2).

▶ **Corollary 6.** *For every formula $\varphi \in \mathbf{\Sigma}_1[\sigma]$ with k free variables, there exists a finite set $S \subseteq SW_k$ such that $L_\varphi = \uparrow_s S$.*

In contrast, the characterizations of the $\mathbf{\Sigma}_2[\sigma]$- and the $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$-fragments reduce to the corresponding logical fragments for word languages.

▶ **Theorem 7.** *For any relation R, $R \in \|\mathbf{\Sigma}_2[\sigma]\|$ if and only if $L_R \in |\mathbf{\Sigma}_2[<]|$.*

▶ **Corollary 8.** *For any relation R, $R \in \|\mathcal{B}\mathbf{\Sigma}_2[\sigma]\|$ if and only if $L_R \in |\mathcal{B}\mathbf{\Sigma}_2[<]|$.*

These characterizations can then be used to prove the decidability of the membership problems for the different fragments of $\mathbf{FO}[\sigma]$.

▶ **Theorem 9.** *Given a fragment $\mathcal{F} \in \{\mathbf{\Sigma}_1[\sigma], \mathcal{B}\mathbf{\Sigma}_1[\sigma], \mathbf{\Sigma}_2[\sigma], \mathcal{B}\mathbf{\Sigma}_2[\sigma]\}$ and a synchronous relation R (say, an automaton accepting L_R), it is decidable whether $R \in \|\mathcal{F}\|$.*

▶ Remark 10. The decidability of membership in $\|\mathbf{\Sigma}_2[\sigma]\|$ and $\|\mathcal{B}\mathbf{\Sigma}_2[\sigma]\|$ follows directly from the decidability of $|\mathbf{\Sigma}_2[<]|$ [15] and $|\mathcal{B}\mathbf{\Sigma}_2[<]|$ [16], see also [17].

4 Collapse of the alternation hierarchy

Here we prove Theorem 3. The equality $\|\mathbf{FO}[\sigma]\| = \mathbf{Sync}$ was proved in [9]. The collapse at level Σ_3 is established by a folklore argument, in which the runs of a classical (1-tape) automaton are first-order encoded using additional tapes.

Specifically, let R be a k-ary synchronous relation and let $\mathcal{A}$ be a DFA accepting the language L_R, with state set Q. We fix $0, 1$, two distinct letters of $\mathbb{A}$, which will be used to encode the runs of $\mathcal{A}$. If $\bar{w} \in SW_k$ and if $q \in Q$, we let u_q be the word of length $|\bar{w}|$ which carries the letter 1 at each position i such that $\mathcal{A}$ is in state q after reading the i first letters of $\bar{w}$, and carries letter 0 everywhere else.

A $\mathbf{\Sigma}_3[\sigma]$-formula φ with free variables $z_1, \ldots, z_k$ defining R is obtained as follows. A tuple of variables $Y = (y_q)_{q \in Q}$ is quantified existentially, and the rest of the formula, in $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$, verifies that the words u_q ($q \in Q$) assigned to these variables encode the run of $\mathcal{A}$ as above. More precisely, each of these words must have length $|\bar{w}|$ (verified in $\mathbf{\Pi}_2[\sigma]$) and at every position, exactly one of them carries a 1 (verified in $\mathbf{\Pi}_1[\sigma]$). Moreover, the first letter of each u_q must be 1 exactly when q is the state reached from the initial state when reading the tuple of first letters of the words assigned to $z_1, \ldots, z_k$; the notion of first letter, or rather of length 1 prefix is $\mathbf{\Pi}_2[\sigma]$-definable. Similarly, the last state must be one of the final states of $\mathcal{A}$, which is readily verified using (without any quantifier) the last letter predicates ℓ_a. Finally, the compatibility of the u_q's with the transitions of the run of $\mathcal{A}$ on $\bar{w}$ can be encoded with a $\mathbf{\Pi}_2[\sigma]$-formula. Thus $\mathbf{Sync}$ is contained in $\|\mathbf{\Sigma}_3[\sigma]\|$

We now turn to the proof of the strictness of containments in Theorem 3. We observe that a unary relation on $\mathbb{A}$ is nothing but a language in $\mathbb{A}^*$. In that case, the notion of type is trivial, and $w \sqsubseteq_{\mathsf{s}} w'$ if and only if $w \sqsubseteq w'$ and they have the same last letter. In view of Theorems 4 and 5, it follows that a^* is in $\|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$ but not in $\|\mathbf{\Sigma}_1[\sigma]\|$. The other containments are strict because they are in the classical framework. This completes the proof of Theorem 3.

5 $\Sigma_1[\sigma]$ and its boolean closure

We prove Theorems 4 and 5, and we exhibit a decision procedure for membership in $(\mathcal{B})\mathbf{\Sigma}_1[\sigma]$.

5.1 Characterization of $\Sigma_1[\sigma]$

The proof of Theorem 4 is a consequence of the following three properties:

- if φ is a formula of $\mathbf{\Sigma}_1[\sigma]$, then L_φ is $\sqsubseteq_{\mathsf{s}}$-upward closed (Lemma 12 below);
- the relation defined by the $\sqsubseteq_{\mathsf{s}}$-upward closure of any synchronized word is $\mathbf{\Sigma}_1[\sigma]$-definable (Lemma 13 below);
- $\sqsubseteq_{\mathsf{s}}$ is a well-quasi-order on SW_k (Proposition 2 above).

We first show how these three properties imply Theorem 4.

Proof of Theorem 4. If R is $\mathbf{\Sigma}_1[\sigma]$-definable, then $L_R = \uparrow_{\mathsf{s}} L_R$ by Lemma 12. Conversely, suppose that $L_R = \uparrow_{\mathsf{s}} L_R$ and let S be a $\sqsubseteq_{\mathsf{s}}$-minimal subset of L_R, such that $\uparrow_{\mathsf{s}} S = \uparrow_{\mathsf{s}} L_R$. Since $\sqsubseteq_{\mathsf{s}}$ is a wqo by Proposition 2, S is finite, say, $S = \{\bar{w}_1, \ldots, \bar{w}_m\}$. By Lemma 13, for every $1 \leq i \leq m$, there exists a formula $\varphi_i \in \mathbf{\Sigma}_1[\sigma]$ such that $L_{\varphi_i} = \uparrow_{\mathsf{s}} \bar{w}_i$. Letting $\varphi = \bigvee_{1 \leq i \leq m} \varphi_i$, we see that $L_\varphi = \bigcup_{1 \leq i \leq m} \uparrow_{\mathsf{s}} \bar{w}_i = \uparrow_{\mathsf{s}} S = \uparrow_{\mathsf{s}} L_R = L_R$, and hence $R = \|\varphi\| \in \|\mathbf{\Sigma}_1[\sigma]\|$. ◀

Before we prove Lemma 12, we establish the following technical lemma.

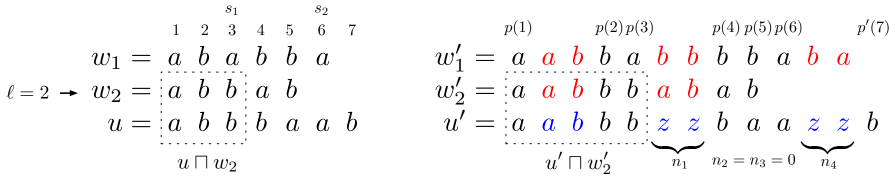
▶ **Lemma 11.** *Let $\bar{w} = w_1 \otimes \cdots \otimes w_k$, and $\bar{w}' = w_1' \otimes \cdots \otimes w_k'$ such that $\bar{w} \sqsubseteq_{\mathsf{s}} \bar{w}'$. For all $u \in \mathbb{A}^*$, there exists $u' \in \mathbb{A}^*$ such that $\bar{w} \otimes u \sqsubseteq_{\mathsf{s}} \bar{w}' \otimes u'$.*

Proof. Let $p\colon \{1, \ldots, |\bar{w}|\} \to \{1, \ldots, |\bar{w}'|\}$ be the witness function for $\bar{w} \sqsubseteq_{\mathsf{s}} \bar{w}'$. Let $s_1 = \max_{i \leq k} |u \sqcap w_i|$ and $\ell \leq k$ be such that $|u \sqcap w_\ell| = s_1$. Finally, let $s_2 = \min(|\bar{w}|, |u|)$, and $u[s_1{+}1..s_2] = a_1 \cdots a_m$. That is, $u = (u \sqcap w_\ell)\ (a_1 \cdots a_m)\ u[s_2{+}1..|u|]$, with the understanding that $u[s_2 + 1..|u|] = \varepsilon$ if $|u| \leq |\bar{w}|$.

For $1 \leq i \leq m$, let $n_i = p(s_1 + i) - p(s_1) - 1$, and $n_{m+1} = |\bar{w}'| - p(s_2)$. Let also z be an arbitrary letter of $\mathbb{A}$. We then define

$$u' = w_\ell'[1..p(s_1)]\ z^{n_1}\, a_1 \cdots z^{n_m}\, a_m\, z^{n_{m+1}}\ u[s_2 + 1..|u|].$$

Example. Let $\bar{w} = w_1 \otimes w_2$, $\bar{w}' = w_1' \otimes w_2'$ and u be defined as follows.



Now let p' be the function defined on $\{1, \ldots, \max(|\bar{w}|, |u|)\}$, which extends p by letting $p'(|\bar{w}| + j) = |\bar{w}'| + j$ for $1 \leq j \leq |u| - |\bar{w}|$. We show that p' is a witness for $\bar{w} \otimes u \sqsubseteq_{\mathsf{s}} \bar{w}' \otimes u'$.

By construction, p' is increasing and $(\bar{w} \otimes u)[i] = (\bar{w}' \otimes u')[p'(i)]$ for every $i \leq \max(|\bar{w}|, |u|)$ $= |\bar{w} \otimes u|$. We must now show that, for each such i, $\mathsf{type}((\bar{w} \otimes u)[1..i]) = \mathsf{type}((\bar{w}' \otimes u)[1..p'(i)])$, and that $p'(|u|) = |u'|$. For convenience, we write $\bar{w}_u$ for $\bar{w} \otimes u$ and $\bar{w}'_{u'}$ for $\bar{w}' \otimes u'$.

- If $1 \leq i \leq s_1$, then $p'(i) = p(i)$, the u-component of each letter of $\bar{w}_u[1..i]$ coincides with its w_ℓ-component, and the u'-component of each letter of $\bar{w}'_{u'}[1..p(i)]$ coincides with its w'_ℓ-component. It follows that $\mathsf{type}(\bar{w}_u[1..i])$ is the symmetric transitive closure of $\mathsf{type}(\bar{w}[1..i]) \cup \{(\ell, k + 1)\}$. Similarly, $\mathsf{type}(\bar{w}'_{u'}[1..p(i)])$ is the symmetric transitive closure of $\mathsf{type}(\bar{w}'[1..p(i)]) \cup \{(\ell, k + 1)\}$. Since $\mathsf{type}(\bar{w}[1..i]) = \mathsf{type}(\bar{w}'[1..p(i)])$, we have $\mathsf{type}(\bar{w}_u[1..i]) = \mathsf{type}(\bar{w}'_{u'}[1..p'(i)])$. In particular, we have $u[i] = w_\ell[i] = w'_\ell[p'(i)] = u'[p'(i)]$. If $i = |u|$, then $s_1 = i$ and, by definition, $u' = w'_\ell[1..p(i)]$. It follows that $|u'| = p(i) = p'(i)$.

- If $s_1 < i \leq s_2$, again we have $p'(i) = p(i)$. Moreover, $\mathsf{type}(\bar{w}_u[1..i]) = \mathsf{type}(\bar{w}[1..i]) \cup \{(k+1, k+1)\}$ since the u-component differs from any other component on at least one position less than or equal to i. For the same reason, $\mathsf{type}(\bar{w}'_{u'}[1..p'(i)]) = \mathsf{type}(\bar{w}'[1..p(i)]) \cup \{(k+1, k+1)\} = \mathsf{type}(\bar{w}_u[1..i])$. Also, by definition of the n_j, we get $u[i] = u'[p(i)]$ and, as above, if $i = |u|$, we find that $p(i) = p(|u'|)$.
- If $s_2 < i \leq |u|$, then $p'(i) - |\bar{w}'| = i - |\bar{w}| = |u[s_2+1..i]|$. In particular, $p'(|u|) = |\bar{w}'| + |u[s_2+1..i]| = |u'|$. Moreover, $\mathsf{type}(\bar{w}_u[1..i]) = \{(k+1, k+1)\} = \mathsf{type}(\bar{w}'_{u'}[1..p'(i)])$. ◀

▶ **Lemma 12.** *If φ is a formula in $\mathbf{\Sigma}_1[\sigma]$, then L_φ is $\sqsubseteq_s$-upward closed.*

Proof. First observe that if the synchronized words $\bar{w} = w_1 \otimes \cdots \otimes w_k$ and $\bar{w}' = w'_1 \otimes \cdots \otimes w'_k$ satisfy $\bar{w} \sqsubseteq_{\mathsf{s}} \bar{w}'$, then, for all $i, j \in \{1, \ldots, k\}$, we have:

- $w_i \preceq w_j$ if and only if $w'_i \preceq w'_j$;
- $|w_i| = |w_j|$ if and only if $|w'_i| = |w'_j|$;
- if $|w_i| = |w'_i| > 0$, then w_i and w'_i have the same last letter.

We now proceed by induction on the number of quantified variables of φ. If φ is quantifier-free, these three properties show that L_φ is $\sqsubseteq_{\mathsf{s}}$-upward closed.

If φ is not quantifier-free, we have $\varphi(y_1, \ldots, y_k) = \exists x\ \psi(y_1, \ldots, y_k, x)$ for some $\psi \in \mathbf{\Sigma}_1[\sigma]$. Let $\bar{w}, \bar{w}' \in SW_k$ such that $\bar{w} \sqsubseteq_{\mathsf{s}} \bar{w}'$ and $\bar{w} \models \varphi$. Then there is $u \in \mathbb{A}^*$ such that $\bar{w} \otimes u \models \psi$. By Lemma 11, there also exists $u' \in \mathbb{A}^*$ such that $\bar{w} \otimes u \sqsubseteq_{\mathsf{s}} \bar{w}' \otimes u'$. Since $\|\psi\|$ is $\sqsubseteq_{\mathsf{s}}$-upward closed by induction, $\bar{w}' \otimes u' \models \psi$, and hence $\bar{w}' \models \varphi$. This completes the proof. ◀

▶ **Lemma 13.** *If $\bar{w}$ is a synchronized word, then the relation defined by $\uparrow_{\mathsf{s}}\bar{w}$ is $\mathbf{\Sigma}_1[\sigma]$-definable.*

Proof. Let $\bar{w} = w_1 \otimes \cdots \otimes w_k \in SW_k$. We define a formula $\varphi(z_1, \ldots, z_k)$ (dependent on $\bar{w}$) whose synchronized language is $\uparrow_{\mathsf{s}}\bar{w}$, using existential quantification on a set consisting of one variable for each w_i and one for each position within w_i. Formally, let $X = \{x_{i,j} : 1 \leq i \leq k, 1 \leq j \leq |w_i|\}$. Then $\varphi(z_1, \ldots, z_k) = \exists X.\psi(z_1, \ldots, z_k, X)$, where ψ is the conjunction of the following formulæ for every $i \in \{1, \ldots, k\}$:

(1) $z_i = x_{i,|w_i|}$;
(2) for every $1 \leq j < |w_i|$: $x_{i,j} \prec x_{i,j+1}$;
(3) for every $1 \leq j \leq |w_i|$: $\ell_{w_i[j]}(x_{i,j})$;
(4) for every $1 \leq i' \leq k$ and $j \leq \min\{|w_i|, |w_{i'}|\}$: $\mathsf{eq}(x_{i,j}, x_{i',j})$;
(5) for every $1 \leq i' \leq k$ and $j \leq \min\{|w_i|, |w_{i'}|\}$ such that $w_i[1..j] = w_{i'}[1..j]$: $x_{i,j} = x_{i',j}$.

Let $\bar{w}' = w'_1 \otimes \cdots \otimes w'_k$. First assume that $\bar{w}' \in \uparrow_{\mathsf{s}}\bar{w}$: we want to show that $\bar{w}' \in L_\varphi$. Let p be a witness function for $\bar{w} \sqsubseteq_{\mathsf{s}} \bar{w}'$. For each $1 \leq i \leq k$ and $1 \leq j \leq |w_i|$, let the word $w'_i[1..p(j)]$ be assigned to variable $x_{i,j}$. It is readily verified that $(w'_1, \ldots, w'_k)$ satisfies $\varphi(z_1, \ldots, z_k)$ and hence $\bar{w} \in \|\varphi\|$.

Conversely, suppose that $\bar{w}' \in L_\varphi$. There exists an assignment α for the variables in φ such that $(\bar{w}, \alpha) \vDash \varphi$. We define a function $p\colon \{1, \ldots, |\bar{w}|\} \to \{1, \ldots, |\bar{w}'|\}$ as follows.

If $1 \leq j \leq |\bar{w}|$, there exists $1 \leq i \leq k$ such that $j \leq |w_i|$, and we let $p(j) = |\alpha(x_{i,j})|$. Condition (4) in the definition of φ shows that p is well-defined. Condition (2) shows that it is increasing and Condition (3) shows that $\bar{w}[j] = \bar{w}'[p(j)]$, so that p is a witness function for $\bar{w} \sqsubseteq \bar{w}'$. Finally, Conditions (5) and (1) show that p is type- and end-preserving, thus establishing that it is a witness for $\bar{w} \sqsubseteq_s \bar{w}'$. ◀

5.2 Characterization of $\mathcal{B}\Sigma_1[\sigma]$

The following lemma establishes one of the implications of Theorem 5.

▶ **Lemma 14.** *For every $\varphi \in \mathcal{B}\mathbf{\Sigma}_1[\sigma]$, $L_\varphi \in \bar{\mathcal{V}}$.*

Proof. By a standard transformation, φ is logically equivalent to a formula $\bigvee_{i=1}^{n} \psi_i \wedge \psi_i'$ in disjunctive normal form, where $\psi_i \in \mathbf{\Sigma}_1[\sigma]$ and $\psi_i' \in \mathbf{\Pi}_1[\sigma]$ for every i. By Corollary 6, there exist finite sets $\mathcal{S}_i$ and $\mathcal{S}_i'$ $(1 \leq i \leq n)$ of synchronized words such that $L_{\psi_i} = \uparrow_{\mathsf{s}}\mathcal{S}_i$ and $L_{\neg\psi_i'} = \uparrow_{\mathsf{s}}\mathcal{S}_i'$. Let $\mathcal{S} = \bigcup_{i=1}^{n} \mathcal{S}_i \cup \mathcal{S}_i'$ and let $h = \max\{|\bar{u}| : \bar{u} \in \mathcal{S}\}$.

If $\bar{w}, \bar{w}'$ are synchronized words such that $\bar{w} \approx_h \bar{w}'$, then $\bar{w}$ and $\bar{w}'$ have the same synchronized subwords of length at most h, and hence the same synchronized subwords in each $\mathcal{S}_i$ and each $\mathcal{S}_i'$ (since these sets contain only words of length at most h). If $\bar{w} \in L_\varphi$, then for some i, $\bar{w}$ contains a synchronized subword in $\mathcal{S}_i$ and none in $\mathcal{S}_i'$. The same holds therefore for $\bar{w}'$, and $\bar{w}' \in L_{\varphi_i \wedge \psi_i'}$. Thus $\bar{w}' \in L_\varphi$, which completes the proof. ◀

To establish the converse implication, we consider $h \in \mathbb{N}$ and $L \in \bar{\mathcal{V}}_h$, such that L is a finite union of $\approx_h$-classes $[\bar{w}_1]_h, \ldots, [\bar{w}_n]_h$, and we show that $L = L_\varphi$ for some $\varphi \in \mathcal{B}\mathbf{\Sigma}_1[\sigma]$.

For each $1 \leq i \leq n$, let $\mathcal{S}_i$ be the set of synchronized subwords of $\bar{w}_i$ of length at most h, and $\mathcal{S}_i'$ be the complement of $\mathcal{S}_i$ within the set of synchronized words of length at most h. Both are finite and, by Lemma 13, there exist $\mathbf{\Sigma}_1[\sigma]$-formulæ ψ_i and ψ_i' such that $L_{\psi_i} = \uparrow_{\mathsf{s}}\mathcal{S}_i$ and $L_{\psi_i'} = \uparrow_{\mathsf{s}}\mathcal{S}_i'$. Then, for each $1 \leq i \leq n$, $[\bar{w}_i]_h = \uparrow_{\mathsf{s}}\mathcal{S}_i \setminus \uparrow_{\mathsf{s}}\mathcal{S}_i' = L_{\psi_i \wedge \neg\psi_i'}$ and hence, $L = L_\varphi$ with $\varphi = \bigvee_i \psi_i \wedge \neg\psi_i'$. This completes the proof of Theorem 5, since $\varphi \in \mathcal{B}\mathbf{\Sigma}_1[\sigma]$.

5.3 Deciding membership in $\|\mathbf{\Sigma}_1[\sigma]\|$

In view of Theorem 4 and of the properties of regular languages (namely the decidability of equality), membership decidability for $\|\mathbf{\Sigma}_1[\sigma]\|$ reduces to proving the following proposition.

▶ **Proposition 15.** *Given a regular language $L \subseteq SW_k$, its upward-closure $\uparrow_{\mathsf{s}}L$ is regular and computable.*

We begin with some preliminary definitions, which will also be used in the next section. For $\mathcal{S} \subseteq SW_k$, let $\uparrow^{\ell} \mathcal{S} = \{\bar{w} \in SW_k : \exists \bar{u} \in \mathcal{S}\ \bar{u} \sqsubseteq \bar{w}$ and $\bar{u}, \bar{w}$ have the same last letter$\}$. Let $\mathcal{A}$ be a deterministic automaton accepting L, with state set Q and initial state q_0. For $p, r \in Q$, we let $\mathcal{A}(p, r)$ be the same as $\mathcal{A}$, with p as initial state and $\{r\}$ as final states, and denote by $\mathsf{Lang}(\mathcal{A}(p, r))$ the language accepted by $\mathcal{A}(p, r)$.

We say that a state sequence $\bar{q} = (q_1, \ldots, q_n) \in Q^n$ is *$\bar{T}$-compatible* in $\mathcal{A}$ if q_1 is reachable from q_0 by reading a word in $\mathbb{A}_{-,T_1}\mathbb{A}_{T_1}^*$, q_2 is reachable from q_1 by reading a word in $\mathbb{A}_{T_1,T_2}\mathbb{A}_{T_2}^*$, etc. In addition, we require q_n to be a final state of $\mathcal{A}$. Observe that, given $\bar{T}$, the set of $\bar{T}$-compatible state sequences is finite and computable.

If $\bar{q}$ is $\bar{T}$-compatible, we let $L(\bar{T}, \bar{q}, i)$ be the intersection of the language accepted by $\mathcal{A}(q_{i-1}, q_i)$ with $\mathbb{A}_{T_{i-1},T_i}\mathbb{A}_{T_i}^*$ ($\mathbb{A}_{-,T_1}\mathbb{A}_{T_1}^*$ if $i = 1$). In particular, if $\bar{w} \in SW_k$ and $\mathsf{type\text{-}seq}(\bar{w}) = \bar{T}$, then $\bar{w} \in L$ if and only if there exists a $\bar{T}$-compatible sequence $\bar{q}$ such that $\bar{w} \in L(\bar{T}, \bar{q}, 1) \cdots L(\bar{T}, \bar{q}, n)$ (uniquely determined, due to determinism). Note that the n factors of $\bar{w}$ thus determined are its type factors. In particular, $L = \bigcup L(\bar{T}, \bar{q}, 1) \cdots L(\bar{T}, \bar{q}, n)$, where the union runs over all type sequences $\bar{T}$ and all $\bar{T}$-compatible state sequences $\bar{q}$ of $\mathcal{A}$. This is a finite union, all of whose terms are explicitly computable.

Proof of Proposition 15. For L, $\mathcal{A}$, $\bar{T}$, $\bar{q}$, i as above, let $\widehat{L}(\bar{T}, \bar{q}, i)$ be $\widehat{L}(\bar{T}, \bar{q}, i) = \uparrow L(\bar{T}, \bar{q}, i) \cap \mathbb{A}_{T_i}^*$ if T_i is not an end type or $\widehat{L}(\bar{T}, \bar{q}, i) = \uparrow^{\ell} L(\bar{T}, \bar{q}, i) \cap \mathbb{A}_{T_i}^*$ otherwise. We now show that

$$\uparrow_{\mathsf{s}}L = \bigcup \widehat{L}(\bar{T}, \bar{q}, 1) \cdots \widehat{L}(\bar{T}, \bar{q}, n).$$

Note that the closure $\uparrow K$ of a regular language K is regular and computable (by adding self loops to the states of an automaton accepting K), and the operation $L(\bar{T}, \bar{q}, i) \mapsto \widehat{L}(\bar{T}, \bar{q}, i)$ is therefore computable, implying Proposition 15.

The proof is essentially an application of Lemma 1. Suppose first that $\bar{w} \in {\uparrow_{\mathsf{s}}} L$, that is, there exists $\bar{u} \in L$ such that $\bar{u} \sqsubseteq_{\mathsf{s}} \bar{w}$. Let $\bar{T} = \mathsf{type\text{-}seq}(\bar{u}) = (T_1, \dots, T_n)$ and let $\bar{q}$ be the $\bar{T}$-compatible state sequence determined by reading $\bar{u}$ in $\mathcal{A}$. By Lemma 1, $\bar{T} \sqsubseteq \mathsf{type\text{-}seq}(\bar{w})$, with a witness function t such that, for each $1 \leq i \leq n$, the i-th type factor $\bar{u}_i$ of $\bar{u}$ is a subword of $\bar{w}_{t(i)}$, the $t(i)$-th type factor of $\bar{w}$ (with an additional last letter condition if T_i is an end type). Therefore $\bar{u}_i$ is also a subword of $\bar{w}_{t(i-1)+1} \cdots \bar{w}_{t(i)}$, with the same last letter condition in the case of end types. Since $\bar{u}_i \in L(\bar{T}, \bar{q}, i)$, this means that $\bar{w}_{t(i-1)+1} \cdots \bar{w}_{t(i)} \in \widehat{L}(\bar{T}, \bar{q}, i)$ and hence $\bar{w} \in \widehat{L}(\bar{T}, \bar{q}, 1) \cdots \widehat{L}(\bar{T}, \bar{q}, n)$.

Conversely, suppose that $\bar{w} \in \widehat{L}(\bar{T}, \bar{q}, 1) \cdots \widehat{L}(\bar{T}, \bar{q}, n)$ for some type sequence $\bar{T}$ and $\bar{T}$-compatible state sequence $\bar{q}$. For each $1 \leq i \leq n$, let $\bar{u}_i \in L(\bar{T}, \bar{q}, i)$ be such that $\bar{u}_i \sqsubseteq \bar{w}_i$, with witness function p_i (and such that $p_i(|u_i|) = |w_i|$ if T_i is an end type). By construction of the $L(\bar{T}, \bar{q}, i)$'s, the word $\bar{u} = \bar{u}_1 \cdots \bar{u}_n$ is in L, with type factors $\bar{u}_1, \dots, \bar{u}_n$. Moreover $\bar{u} \sqsubseteq_{\mathsf{s}} \bar{w}$ for the witness function obtained by "concatenating" the functions p_i: $p(i) = p_1(i)$ for $i \leq |\bar{u}_1|$, and $p(|\bar{u}_1 \cdots \bar{u}_{i-1}| + h) = |\bar{w}_1 \cdots \bar{w}_{i-1}| + p_i(h)$ for every $1 < i \leq n$ and $1 \leq h \leq |\bar{u}_i|$. ◀

5.4 Deciding membership in $\|\mathcal{B}\boldsymbol{\Sigma}_1[\sigma]\|$

As a first step, we note the following.

▶ **Lemma 16.** *If $\bar{T}$ is a type sequence, then $K(\bar{T})$ is $\mathcal{B}\boldsymbol{\Sigma}_1[\sigma]$-definable.*

Proof. Let $S_{\bar{T}}$ be the set of $\sqsubseteq_s$-minimal elements of $K(\bar{T})$, a finite set by Proposition 2. Then $K(\bar{T}) \subseteq {\uparrow_{\mathsf{s}}} S_{\bar{T}}$. Moreover, if $\bar{u} \in {\uparrow_{\mathsf{s}}} S_{\bar{T}}$, then $\bar{T} \sqsubseteq \mathsf{type\text{-}seq}(\bar{u})$ by Lemma 1. It follows that $K(\bar{T}) = {\uparrow_{\mathsf{s}}} S_{\bar{T}} \setminus \bigcup \{{\uparrow_{\mathsf{s}}} S_{\bar{T}'} : \bar{T} \sqsubseteq \bar{T}', \bar{T} \neq \bar{T}'\}$. The statement then follows from Theorem 4. ◀

Since there are finitely many type sequences $\bar{T}$ and each $K(\bar{T})$ is computable, membership of a language L in $\|\mathcal{B}\boldsymbol{\Sigma}_1[\sigma]\|$ is equivalent to the membership of each $L \cap K(\bar{T})$ in $\|\mathcal{B}\boldsymbol{\Sigma}_1[\sigma]\|$.

We now fix a type sequence $\bar{T} = (T_1, \dots, T_n)$. Our next step is a technical characterization of $\|\mathcal{B}\boldsymbol{\Sigma}_1[\sigma]\|$ for languages within $K(\bar{T})$. For each $1 \leq i \leq n$, let $\mathcal{F}_i = \mathcal{B}\boldsymbol{\Sigma}_1[<, \mathbf{max}](\mathbb{A}^*_{T_i})$ if T_i is an end type in $\bar{T}$, and $\mathcal{F}_i = \mathcal{B}\boldsymbol{\Sigma}_1[<](\mathbb{A}^*_{T_i})$ otherwise. Let also $\mathcal{G}_1 = \{\mathbb{A}_{-,T_1} \mathbb{A}^*_{T_1} \cap H \colon H \in |\mathcal{F}_1|\}$ and, for $i \geq 2$, $\mathcal{G}_i = \{\mathbb{A}_{T_{i-1},T_i} \mathbb{A}^*_{T_i} \cap H \colon H \in |\mathcal{F}_i|\}$.

▶ **Lemma 17.** *A regular language $L \subseteq K(\bar{T})$ is in $\|\mathcal{B}\boldsymbol{\Sigma}_1[\sigma]\|$ if and only if, for each $\bar{T}$-compatible state sequence $\bar{q}$ and $1 \leq i \leq n$, $L(\bar{T}, \bar{q}, i) \in \mathcal{G}_i$.*

Proof. For convenience, we write $L(\bar{q}, i)$ for $L(\bar{T}, \bar{q}, i)$. First assume that every $L(\bar{q}, i) \in \mathcal{G}_i$, that is, there exists a $\mathcal{B}\boldsymbol{\Sigma}_1[<]$-definable language $H(\bar{q}, i) \subseteq \mathbb{A}^*_{T_i}$ such that $L(\bar{q}, i) = \mathbb{A}_{T_{i-1},T_i} \mathbb{A}^*_{T_i} \cap H(\bar{q}, i)$ (or, if $i = 1$, $L(\bar{q}, 1) = \mathbb{A}_{-,T_1} \mathbb{A}^*_{T_1} \cap H(\bar{q}, 1)$). Then $H(\bar{q}, i)$ is the finite union of languages of the form $H(\bar{q}, i, j) = {\uparrow} S(\bar{q}, i, j) \setminus {\uparrow} S'(\bar{q}, i, j)$ (or ${\uparrow^\ell} S(\bar{q}, i, j) \setminus {\uparrow^\ell} S'(\bar{q}, i, j)$ if T_i is an end type), for $1 \leq j \leq n_{\bar{q},i}$, where the $S(\bar{q}, i, j)$'s and $S'(\bar{q}, i, j)$'s are finite sets. Then $L(\bar{q}, i)$ is the union of the $L(\bar{q}, i, j) = \mathbb{A}_{T_{i-1},T_i} \mathbb{A}^*_{T_i} \cap H(\bar{q}, i, j)$ (or, if $i = 1$, $L(\bar{q}, 1, j) = \mathbb{A}_{-,T_1} \mathbb{A}^*_{T_1} \cap H(\bar{q}, 1, j)$). If $\bar{\jmath} = (j_1, \dots, j_n)$ is such that $1 \leq j_i \leq n_{\bar{q},i}$ for each $1 \leq i \leq n$, let $L(\bar{q}, \bar{\jmath}) = L(\bar{q}, 1, j_1) \cdots L(\bar{q}, n, j_n)$. Then L is the (finite) union of the $L(\bar{q}, \bar{\jmath})$. Now let

- $\mathcal{S}(\bar{q}, \bar{\jmath}) = \{\bar{w} \in K(\bar{T}) \colon$ for all $i \in \{1, \dots, n\}, \mathsf{type\text{-}factor}_i(\bar{w}) \in S(\bar{q}, i, j_i)\}$ and
- $\mathcal{S}'(\bar{q}, \bar{\jmath}) = \{\bar{w} \in K(\bar{T}) \colon$ for some $i \in \{1, \dots, n\}, \mathsf{type\text{-}factor}_i(\bar{w}) \in S'(\bar{q}, i, j_i)\}$.

Then, $L(\bar{q}, \bar{\jmath}) = K(\bar{T}) \cap ({\uparrow} \mathcal{S}(\bar{q}, \bar{\jmath}) \setminus {\uparrow} \mathcal{S}'(\bar{q}, \bar{\jmath})) \in \|\mathcal{B}\boldsymbol{\Sigma}_1[\sigma]\|$. It follows that $L \in \|\mathcal{B}\boldsymbol{\Sigma}_1[\sigma]\|$.

Conversely, suppose that for some $\bar{q}$ and some i, $L(\bar{q}, i) \notin \mathcal{G}_i$. In view of Theorem 5, we want to show that L is not a union of $\approx_r$-classes for any $r \geq 1$. Let r be now fixed. We only need to exhibit words $\bar{w} \in L$ and $\bar{w}' \notin L$ such that $\bar{w} \approx_r \bar{w}'$.

Let $\sim_r^i$ be the relation on $\mathbb{A}_{T_i}^*$ given by $\bar{u} \sim_r^i \bar{v}$ if $\bar{u} \sim_r \bar{v}$ and, either the first letters of $\bar{u}$ and $\bar{v}$ are both in $\mathbb{A}_{T_{i-1}}$ or neither is. By means of contradiction, suppose $L(\bar{q}, i)$ is the finite union of the $\sim_r^i$ classes $[\bar{u}_1], \ldots, [\bar{u}_m]$. Then each $\bar{u}_j \in \mathbb{A}_{T_{i-1},T_i}\mathbb{A}_{T_i}^*$ and, by definition of $\sim_r^i$, $[\bar{u}_j] = \mathbb{A}_{T_{i-1},T_i}\mathbb{A}_{T_i}^* \cap [\![\bar{u}_j]\!]$, where $[\![\bar{u}_j]\!]$ denotes the $\sim_r$-class of $\bar{u}_j$. Therefore $L(\bar{q}, i) = \mathbb{A}_{T_{i-1},T_i}\mathbb{A}_{T_i}^* \cap M$, where M is the union of the $[\![\bar{u}_j]\!]$'s. Since $M \in |\mathcal{B}\mathbf{\Sigma}_1[<](\mathbb{A}_{T_i})|$, this shows that $L(\bar{q}, i) \in \mathcal{G}_i$, a contradiction. (In the case where T_i is an end type, we need to reason with the intersection of $\sim_r^i$ with the same-last-letter equivalence.)

Now, since $L(\bar{q}, i)$ is not a union of $\sim_r^i$-classes, there exist words $\bar{u}_i, \bar{u}'_i \in \mathbb{A}_{T_{i-1},T_i}\mathbb{A}_{T_i}^*$ such that $\bar{u}_i \sim_r \bar{u}'_i$ (and if T_i is an end type they have the same last letter) and exactly one of them is in $L(\bar{q}, i)$. Say $\bar{u}_i \in L(\bar{q}, i)$, such that $\bar{u}_i \in \mathsf{Lang}(\mathcal{A}(q_{i-1}, q_i))$ and $\bar{u}'_i \in \mathsf{Lang}(\mathcal{A}(q_{i-1}, q'_i))$ for some $q_i \neq q'_i$. Assuming wlog that $\mathcal{A}$ is minimal for L, there exist a word $\bar{y}$ and states p, p' of which exactly one is accepting, such that $\bar{y} \in \mathsf{Lang}(\mathcal{A}(q_i, p)) \cap \mathsf{Lang}(\mathcal{A}(q'_i, p'))$. Let $\bar{x} \in L(\bar{q}, 1) \cdots L(\bar{q}, i-1)$, $\bar{w} = \bar{x}\bar{u}_i\bar{y}$ and $\bar{w}' = \bar{x}\bar{u}'_i\bar{y}$. Then exactly one of $\bar{w}$, $\bar{w}'$ is in L. Since $L \subseteq K(\bar{T})$, this implies that $\bar{y} \in \mathbb{A}_{T_i}^*\mathbb{A}_{T_i,T_{i+1}}\mathbb{A}_{T_{i+1}}^* \cdots \mathbb{A}_{T_{n-1},T_n}\mathbb{A}_{T_n}^*$ ($\mathbb{A}_{T_n}^*$ if $i = n$). Consequently, $\bar{w}$ and $\bar{w}'$ have the same type sequence $\bar{T}$, with the same type factors except for the i-th one. Moreover, $\mathsf{type\text{-}factor}_i(\bar{w}) = \bar{u}_i\bar{u}'$ and $\mathsf{type\text{-}factor}_i(\bar{w}') = \bar{u}'_i\bar{u}'$, where $\bar{u}'$ is the longest $\mathbb{A}_{T_i}^*$ prefix of $\bar{u}$. Since $\bar{u}_i \sim_r \bar{u}'_i$, $\bar{u}_i\bar{u}' \sim_r \bar{u}'_i\bar{u}'$. Then $\bar{w} \approx_r \bar{w}'$ is a consequence of Lemma 1. ◀

In view of Lemma 17 and since each $L(\bar{T}, \bar{q}, i)$ is computable (Section 5.3), the decidability of $\|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$ will be established if we show that membership in each $\mathcal{G}_i$ is decidable, which is the object of the following lemma.

▶ **Lemma 18.** *Let A be an alphabet and $B \subseteq A$. Then, it is decidable whether a regular language is in $\mathcal{W}_B = \{BA^* \cap L \colon L \in \mathcal{B}\mathbf{\Sigma}_1[<](A)\}$.*

Proof. We will prove the following characterization of $\mathcal{W}_B$: a regular language $K \in \mathcal{W}_B$ if and only if $K \subseteq BA^*$ and for every $b \in B$, $b^{-1}K = \{u \in A^* \colon bu \in K\} \in |\mathcal{B}\mathbf{\Sigma}_1[<](A)|$. Since $\mathcal{B}\mathbf{\Sigma}_1[<]$ membership is decidable [20], the result follows directly.

If $K \in \mathcal{W}_B$, then $K = BA^* \cap L$ for some $L \in |\mathcal{B}\mathbf{\Sigma}_1[<](A)|$. Therefore, $K \subseteq BA^*$ and, for every $b \in B$, $b^{-1}K = b^{-1}L \in |\mathcal{B}\mathbf{\Sigma}_1[<]|$ (since $\mathcal{B}\mathbf{\Sigma}_1[<$ is closed under left quotients).

Conversely, suppose that each $b^{-1}K$ ($b \in B$) is a $\mathcal{B}\mathbf{\Sigma}_1[<]$-language. Then there exists r such that each of these languages is a union of $\sim_r$-classes. Say that $u \sim_{r+1}^B v$ if $u \sim_{r+1} v$ and, either u, v have the same first letter in B or both their first letters are in $A \setminus B$. Suppose there exist words u, v such that $u \sim_{r+1}^B v$ and $u \in K$. Then $u \in BA^*$ and v has same first letter as u, say b, so that $u = bu'$ and $v = bv'$. In particular, $u' \sim_r v'$. Since $u' \in b^{-1}K$ and $b^{-1}K$ is a union of $\sim_r$-classes, it follows that $v' \in b^{-1}K$ and hence $v \in K$. Therefore, K is the union of the $\sim_{r+1}^B$-classes of a finite set of words $u_1, \ldots, u_n$ and if L is the union of the $\sim_{r+1}$-classes of the same words, then $L \in |\mathcal{B}\mathbf{\Sigma}_1[<]|$ and $L \cap BA^* = K$. That is, $K \in \mathcal{W}_B$. ◀

6 $\Sigma_2[\sigma]$ and its boolean closure

For any alphabet A, an *A-monomial* is a language of the form $A_1^*a_1A_2^*a_2 \cdots A_n^*a_nA_{n+1}^*$, where $A_1, A_2, \ldots, A_{n+1} \subseteq A$ and $a_1, a_2, \ldots, a_n \in A$. An *$A$-polynomial* is a finite union of A-monomials.

▶ Remark 19. It is known [14] that $\Sigma_2[<](A)$ sentences define exactly the set of A-polynomials. A non-trivial consequence is that the set of A-polynomials is closed under intersection.

Not every $\mathbb{A}_\perp^k$-polynomial respects the structural properties (on the positions of $\perp$) of synchronized words. For example $(a,\perp)^*(b,b)(a,a)^*$ is a polynomial over $\mathbb{A}_\perp^k$ for $\mathbb{A} = \{a,b\}$ and $k = 2$ but it does not define a relation. In order to characterize subsets of SW_k which are polynomials, we introduce the notion of $\perp$-*consistency.*

If $\bar{a} = a_1 \otimes \cdots \otimes a_k$ is a synchronized letter in $\mathbb{A}_\perp^k$, we denote by $\tau(\bar{a})$ the set $\{i \in \{1,\ldots,k\} \colon a_i = \perp\}$. A non-empty subset $\bar{A} \subseteq \mathbb{A}_\perp^k$ is said to be $\perp$-*consistent* if all the $\tau(\bar{a})$ ($\bar{a} \in \bar{A}$) take the same value. If that is the case, we let $\tau(\bar{A}) = \tau(\bar{a})$ (for any $\bar{a} \in \bar{A}$). Finally we say that a monomial $\bar{A}_0^*\bar{a}_1\bar{A}_1^* \ldots \bar{a}_n\bar{A}_n^*$ (over $\mathbb{A}_\perp^k$) is $\perp$-*consistent* if and only if every non-empty $\bar{A}_i$ is $\perp$-consistent and the sequence $\tau(\bar{A}_0), \tau(\bar{a}_1), \tau(\bar{A}_1), \ldots, \tau(\bar{a}_n), \tau(\bar{A}_n)$ is $\subseteq$-increasing (where the term $\tau(\bar{A}_i)$ is skipped if $\bar{A}_i = \emptyset$).

We denote by $\bar{\mathcal{P}}$ the set of all $\perp$-*consistent polynomials*, that is, of finite unions of $\perp$-consistent monomials. The following statement follows directly from this definition.

▶ **Lemma 20.** *Let $L = \bar{A}_0^*\bar{a}_1\bar{A}_1^* \ldots \bar{a}_n\bar{A}_n^*$ be an $\mathbb{A}_\perp^k$-monomial. Then $L \subseteq SW_k$ if and only if $L \in \bar{\mathcal{P}}$. Moreover, if L is $\perp$-consistent and $S \subseteq \{1,\ldots,k\}$, then $\pi_S(L)$ is a $\perp$-consistent monomial as well.*

We can now proceed with the proof of Theorem 7. We first show that a $\mathbf{\Sigma}_2[\sigma]$-definable k-ary relation R satisfies $L_R \in \bar{\mathcal{P}}$. Indeed, without loss of generality, R is defined by a $\mathbf{\Sigma}_2[\sigma]$-formula φ with free variables $S = \{z_1,\ldots,z_k\}$, of the form $\varphi = \exists x_1 \ldots \exists x_n\ \psi(x_1,\ldots,x_n,z_1,\ldots,z_k)$, with $\psi \in \mathbf{\Pi}_1[\sigma]$. In particular, $\|\psi\|$ is a $(n+k)$-ary relation and $R = \pi_S(\|\psi\|)$. Lemmas 20 and 21 therefore establish that $L_R \in \bar{\mathcal{P}}$.

▶ **Lemma 21.** *If $R \in \|\mathbf{\Pi}_1[\sigma]\|$ then $L_R \in \bar{\mathcal{P}}$.*

Proof. Since $R \in \|\mathbf{\Pi}_1[\sigma]\|$, it is the complement of a $\mathbf{\Sigma}_1[\sigma]$-definable relation. By Corollary 6, we have $L_R = SW_k \setminus {\uparrow_{\mathsf{s}}}\mathcal{S}$ for some finite set $\mathcal{S}$. Since $\bar{\mathcal{P}}$ is closed under intersection (see Remark 19), we only need to show that $SW_k \setminus {\uparrow_{\mathsf{s}}}\bar{w} \in \bar{\mathcal{P}}$ for a single synchronized word $\bar{w}$.

Let $\bar{T} = (T_1,\ldots,T_n) = \mathsf{type\text{-}seq}(\bar{w})$. By Lemma 1, we see that $\bar{u} \in L_R$ if and only either (1) $\bar{T} \not\sqsubseteq \mathsf{type\text{-}seq}(\bar{u})$, or (2) $\bar{T} \sqsubseteq \mathsf{type\text{-}seq}(\bar{u})$, with witness function t and $\bar{w}_i \not\sqsubseteq \bar{u}_{t(i)}$ for some $1 \leq i \leq n$ or, (3) again $\bar{T} \sqsubseteq \mathsf{type\text{-}seq}(\bar{u})$ with witness t, where $\bar{w}_i$ and $\bar{u}_{t(i)}$ do not have the same last letter for some i such that T_i is an end-type for $\bar{w}$.

The first condition means that $\bar{u} \in \bigcup K(\bar{T}')$, where the union runs over type sequences $\bar{T}'$ such that $\bar{T} \not\sqsubseteq \bar{T}'$. We saw in Section 2 that this union is in $\bar{\mathcal{P}}$. The second condition places $\bar{u}$ in $C_i = \mathbb{A}_{-,T_1}\mathbb{A}_{T_1}^* \cdots \mathbb{A}_{T_{i-2},T_{i-1}}\mathbb{A}_{T_{i-1}}^*\ L_i\ \mathbb{A}_{T_i,T_{i+1}}\mathbb{A}_{T_{i+1}}^* \cdots \mathbb{A}_{T_{n-1},T_n}\mathbb{A}_{T_n}^*$, where L_i is the set of words in $\mathbb{A}_{T_i-1T_i}\mathbb{A}_{T_i}^*$ that do not have $\bar{w}_i$ as a subword. Then L_i is $\mathbf{\Pi}_1[<]$-, and hence $\mathbf{\Sigma}_2[<]$-definable. As a consequence, $L_i \in \bar{\mathcal{P}}$ and, by associativity, $C_i \in \bar{\mathcal{P}}$. Finally, the third condition places $\bar{u}$ in $C_i' = \mathbb{A}_{-,T_1}\mathbb{A}_{T_1}^* \cdots \mathbb{A}_{T_{i-2},T_{i-1}}\mathbb{A}_{T_{i-1}}^*\ L_i'\ \mathbb{A}_{T_i,T_{i+1}}\mathbb{A}_{T_{i+1}}^* \cdots \mathbb{A}_{T_{n-1},T_n}\mathbb{A}_{T_n}^*$, where $L_i' = (\mathbb{A}_{T_i} \setminus \mathbb{A}_{T_{i-1}})\mathbb{A}_{T_i}^* \cap \mathbb{A}_{T_i}^* B_i$, with B_i the set of letters of $\mathbb{A}_{T_i}$ different from the last letter of $\bar{w}_i$. Here too, $L_i' \in \bar{\mathcal{P}}$ and hence $C_i' \in \bar{\mathcal{P}}$. ◀

The following lemma then concludes the proof of Theorem 7.

▶ **Lemma 22.** *If R is a relation such that L_R is a $\perp$-consistent polynomial, then $R \in \|\mathbf{\Sigma}_2[\sigma]\|$.*

Proof. By definition of $\bar{\mathcal{P}}$, the proof reduces to the case where L_R is a $\perp$-consistent monomial, say $L_R = \bar{A}_0^*\bar{a}_1\bar{A}_1^* \cdots \bar{a}_n\bar{A}_n^*$. We now construct a $\mathbf{\Sigma}_2[\sigma]$-formula φ, with set of free variables $Z = \{z_1,\ldots,z_k\}$, which defines R.

Let $w_1,\ldots,w_k \in \mathbb{A}^*$ be such that $\bar{w} = w_1 \otimes \cdots \otimes w_k = \bar{a}_1 \cdots \bar{a}_n$ (they exist due to $\perp$-consistency). Let $X = \{x_{i,j} : 1 \leq i \leq k, 1 \leq j \leq |w_i|\}$ and $Y = \{y_1,\ldots,y_k\}$ be sets of variables. We first let $\psi_1(X,Z)$ be the conjunction of the following formulæ for $1 \leq i \leq k$:

(1) for every $1 \leq j < |w_i|$: $x_{i,j} \prec x_{i,j+1}$; (2) $x_{i,|w_i|} \preceq z_i$; (3) for every $1 \leq j \leq |w_i|$: $\ell_{w_i[j]}(x_{i,j})$; (4) for every $1 \leq i' \leq k$ and $j \leq \min\{|w_i|, |w_{i'}|\}$: $\mathsf{eq}(x_{i,j}, x_{i',j})$. Notice that $\bar{u} \in SW_k$ satisfies $\exists X\ \psi_1(X, Z)$ if and only if $\bar{w}$ is a subword of $\bar{x}$ with witness function given by $p(j) = |x_{h,j}|$ for any $1 \leq h \leq k$.

The variables in Y are meant to represent the k components of a prefix of $\bar{u}$, which is expressed by $\psi_2(X, Y)$, the disjunction over all subsets H of $\{1, \ldots, k\}$ (H represents the components of $\bar{u}$ which are shorter than that prefix) of the formulæ

$$\bigwedge_{h \in H} (y_h \preceq z_h \wedge \mathsf{eq}(y_h, z_h)) \wedge \bigwedge_{h, i \notin H} (y_h \prec z_h) \wedge \mathsf{eq}(y_h, y_i)) \wedge \bigwedge_{h \in H, i \notin H} \exists r (r \preceq y_i \wedge \mathsf{eq}(r, y_h)).$$

Next, for $\bar{a} \in \mathbb{A}_{\perp}^{k}$ and $\bar{A} \subseteq \mathbb{A}_{\perp}^{k}$, and recalling that $\tau(\bar{a}) = \{h : \pi_h(\bar{a}) = \perp\}$, we define $\psi_{\bar{a}}(Y) = \bigwedge_{h \notin \tau(\bar{a})} \ell_{\pi_h(\bar{a})}(y_h)$ and $\psi_{\bar{A}}(Y) = \bigvee_{\bar{a} \in \bar{A}} \psi_{\bar{a}}(Y)$. Once $\bar{y}$ is a prefix of $\bar{u}$ and $\bar{w}$ is a subword of $\bar{u}$ with witness function p, if for some $1 \leq j \leq n$, we have $|\bar{y}| = p(j)$, then $\bar{y}$ satisfies $\psi_{\bar{a}_j}$. We now only need to verify that if $|\bar{y}|$ sits between $p(j)$ and $p(j+1)$ (for some $0 \leq j \leq n$), then $\bar{y}$ satisfies $\psi_{\bar{A}_j}$. This is done by the formula $\psi_3(X, Y) = \bigwedge_{j=0}^{n} \chi_j$, where $\chi_0(X, Y) = \left(\bigwedge_{h \notin \tau(\bar{A}_0)} y_h \prec x_{h,1}\right) \to \psi_{\bar{A}_0}(Y)$, $\chi_n(X, Y) = \left(\bigwedge_{h \notin \tau(\bar{A}_n)} x_{h,n} \prec y_h\right) \to \psi_{\bar{A}_n}(Y)$, and for every $0 < j < n$,

$$\chi_j(X, Y) = \left(\bigwedge_{h \notin \tau(\bar{A}_j)} (x_{h,j} \prec y_h) \wedge (y_h \prec x_{h,j+1}) \right) \to \psi_{\bar{A}_j}(Y).$$

Finally, R is defined by the $\mathbf{\Sigma}_2[\sigma]$ formula $\varphi(Z) = \exists X \psi_1(X, Z) \wedge \forall Y\ (\psi_2(X, Y) \wedge \psi_3(X, Y))$. ◀

References

1 Parosh Aziz Abdulla, Bengt Jonnson, Marcus Nilsson, and Mayank Saksena. A survey of regular model checking. In *International Conference on Concurrency Theory (CONCUR)*, pages 35–48, 2003.

2 Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. Expressive Languages for Path Queries over Graph-Structured Data. *ACM Transactions on Database Systems (TODS)*, 37(4):31, 2012. doi:10.1145/2389241.2389250.

3 Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. Definable relations and first-order query languages over strings. *Journal of the ACM*, 50(5):694–751, 2003. doi:10.1145/876638.876642.

4 Jean Berstel. *Transductions and Context-Free Languages.* B. G. Teubner, 1979.

5 Achim Blumensath and Erich Grädel. Automatic Structures. In *Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 51–62. IEEE Computer Society Press, 2000. doi:10.1109/LICS.2000.855755.

6 Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular Model Checking. In *International Conference on Computer Aided Verification (CAV)*, pages 403–418. Springer, 2000.

7 J. Richard Büchi. On a Decision Method in Restricted Second-Order Arithmetic. In *Proc. Int. Congr. for Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford Univ. Press, 1962.

8 Christian Choffrut. Relations over Words and Logic: A Chronology. *Bulletin of the EATCS*, 89:159–163, 2006.

9 Samuel Eilenberg, Calvin C. Elgot, and John C. Shepherdson. Sets recognized by n-tape automata. *Journal of Algebra*, 13(4):447–464, 1969. doi:10.1016/0021-8693(69)90107-0.

10 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society (3)*, 2(7):326–336, 1952. `doi:10.1112/plms/s3-2.1.326`.

11 M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, Reading, MA, 1983. Reprinted by *Cambridge University Press*, 1997.

12 Robert McNaughton and Seymour Papert. *Counter-Free Automata*. The MIT Press, Cambridge, Mass., 1971.

13 Jean-Éric Pin. *Varieties of Formal Languages*. North Oxford Academic, London, 1986.

14 Jean-Éric Pin and Howard Straubing. Monoids of upper triangular matrices. In *Colloquia Mathematica Societatis Janos Bolyai*, pages 259–272, 1981.

15 Jean-Éric Pin and Pascal Weil. Polynomial Closure and Unambiguous Product. *Theory of Computing Systems*, 30(4):383–422, 1997.

16 Thomas Place and Marc Zeitoun. Going Higher in the First-Order Quantifier Alternation Hierarchy on Words. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 342–353, 2014. `doi:10.1007/978-3-662-43951-7_29`.

17 Thomas Place and Marc Zeitoun. The tale of the quantifier alternation hierarchy of first-order logic over words. *SIGLOG News*, 2(3):4–17, 2015. `doi:10.1145/2815493.2815495`.

18 Thomas Place and Marc Zeitoun. Concatenation Hierarchies: New Bottle, Old Wine. In *International Computer Science Symposium in Russia (CSR)*, pages 25–37, 2017. `doi:10.1007/978-3-319-58747-9_5`.

19 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

20 Imre Simon. Piecewise testable events. In H. Barkhage, editor, *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 22–23, 1975*, volume 33 of *LNCS*, pages 214–222. Springer, 1975.

21 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, Basel and Berlin, 1994.

22 Howard Straubing and Pascal Weil. An introduction to automata theory. In Deepak D'Souza and Priti Shankar, editors, *Modern applications of automata theory*, volume 2 of *I.I.Sc. Monographs*, pages 3–43. World Scientific, 2012.

Two variable fragment of Term Modal Logic

Anantha Padmanabha
Institute of Mathematical Sciences, HBNI, Chennai, India
ananthap@imsc.res.in

R. Ramanujam
Institute of Mathematical Sciences, HBNI, Chennai, India
jam@imsc.res.in

Abstract

Term modal logics (TML) are modal logics with unboundedly many modalities, with quantification over modal indices, so that we can have formulas of the form $\exists y \forall x\ (\Box_x P(x, y) \supset \Diamond_y P(y, x))$. Like First order modal logic, TML is also "notoriously" undecidable, in the sense that even very simple fragments are undecidable. In this paper, we show the decidability of one interesting fragment, that of two variable TML. This is in contrast to two-variable First order modal logic, which is undecidable.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Term modal logic, satisfiability problem, two variable fragment, decidability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.30

Related Version An extended version is available on arXiv [16], https://arxiv.org/abs/1904.10260

Acknowledgements We thank Kamal Lodaya, Sreejith and Yanjing Wang for the insightful discussions. We also thank the anonymous referees for their comments and suggestions that helped better present the paper.

1 Introduction

Propositional multi-modal logics (ML) are extensively used in many areas of computer science and artifical intelligence ([2, 9]). ML is built upon propositional logic by adding modal operators $\Box_i$ and $\Diamond_i$ for every index i in a fixed finite set Ag which is often interpreted as a set of agents (or reasoners). Typically, the satisfiability problem is decidable for most instances of ML.

A natural question arises when we wish the set of modalities to be unbounded. This is motivated by a range of applications such as client-server systems, dynamic networks of processes, games with unboundedly many players, etc. In such systems, the number of agents is not fixed a priori. For some cases, the agent set can vary not only across models, but also from state to state (ex. when new clients enter the system or old clients exit the system).

Term Modal logic (TML) introduced by Fitting, Voronkov and Thalmann [6] addresses this requirement. TML is built upon first order logic, but the variables now range over modalities: so we can index the modality by terms ($\Box_x \alpha$) and these terms can be quantified over. State assertions describe properties of these "agents". Thus we can write formulas of the form: $\forall x(\Box_x P(x) \supset \exists y\ \Box_y \Diamond_x R(x, y))$. In [15] we have advocated PTML, the propositional fragment of TML, as a suitable logical language for reasoning about systems with unboundedly many agents. TML has been studied in dynamic epistemic contexts in [11] and in modelling situations where the identity of agents is not common knowledge among the agents [22].

The following examples illustrate the flavour of properties that can be expressed in TML.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 30; pp. 30:1–30:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- For every agent x there is some agent y such that $P(x,y)$ holds at all x-successors or there is some y-successor where $\neg P(x,y)$ holds.
  $\forall x \exists y\ \big(\Box_x P(x,y) \vee \Diamond_y(\neg P(x,y))\big)$
- Every agent of type A has a successor where some agent of type B exists.
  $\forall x\big(A(x) \supset \Diamond_x \exists y\ B(y)\big)$.
- There is some agent x such that for all agents y if there are no y successors then in all successors of x, there is a y successor.
  $\exists x \forall y\ \big(\Box_y \bot \supset \Box_x \Diamond_y \top\big)$.

Since TML contains first order logic, its satisfiability is clearly undecidable. We are then led to ask: can we build term modal logics over decidable fragments of first order logic? Natural candidates are the monadic fragment, the two-variable fragment and the guarded fragment [13, 1].

TML itself can be seen as a fragment of first order modal logic (FOML) [5] which is built upon first order logic by adding modal operators. There is a natural translation of TML into FOML by inductively translating $\Box_x \alpha$ into $\Box(P(x) \supset \alpha)$ and $\Diamond_x \alpha$ into $\Diamond(P(x) \wedge \alpha)$ to get an equi-satisfiable formula, where P is a new unary predicate. Sadly, this does not help much, since FOML is notorious for undecidability. The modal extension of many simple decidable fragments of first order logic become undecidable. For instance, the monadic fragment[12] or the two variable fragment [10] of FOML are undecidable. In fact FOML with two variables and a single unary predicate is already undecidable [18]. Analogously, in [15] we show that the satisfiability problem for TML is undecidable even when the atoms are restricted to propositions. In the presence of equality (even without propositions), this result can be further strengthened to show "Trakhtenbrot" like theorem of mutual recursive inseparability.

On the other hand, as we show in [15], the monodic fragment of PTML (the propositional fragment) is decidable (a formula φ is monodic if each of its modal subformulas of the form $\Box_x \psi$ or $\Diamond_x \psi$ has a restriction that the free variables of ψ is contained in $\{x\}$). Further, via the FOML translation above, we can show that the monodic restriction of TML based on the guarded fragment of first order logic and monadic first order logic are decidable [23].

In a different direction, Wang ([21]) considered a fragment of FOML in which modalities and quantifiers are bound to each other. In particular he considered the fragment with $\exists\Box$ and showed it to be decidable in PSPACE. In [17] it is proved that this technique of bundling quantifiers and modalities gives us interesting decidable fragments of FOML, and as a corollary, the bundled fragment of TML is decidable where quantifiers and modalities always occur in bundled form: $\forall x \Box_x \alpha, \exists x \Box_x \alpha$ and their duals. However, more general bundled fragments of TML (such as those based on the guarded fragment of first order logic) have been shown to be decidable by Orlandelli and Corsi ([14]), and by Shtakser ([19]). From all these results, it is clear that the one variable fragment of TML is decidable, and that the three variable fragment of PTML is undecidable.

In this paper, we show that the two variable fragment of TML (TML^2) is decidable. This is in contrast with FOML, for which the two variable fragment is undecidable [10]. Quoting Wolter and Zakharyaschev from [23], where they discuss the root of undecidability of FOML fragments:

> All undecidability proofs of modal predicate logics exploit formulas of the form $\Box\ \psi(x,y)$ in which the necessity operator applies to subformulas of more than one free variable; in fact, such formulas play an essential role in the reduction of undecidable problems to those fragments ...

Note that this is not expressible in TML^2 where there is no "free" modality; every modality is bound an index (x or y). With a third variable z, we could indeed encode $\Box P(x,y)$ as $\forall z \Box_z P(x,y)$, but we do not have it. The decidability of the two variable fragment of TML, *without constants or equality*, hinges crucially on this lack of expressiveness. Thus, TML^2 provides a decidable fragment of FOML^2. From FO^2 view point, Gradel and Otto[8] show that most of the natural extensions of FO^2 (like transitive closure, lfp) are undecidable except for the counting quantifiers. In this sense, 2-variable TML can be seen as another rare extension of FO^2 that still remains decidable. Note that in this paper we consider the two variable fragment of TML without the bundling or guarded or monodic restriction. Also, there is no natural translation of two variable TML to any known decidable fragment of FO such as the two variable fragment of FO with 2 equivalence relations etc (cf [20]).

Thus, the contribution of this paper is technical, mainly in the identification of a decidable fragment of TML. As is standard with two variable logics, we first introduce a normal form which is a combination of Fine's normal form for modal logics ([4]) and the Scott normal form ([7]) for FO^2. We then prove a bounded agent property using an argument that can be construed as modal depth induction over the "classical" bounded model construction for FO^2.

2 TML syntax and semantics

We consider relational vocabulary with no constants or function symbols, and without equality.

▶ **Definition 1** (TML syntax). *Given a countable set of variables* Var *and a countable set of predicate symbols $\mathcal{P}$, the syntax of* TML *is defined as follows:*

$$\varphi ::= P(\overline{x}) \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \exists x\ \varphi \mid \forall x\ \varphi \mid \Box_x\varphi \mid \Diamond_x\varphi$$

where $x \in$ Var, $\overline{x}$ is a vector of length n over Var and $P \in \mathcal{P}$ of arity n.

The *free* and *bound* occurrences of variables are defined as in FO with $\mathsf{Fv}(\Box_x\varphi) = \mathsf{Fv}(\varphi) \cup \{x\}$. We write $\varphi(\overline{x})$ if all the free variables in φ are included in $\overline{x}$. Given a TML formula φ and $x, y \in \mathsf{Var}$, if $y \notin \mathsf{Fv}(\varphi)$ then we write $\varphi[y/x]$ for the formula obtained by replacing every occurrence of x by y in φ. A formula φ is called a *sentence* if $\mathsf{Fv}(\varphi) = \emptyset$. The notion of modal depth of a formula φ (denoted by $\mathsf{md}(\varphi)$) is also standard, which is simply the maximum number of nested modalities occurring in φ. The length of a formula φ is denoted by $|\varphi|$ and is simply the number of symbols occurring in φ.

In the semantics, the number of accessibility relations is not fixed, but specified along with the structure. Thus the Kripke frame for TML is given by (W, D, R) where W is a set of worlds, D is the potential set of agents and $R \subseteq (W \times D \times W)$. The agent dynamics is captured by a function ($\delta : W \to 2^D$ below) that specifies, at any world w, the set of agents *live* (or meaningful) at w. The condition that whenever $(u, d, v) \in R$, we have that $d \in \delta(u)$ ensures only an agent alive at u can consider v accessible.

A *monotonicity* condition is imposed on the accessibility relation as well: whenever $(u, d, v) \in R$, we have that $\delta(u) \subseteq \delta(v)$. This is required to handle interpretations of free variables (cf [3, 6, 5]). Hence the models are called "increasing agent" models.

▶ **Definition 2** (TML structure). *An increasing agent model for* TML *is defined as the tuple $M = (W, D, \delta, R, \rho)$ where W is a non-empty countable set of worlds, D is a non-empty countable set of agents, $R \subseteq (W \times D \times W)$ and $\delta : W \to 2^D$. The map δ assigns to each $w \in W$ a non-empty local domain such that whenever $(w, d, v) \in R$ we have $d \in \delta(w) \subseteq \delta(v)$ and $\rho : (W \times \mathcal{P}) \to \bigcup_{n\in\omega} 2^{D^n}$ is the valuation function where for all $P \in \mathcal{P}$ of arity n we have $\rho(w, P) \subseteq [\delta(w)]^n$.*

For a given model M, we use $W^M, D^M, \delta^M, R^M, \rho^M$ to refer to the corresponding components. We drop the superscript when M is clear from the context. We often write D_w for $\delta(w)$. A *constant agent* model is one where $D_w = D$ for all $w \in W$. To interpret free variables, we need a variable assignment $\sigma : \mathsf{Var} \to D$. Call σ *relevant* at $w \in W$ if $\sigma(x) \in \delta(w)$ for all $x \in \mathsf{Var}$. The increasing agent condition ensures that if σ is relevant at w and $(w, d, v) \in R$ then σ is relevant at v as well. In a constant agent model, every assignment σ is relevant at all the worlds.

▶ **Definition 3** (TML semantics). *Given a* TML *structure $M = (W, D, \delta, R, \rho)$ and a* TML *formula φ, for all $w \in W$ and σ relevant at w, define $M, w, \sigma \vDash \varphi$ inductively as follows:*

$M, w, \sigma \vDash P(x_1, \ldots, x_n)$	$\Leftrightarrow$	$(\sigma(x_1), \ldots, \sigma(x_n)) \in \rho(w, P)$
$M, w, \sigma \vDash \neg\varphi$	$\Leftrightarrow$	$M, w, \sigma \nvDash \varphi$
$M, w, \sigma \vDash (\varphi \wedge \psi)$	$\Leftrightarrow$	$M, w, \sigma \vDash \varphi$ *and* $M, w, \sigma \vDash \psi$
$M, w, \sigma \vDash \exists x\ \varphi$	$\Leftrightarrow$	*there is some* $d \in \delta(w)$ *such that* $M,\ w,\ \sigma_{[x \mapsto d]} \vDash \varphi$
$M, w, \sigma \vDash \Box_x\ \varphi$	$\Leftrightarrow$	$M, v, \sigma \vDash \varphi$ *for all* v *s.t.* $(w, \sigma(x), v) \in R$

where $\sigma_{[x \mapsto d]}$ denotes another assignment that is the same as σ except for mapping x to d.

The semantics for $\varphi \vee \psi, \forall x\ \varphi$ and $\Diamond_x\ \varphi$ are defined analogously. Note that $M, w, \sigma \vDash \varphi$ is inductively defined only when σ is relevant at w. We often abuse notation and say “for all w and for all interpretations σ”, when we mean “for all w and for all interpretations σ relevant at w” (and we will ensure that relevant σ are used in proofs). In general, when considering the truth of φ in a model, it suffices to consider $\sigma : \mathsf{Fv}(\varphi) \mapsto D$, assignment restricted to the variables occurring free in φ. When $\mathsf{Fv}(\varphi) \subseteq \{x_1, \ldots, x_n\}$ and $\overline{d} \in [D_w]^n$ is a vector of length n over D_w, we write $M, w \vDash \varphi[\overline{d}]$ to denote $M, w, \sigma \vDash \varphi(\overline{x})$ where for all $i \leq n,\ \sigma(x_i) = d_i$. When φ is a sentence, we simply write $M, w \models \varphi$. A formula φ is *valid*, if φ is true in all models M at all w for all interpretations σ (relevant at w). A formula φ is *satisfiable* if $\neg\varphi$ is not valid.

Now we take up the satisfiability problem which is the central theme of this paper. First we observe that the satisfiability problem is equally hard for constant and increasing agent models for TML.

▶ **Proposition 4.** *For any* TML *formula φ, there is a corresponding formula $\hat{\varphi} \in$* TML *such that φ is satisfiable in an increasing agent model with agent set D iff $\hat{\varphi}$ is satisfiable in a constant agent model with agent set D.*

To see why the proposition is true, if φ is satisfiable in an increasing agent model, then we can turn the model into constant agent model as follows. We introduce a new unary predicate E and ensure that $E(d)$ is true at w if d is a member of $\delta(w)$ in the given increasing agent model. But now, all quantifications have to be relativized with respect to the new predicate E. Thus, the syntactic translation is defined as follows:

▶ **Definition 5.** *Let φ be any* TML *formula amd let E be a new unary predicate not occurring in φ. The translation is defined inductively as follows:*

- $\mathit{Tr}_1(P(x_1, \ldots, x_n)) = P(x_1, \ldots, x_n)$
- $\mathit{Tr}_1(\neg\varphi) = \neg\mathit{Tr}_1(\varphi)$ *and* $\mathit{Tr}_1(\varphi \wedge \psi) = \mathit{Tr}_1(\varphi) \wedge \mathit{Tr}_1(\psi)$
- $\mathit{Tr}_1(\Box_x\varphi) = \Box_x(\mathit{Tr}_1(\varphi))$
- $\mathit{Tr}_1(\exists x\ \varphi) = \exists x\ (E(x) \wedge \mathit{Tr}_1(\varphi))$

With this translation, we also need to ensure that the predicate E respects monotonicity. Hence we have $\gamma_\varphi = \bigwedge\limits_{i+j \leq \mathsf{md}(\varphi)} (\forall y \Box_y)^i \big(\forall x\ E(x) \supset (\forall y \Box_y)^j E(x)\big)$. Now, we can prove that φ is satisfiable in an increasing model M with agent set D iff $\mathsf{Tr}_1(\varphi) \wedge \gamma_\varphi$ is satisfiable in a constant agent model M' with agent set D. This translation is similar in approach to the one for FOML[23]. The formal proof of Prop. 4 is provided in [16].

The propositional term modal logic (PTML) is a fragment of TML where the atoms are restricted to propositions. Note that the variables still appear as index of modalities. For PTML, the valuation function can be simply written as $\rho : W \mapsto 2^{\mathcal{P}}$ where $\mathcal{P}$ is the set of propositions. Now we prove that the satisfiability problem for PTML is as hard as that for TML.

▶ **Proposition 6.** *Any* TML *formula* φ *has a corresponding formula* $\hat{\varphi} \in$ PTML*, such that* φ *is satisfiable in an increasing (constant) agent model with agent set* D *iff* $\hat{\varphi}$ *is satisfiable in an increasing (constant) agent model with agent set* D*.*

The reduction is based on the translation of an arbitrary atomic predicate $P(x_1, \ldots, x_n)$ to $\Diamond_{x_1} \ldots \Diamond_{x_n} p$ where p is a new proposition which represents the predicate P. However, this cannot be used always[1]. Thus, we use a new proposition q, to distinguish the "real worlds" from the ones that are added because of the translation. But now, the modal formulas have to be relativized with respect to the proposition q. The formal translation is given as follows:

▶ **Definition 7.** *Let* φ *be any* TML *formula where* $P_1, \ldots, P_k$ *are the predicates that occur in* φ*. Let* $\{p_1, \ldots, p_n\} \cup \{q\}$ *be a new set of propositions not occurring in* φ*. The translation with respect to* q *is defined inductively as follows:*

- $Tr_2(P_i(x_1, \ldots, x_n); q) = \Diamond_{x_1}(\neg q \wedge \Diamond_{x_2}(\ldots \neg q \wedge \Diamond_{x_n}(\neg q \wedge p_i)\ldots))$
- $Tr_2(\neg\varphi; q) = \neg Tr_2(\varphi; q)$ *and* $Tr_2(\varphi \wedge \psi; q) = Tr_2(\varphi; q) \wedge Tr_2(\psi; q)$
- $Tr_2(\Box_x \varphi; q) = \Box_x(q \supset Tr_2(\varphi; q))$
- $Tr_2(\exists x\ \varphi; q) = \exists x\ Tr_2(\varphi; q)$

Using this translation, we can prove that φ is satisfiable iff $q \wedge \mathsf{Tr}_2(\varphi; q)$ is satisfiable, over the same D. The proof details of Prop 6 are given in [16].

3 Two variable fragment

Note that all the examples discussed in the introduction section use only 2 variables. Thus, TML can express interesting properties even when restricted to two variables. We now consider the satisfiability problem of TML^2. The translation in Def. 7 preserves the number of variables. Therefore it suffices to consider the satisfiability problem for the two variable fragment of PTML.

Let PTML^2 denote the two variable fragment of PTML. We first consider a normal form for the logic. In [4], Fine introduces a normal form for propositional modal logics which is a disjunctive normal form (DNF) with every clause of the form $(\bigwedge\limits_i (s_i) \wedge \Box\alpha \wedge \bigwedge\limits_j \Diamond\beta_j)$ where s_i are literals and α, β_j are again in the normal form. For FO^2, we have Scott normal form [7] where every FO^2 sentence has an equi-satisfiable formula of the form $\forall x \forall y\ \varphi \wedge \bigwedge\limits_i \forall x \exists y\ \psi_i$ where φ and ψ_i are all quantifier free. For PTML^2, we introduce a combination of these two

[1] for instance, this translation will not work for the formula $\exists x\ P(x) \wedge \forall y\ \Box_y \bot$

normal forms, which we call the *Fine Scott Normal form* given by a DNF, where every clause is of the form:

$$\bigwedge_{i\leq a} s_i \wedge \bigwedge_{z\in\{x,y\}} (\Box_z\alpha \wedge \bigwedge_{j\leq m_z} \Diamond_z\beta_j) \wedge \bigwedge_{z\in\{x,y\}} (\forall z\ \gamma \wedge \bigwedge_{k\leq n_z} \exists z\ \delta_k) \wedge \forall x\forall y\ \varphi \wedge \bigwedge_{l\leq b} \forall x\exists y\ \psi_l$$

where $a, m_x, m_y, n_x, n_y, b \geq 0$ and s_i denotes literals. Further, α, β_j are recursively in the normal form and $\gamma, \delta_k, \varphi, \psi_l$ do not have quantifiers at the outermost level and all modal subformulas occurring in these formulas are (recursively) in the normal form. The normal form is formally defined in the next subsection.

Note that the first two conjuncts mimic the modal normal form and the last two conjuncts mimic the FO^2 normal form. The additional conjuncts handle the intermediate step where only one of the variable is quantified and the other is free.

We now formally define the normal form and prove that every PTML^2 formula has a corresponding equi-satisfiable formula in the normal form. After this we prove the bounded agent property for formulas in the normal form using an inductive FO^2 type model construction.

3.1 Normal form

We use $\{x, y\} \subseteq \mathsf{Var}$ as the two variables of PTML^2. We use z to refer to either x or y and refer to variables z_1, z_2 to indicate the variables x, y in either order. We use Δ_z to denote any modal operator $\Delta \in \{\Box, \Diamond\}$ and $z \in \{x, y\}$. A literal is either a proposition or its negation. Also, we assume that the formulas are given in negation normal form(NNF) where the negations are pushed in to the literals.

▶ **Definition 8** (FSNF normal form). *We define the following terms to introduce the* Fine Scott normal form *(FSNF) for* PTML^2*:*

- *A formula φ is a module if φ is a literal or φ is of the form $\Delta_z\alpha$.*
- *For any formula φ, the outer most components of φ given by $\mathsf{C}(\varphi)$ is defined inductively where for any φ which is a module, $\mathsf{C}(\varphi) = \{\varphi\}$ and $\mathsf{C}(Qz\ \varphi) = \{Qz\ \varphi\}$ where $z \in \{x, y\}$ and $Q \in \{\forall, \exists\}$. Finally $\mathsf{C}(\varphi \odot \psi) = \mathsf{C}(\varphi) \cup \mathsf{C}(\psi)$ where $\odot \in \{\wedge, \vee\}$.*
- *A formula φ is quantifier-safe if every $\psi \in \mathsf{C}(\varphi)$ is a module.*
- *We define Fine Scott normal form(FSNF) normal form (*DNF *and conjunctions) inductively as follows:*
  - *Any conjunction of literals is an FSNF conjunction.*
  - *φ is said to be in FSNF* DNF *if φ is a disjunction where every clause is an FSNF conjunction.*
  - *Suppose φ is quantifier-safe and for every $\Delta_z\psi \in \mathsf{C}(\varphi)$ if ψ is in FSNF* DNF *normal form then we call φ a quantifier-safe normal formula.*
  - *Let $a, b, m_x, m_y, n_x, n_y \geq 0$.*
    Suppose $s_1, \ldots, s_a$ are literals, $\alpha^x, \alpha^y, \beta^x_1, \ldots, \beta^x_{m_x}, \beta^y_1, \ldots, \beta^y_{m_y}$ are formulas in FSNF DNF *and $\gamma^x, \gamma^y, \delta^x_1, \ldots, \delta^x_{n_x}, \delta^y_1, \ldots, \delta^y_{n_y}, \varphi, \psi_1, \ldots, \psi_b$ are quantifier-safe normal formulas then:*

$$\bigwedge_{i\leq a} s_i \wedge \bigwedge_{z\in\{x,y\}} (\Box_z\alpha^z \wedge \bigwedge_{j\leq m_z} \Diamond_z\beta^z_j) \wedge \bigwedge_{z_1\in\{x,y\}} (\forall z_2\ \gamma^{z_1} \wedge \bigwedge_{k\leq n_z} \exists z_2\ \delta^{z_1}_k) \wedge \forall x\forall y\ \varphi \wedge \bigwedge_{l\leq b} \forall x\exists y\ \psi_l$$

is an FSNF conjunction.

Quantifier-safe formulas are those in which no quantifiers occur outside the scope of modalities. Note that the superscripts in α^x, α^y etc only indicate which variable the formula is associated with, so that it simplifies the notation. For instance, α^x does not say anything about the free variables in α^x. In fact there is no restriction on free variables in any of these formulas.

Further, note that by setting the appropriate indices to 0, we can have FSNF conjunctions where one or more of the components corresponding to $s_i, \beta^x, \beta^y, \delta^x, \delta^y, \psi_l$ are absent. We also consider the conjunctions where one or more of the components corresponding to $\Box_x\alpha^x, \Box_y\alpha^y, \varphi$ are also absent. As we will see in the next lemma, for any sentence $\varphi \in \mathsf{PTML}^2$, we can obtain an equi-satisfiable sentence, which at the outer most level, is a DNF where every clause is of the form $\bigwedge_{i\leq a} s_i \ \wedge\ \forall x\forall y\ \varphi \wedge \bigwedge_{l\leq b} \forall x\exists y\ \psi_l$.

▶ **Lemma 9.** *For every formula $\varphi \in \mathsf{PTML}^2$ there is a corresponding formula ψ in FSNF* DNF *such that φ and ψ are equi-satisfiable.*

Proof (Sketch). To get the formula in the normal form, we introduce some new unary predicates in the intermediate steps and finally get rid of them using the translation in Def. 7. The proof essentially follows that of reducing an FO^2 formula into its equi-satisfiable Scott normal form.

For the given formula φ, first observe that we can get an equivalent DNF over $\mathsf{C}(\varphi)$ using propositional validities. If φ is modal free, then we can simply ignore the quantifiers, since valuations of propositions do not depend on the quantifiers and the agent set is always non-empty. Thus we get a propositional DNF by erasing the quantifiers and this is in the required form.

If φ contains modal formulas, then we need to reduce every clause of the DNF to an FSNF conjunction. We first translate the formulas at the outer most level to the required form. This is the classical Scott-normal form construction which can be obtained by introducing new unary predicates appropriately to get rid of the nested quantifiers at the outer most level. Then, using the translation in Def 7, we get an equi-satisfiable PTML formula after replacing the newly introduced unary predicates by corresponding propositional translations. Further, replace conjuncts of the form $\Box_z\alpha$ and $\Box_z\beta$ by $\Box_z(\alpha \wedge \beta)$ for $z \in \{x, y\}$ to obtain the resulting formula which has at most one subformula of the from $\Box_x\alpha^x$ and $\Box_y\alpha^y$.

Note that after this translation, the resulting formula is in the required form at the outermost level. We now only need to repeat the entire process for every sub-formula inside the scope of modalities. The lemma is formally proved in [16]. ◀

Since we repeatedly convert the formula into DNF (inside the scope of every modality), if we start with a formula of length n, the final translated formula has length $2^{O(n^2)}$. However, observe that the number of modules in the translated formula is linear in the size of the given formula φ. Furthermore, the given formula is satisfiable in a model M iff the translation is satisfiable in M with appropriate modification of the ρ (valuation function).

3.2 Bounded agent property

Now we prove that any formula $\theta \in \mathsf{PTML}^2$ in FSNF DNF is satisfiable iff θ is satisfiable in a model M where the size of D is bounded. Note that for any PTML formula θ, if $M, w, \sigma \models \theta$ then $M^T, w, \sigma \models \theta$ where M^T is the standard tree unravelling of M with w as root [15]. Further, M^T can be restricted to be of height at most $\mathsf{md}(\theta)$. Hence, we restrict our attention to tree models of finite depth.

First we define the notion of types for agents at every world. In classical FO^2 the 2-types are defined on atomic predicates. In PTML^2 we need to define the types with respect to $\mathsf{modules}$. In any given tree model M rooted at r, for any $w \in W$ and $c, d \in D_w$ the $\mathsf{2\text{-}type}$ of (c, d) at w is simply the set of all $\mathsf{modules}$ that are true at w where the two variables are assigned c, d in either order. The $\mathsf{1\text{-}type}$ of c at w includes the set of all $\mathsf{modules}$ that are true at w when both x, y are assigned c. Further, for every non-root node w, suppose $(w' \xrightarrow{a} w)$ then the $\mathsf{1\text{-}type}$ of any $c \in D_w$ should capture how c behaves with respect to a and the $\mathsf{1\text{-}type}(w, c)$ should also include the information of how c acts with respect to d, for every $d \in D_w$. Thus the $\mathsf{1\text{-}type}$ of c at w is given by a 3-tuple where the first component is the set of all $\mathsf{modules}$ that are true when both x, y are assigned c, the second component captures how c behaves with respect to the incoming edge of w and the third component is a set of subsets of formulas such that for each $d \in D_w$ there is a corresponding subset of formulas capturing the $\mathsf{2\text{-}type}$ of c, d. To ensure that the type definition also carries the information of the height of the world w, if w is at height h then we restrict $\mathsf{1\text{-}type}$ and $\mathsf{2\text{-}type}$ at w to $\mathsf{modules}$ of modal depth at most $\mathsf{md}(\varphi) - h$.

For any formula φ, let $\mathsf{SF}(\varphi)$ be the set of all subformulas of φ closed under negation. We always assume[2] that $\top \in \mathsf{SF}(\varphi)$. Let $\mathsf{SF}^h(\varphi) \subseteq \mathsf{SF}(\varphi)$ be the set of all subformulas of modal depth at most $\mathsf{md}(\varphi) - h$. Thus we have $\mathsf{SF}(\varphi) = \mathsf{SF}^0(\varphi) \supseteq \mathsf{SF}^1(\varphi) \supseteq \ldots \supseteq \mathsf{SF}^{\mathsf{md}(\varphi)}(\varphi)$.

▶ **Definition 10** (PTML type). *For any* PTML^2 *formula* φ *and for any tree model* M *rooted at* r *with height at most* $\mathsf{md}(\varphi)$*, for all* $w \in W$ *at height* h*:*

- *For all* $c, d \in \delta(w)$*, define* 2-type$(w, c, d) = (\Gamma_{xy}; \Gamma_{yx})$ *where* $\Gamma_{xy} = \{\psi(x, y) \in \mathsf{SF}^h(\varphi) \mid M, w \models \psi(c, d)\}$ *and* $\Gamma_{yx} = \{\psi(x, y) \in \mathsf{SF}^h(\varphi) \mid M, w \models \psi(d, c)\}$.
- *If* w *is a non root node, (say* $w' \xrightarrow{a} w$*) then for all* $c \in \delta(w)$ *define* $1\text{-}\mathsf{type}(w, c) = (\Lambda_1; \Lambda_2; \Lambda_3)$ *where* $\Lambda_1 = 2\text{-}\mathsf{type}(w, c, c)$ *and* $\Lambda_2 = 2\text{-}\mathsf{type}(w, c, a)$ *and* $\Lambda_3 = \{2\text{-}\mathsf{type}(w, c, d) \mid d \in \delta(w)\}$.
- *For the root node* r*, for all* $c \in \delta(r)$ *define* $1\text{-}\mathsf{type}(w, c) = (\Lambda_1; \{\top\}; \Lambda_3)$ *where* $\Lambda_1 = 2\text{-}\mathsf{type}(w, c, c)$ *and* $\Lambda_3 = \{2\text{-}\mathsf{type}(w, c, d) \mid d \in \delta(w)\}$.

The second component of $\mathsf{1\text{-}type}(r, c)$ is added to maintain uniformity. For all $w \in W$ define $\mathsf{1\text{-}type}(w) = \{\mathsf{1\text{-}type}(w, c) \mid c \in D_w\}$ and $\mathsf{2\text{-}type}(w) = \{\mathsf{2\text{-}type}(w, c, d) \mid c, d \in D_w\}$. We use Λ, Π to represent elements of $\mathsf{1\text{-}type}(w)$ and Λ_1, Π_2 etc for the respective components.

If a formula θ is satisfiable in a tree model, the strategy is to inductively come up with bounded agent models for every subtree of the given tree (based on types), starting from leaves to the root. While doing this, when we add new type based agents to a world at height h, to maintain monotonicity, we need to propagate the newly added agents throughout its descendants. For this, we define the notion of extending any tree model by addition of some new set of agents.

Suppose in a tree model M, world w has local agent set D_w and we want to extend D_w to $D_w \cup C$, then first we have $\Omega : C \mapsto D_w$ which assigns every new agent to some already existing agent. The intended meaning is that the newly added agent $c \in C$ at w mimics the "type" of $\Omega(c)$. If w is a leaf node, we can simply extend $\delta(w)$ to $D_w \cup C$. If w is at some arbitrary height, along with *adding the new agents to the live agent set* to w, we also need to create successors for every $c \in C$, one for each successor subtree of $\Omega(c)$ and inductively add C to all the successor subtrees.

[2] Let p_0 be some proposition occurring in φ, then $\top$ is defined as $p_0 \vee \neg p_0$.

▶ **Definition 11** (Model extension). *Suppose M is a tree model rooted at r with finite agent set D and for every $w \in W$ let M^w be the subtree rooted at w. Let C be some finite set such that $C \cap D = \emptyset$ and for any $w \in W$ let $\Omega : C \mapsto D_w$ be a function mapping C to agent set live at w. Define the operation of "adding C to M^w guided by Ω" by induction on the height of w to obtain a new subtree rooted at w (denoted by $M^w_{(C,\Omega)}$ and the components denoted by δ', ρ' etc).*

- *If w is a leaf, then $M^w_{(C,\Omega)}$ is a tree with a single node w with new $\delta'(w) = \delta(w) \cup C$ and $\rho'(w) = \rho(w)$.*
- *If w is at height h then the new tree $M^w_{(C,\Omega)}$ is obtained from M^w rooted at w with new $\delta'(w) = \delta(w) \cup C$ and $\rho'(w) = \rho(w)$ and replacing all the subtrees M^u rooted at every successor u of w by $M^u_{(C,\Omega)}$. Furthermore, for every $c \in C$ and every $(w, \Omega(c), u) \in R$ create a new copy of $M^u_{(C,\Omega)}$ and rename its root as u^c and add an edge (w, c, u^c) to R'.*

Since we do not have *equality* in the language, this transformation will still continue to satisfy the same formulas.

▶ **Lemma 12.** *Let M be any tree model of finite depth rooted at r with finite agent set D and let $w \in W$. Let $M^w_{(C,\Omega)}$ (rooted at w) be an appropriate model extension of M^w (rooted at w). For any interpretation $\sigma : \mathsf{Var} \mapsto (C \cup D_w)$ let $\hat{\sigma} : \mathsf{Var} \mapsto D_w$ where $\hat{\sigma}(x) = \Omega(\sigma(x))$ if $\sigma(x) \in C$ and $\hat{\sigma}(x) = \sigma(x)$ if $\sigma(x) \in D_w$. Then for all $u \in W$ which is a descendant of w in M and for all $\sigma : \mathsf{Var} \mapsto (C \cup D_w)$ and for all* PTML *formula φ, we have $M^w_{(C,\Omega)}, u, \sigma \models \varphi$ iff $M, u, \hat{\sigma} \models \varphi$.*

To see why the lemma holds, first note that both models agree on literals since the valuation function remains the same. Further, since every new agent mimics some old agent, all the modal and the universal formulas continue to hold. Witnesses for $\exists$ formulas can still be picked from the old agent set (D_u). The lemma is formally proved in [16].

For any formula in the normal form, we use the same notations as in Def. 8. For a given formula $\theta \in \mathsf{PTML}^2$ in FSNF DNF form, let $\boldsymbol{\delta}^x_\theta = \{\exists y\ \delta^x \in \mathsf{SF}(\theta)\}$. Similarly we have $\boldsymbol{\delta}^y_\theta = \{\exists x\ \delta^y \in \mathsf{SF}(\theta)\}$ and $\boldsymbol{\psi}_\theta = \{\forall x \exists y\ \psi \in \mathsf{SF}(\varphi)\}$.

For any tree model M, let $\# \notin D$. For every $w \in W$ and for all $\exists y\ \delta \in \boldsymbol{\delta}^x_\theta$ let the function $g^w_\delta : D_w \mapsto D_w \cup \{\#\}$ be a mapping such that $M, w \models \delta(c, g^w_\delta(c))$ and $g^w_\delta(c) = \#$ only if there is no $d \in D_w$ such that $M, w \models \delta(c, d)$. Similarly for all $\exists x\ \delta \in \boldsymbol{\delta}^y_\theta$ let $h^w_\delta : D_w \mapsto D_w \cup \{\#\}$ such that $M, w \models \delta(h^w_\delta(c), h)$ and $h^w_\delta(c) = \#$ only if there is no $d \in D_w$ such that $M, w \models \delta(d, c)$. Again for all $\forall x \exists y\ \psi \in \boldsymbol{\psi}_\theta$ let $f^w_\psi : D_w \mapsto D_w \cup \{\#\}$ such that $M, w \models \psi(c, f^w_\psi(c))$ and $f^w_\psi(c) = \#$ only if there is no $d \in D_w$ such that $M, w \models \psi(c, d)$.

The functions g, h, f provide the witnesses at a world for every agent (if it exists) for the existential formulas respectively.

▶ **Theorem 13.** *Let $\theta \in \mathsf{PTML}^2$ be in an FSNF* DNF *sentence. Then θ is satisfiable iff θ is satisfiable in a model with bounded number of agents.*

Proof. It suffices to prove ($\Rightarrow$). Let M be a tree model of height at most $\mathsf{md}(\theta)$ rooted at r such that $M, r \models \theta$.

Let $E_\theta = \boldsymbol{\delta}^x_\theta \cup \boldsymbol{\delta}^y_\theta \cup \boldsymbol{\psi}_\theta$ and hence $|\mathsf{E}_\theta| \leq |\theta|$ (say q). Let $\mathsf{E}_\theta = \{\chi_1, \dots \chi_q\}$ be some enumeration. For every $w \in W$ and $a \in \delta(w)$ let $\mathsf{Wit}(a) = \{b_1 \dots b_q\}$ be the witnesses for a where $b_i = g^w_\delta(c)$ if χ_i is of the form $\exists y\ \delta \in \boldsymbol{\delta}^x_\theta$ (similarly $b_i = h^w_\delta(c)$ or $b_i = f^w_\psi(c)$ corresponding to χ_i of the from $\exists x\ \delta^y$ and $\forall x \exists y\ \psi$ respectively). If $b_i = \#$ then set $b_i = b$ for some arbitrary but fixed $b \in \delta(w)$.

For all $w \in W$ and $\Lambda \in$ 1-$\mathsf{type}(w)$ fix some $a^w_\Lambda \in \delta(w)$ such that 1-$\mathsf{type}(w, a^w_\Lambda) = \Lambda$. Furthermore, if c is the incoming edge of w and 1-$\mathsf{type}(w, c) = \Lambda$ then let $a^w_\Lambda = c$. Let $A^w = \{a^w_\Lambda \mid \Lambda \in$1-$\mathsf{type}(w)\}$.

Now we define the bounded agent model. For every $w \in W$ let M^w be the subtree model rooted at $w \in W$. For every such M^w, we define a corresponding *type based model* with respect to θ (denoted by T^w_θ with components denoted by $\delta^w_\theta, \rho^w_\theta$ etc) inductively as follows:

- If w is a leaf then T^w_θ is a tree with a single node w with $\delta^w_\theta(w) =$ 1-$\mathsf{type}(w) \times [1 \dots q] \times \{0, 1, 2\}$ and $\rho^w_\theta(w) = \rho(w)$.
- If w is at height h, T^w_θ is a tree rooted at w with $\delta^w_\theta(w) =$ 1-$\mathsf{type}(w) \times [1 \dots q] \times \{0, 1, 2\}$ and $\rho^w_\theta(w) = \rho(w)$.
  Before defining the successors of w in T^w_θ note that for every $(w, a, u) \in R$ we have T^u_θ which is the inductively constructed type based model rooted at u. Also, inductively we have $\delta^u_\theta(u) =$1-$\mathsf{type}(u) \times [1 \dots q] \times \{0, 1, 2\}$.

  Now for every $a^w_\Lambda \in A^w$ let $\{b_1 \dots b_q\}$ be the corresponding witnesses as described above. For every successor $(w, a^w_\Lambda, u) \in R$ and for every $1 \leq e \leq q$ and $f \in \{0, 1, 2\}$, create a new copy of T^u_θ (call it $N^{(\Lambda,e,f)}$) and name its root as $u^{(\Lambda,e,f)}$. Now add $\delta^w_\theta(w)$ to $N^{(\Lambda,e,f)}$ at $u^{(\Lambda,e,f)}$ guided by Ω where Ω is defined as follows:
  - For all $\Pi \in$ 1-$\mathsf{type}(w)$ we have $a^w_\Pi \in A^w$. Define $\Omega((\Pi, e, f)) = ($1-$\mathsf{type}(u, a^w_\Pi), e, f)$.
  - for all $k \leq q$ if 1-$\mathsf{type}(u, b_k) = \Pi$ then $\Omega((\Pi, k, f')) = ($1-$\mathsf{type}(u, b_k), e, f)$ where $f' = f + 1 \mod 3$.
  - Let $f' = f - 1 \mod 3$. For all $\Pi \in$ 1-$\mathsf{type}(w)$ let the witness set of a^w_Π be $\{d_1 \dots d_q\}$. For all $l \leq q$ if 1-$\mathsf{type}(w, d_l) = \Lambda$ then by Λ_3 component, there is some $a \in \delta(w)$ such that 2-$\mathsf{type}(w, d_l, a^w_\Pi) =$ 2-$\mathsf{type}(w, a^w_\Lambda, a)$. Define $\Omega((\Pi, l, f')) = ($1-$\mathsf{type}(u, a), e, f)$.
  - For all $(\Pi, e', f') \in \delta^w_\theta(w)$ if $\Omega(\Pi, e', f')$ is not yet defined, then set $\Omega(\Pi, e', f') = ($1-$\mathsf{type}(u, a^w_\Pi), e, f)$.

  Add an edge $(w, (\Lambda, e, f), u^{(\Lambda,e,f)})$ to R^w_θ.

Note that Ω is well defined since the first three steps are defined for the indices f, (f+1 mod 3) and (f-1 mod 3) respectively, which are always distinct. Also note that T^r_θ is a model that satisfies bounded agent property. Thus, it is sufficient to prove that $T^r_\theta, r \models \theta$.

Claim. For every $w \in W$ at height h and for all $\lambda \in \mathsf{SF}^h(\theta)$ the following holds:

1. Suppose λ is a sentence and $M, w \models \lambda$ then $T^w_\theta, w \models \lambda$.
2. If $\mathsf{Fv}(\lambda) \subseteq \{x, y\}$ and for all $\Lambda, \Pi \in$ 1-$\mathsf{type}(w)$ if $M, w, [x \mapsto a^w_\Lambda, y \mapsto a^w_\Pi] \models \lambda$ then for all $1 \leq e \leq q$ and $f \in \{0, 1, 2\}$ we have $T^w_\theta, w, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi, e, f)] \models \lambda$.

Note that the theorem follows from claim (1), since θ is sentence and $M, r \models \theta$.

The proof of the claim is by reverse induction on h. In the base case $h = \mathsf{md}(\theta)$ which implies λ is modal free and hence is a DNF over $\mathsf{literals}$. Thus, both the claims follow since $\rho(w) = \rho^w_\theta(w)$.

For the induction step, let w be at height h. Now we induct on the structure of λ. Again if λ is a $\mathsf{literal}$ then both the the claims follow since $\rho(w) = \rho^w_\theta(w)$. The case of $\wedge$ and $\vee$ are standard.

For the case $\Box_x \lambda$, we only need to prove claim(2). Now suppose $M, w, [x \mapsto a^w_\Lambda, y \mapsto a^w_\Pi] \models \Box_x \lambda$. Pick arbitrary e and f. We need to prove that $T^w_\theta, w, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi, e, f)] \models \Box_x \lambda$. Pick any $(w, (\Lambda, e, f), u^{(\Lambda,e,f)}) \in R^w_\theta$, then by construction we have $(w, a^w_\Lambda, u) \in R$ and since $M, w, [x \mapsto a^w_\Lambda, y \mapsto a^w_\Pi] \models \Box_x \lambda$, we have $M, u, [x \mapsto a^w_\Lambda, y \mapsto a^w_\Pi] \models \lambda$. Let $a^u_{\Pi'} \in A^u$ such that 1-$\mathsf{type}(u, a^u_{\Pi'}) =$1-$\mathsf{type}(u, a^w_\Pi)$ and since a^w_Λ is the incoming edge of u, by Π_2 component, we have 2-type$(u, a^w_\Pi, a^w_\Lambda)$ =2-type$(u, a^u_{\Pi'}, a^w_\Lambda)$ and also $a^w_\Lambda \in A^u$. Hence $M, u, [x \mapsto a^w_\Lambda, y \mapsto a^u_{\Pi'}] \models \lambda$ and by induction hypothesis $T^u_\theta, u, [x \mapsto ($1-type$(u, a^w_\Lambda), e, f), y \mapsto ($1-type$(u, a^u_{\Pi'}), e, f)] \models \lambda$. Now by construction, at $u^{(\Lambda,e,f)}$ we have $\Omega(\Lambda, e, f) = ($1-type$(w, a^w_\Lambda), e, f)$ and $\Omega(\Pi, e, f) = ($1-type$(u, a^u_{\Pi'}), e, f)$. Thus, by Lemma 12, $T^w_\theta, u^{(\Lambda,e,f)}, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi, e, f)] \models \lambda$. Hence, we have $T^w_\theta, w, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi, e, f)] \models \Box_x\lambda$. The case for $\Box_y\lambda$ is analogous.

For the case $\Diamond_y\lambda$, again only claim(2) applies. Suppose $M, w, [x \mapsto a^w_\Lambda, y \mapsto a^w_\Pi] \models \Diamond_y\lambda$. Now pick e and f appropriately. We need to prove that $T^w_\theta, w, [x \mapsto (\Gamma, e, f), y \mapsto (\Pi, e, f)] \models \Diamond_y\lambda$. By supposition, there is some $w \xrightarrow{a^w_\Pi} u$ such that $M, u, [x \mapsto a^w_\Lambda, y \mapsto a^w_\Pi] \models \lambda$. Using the argument similar to the previous case, we can prove that $T^w_\theta, u^{(\Lambda,e,f)}, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi, e, f)] \models \lambda$ and hence $T^w_\theta, w, [x \mapsto (\Gamma, e, f), y \mapsto (\Pi, e, f)] \models \Diamond_y\lambda$. The case of $\Diamond_x\lambda$ is symmetric.

For the case $\exists y\ \lambda$ (where x is free at the outer most level), for claim (2) first note that since θ is in the normal form, λ is quantifier-safe. Also note that $\exists y\ \lambda = \chi_i$ for some $\chi_i \in E_\theta$. Now, suppose $M, w, [x \mapsto a^w_\Lambda] \models \exists y\ \lambda$ then we need to prove that $T^w_\theta, w, [x \mapsto (\Lambda, e, f)] \models \exists y\ \lambda$. Let the i^{th} witness of a^w_Λ be b_i and hence $M, w, [x \mapsto a^w_\Lambda, y \mapsto b_i] \models \lambda$. Let 1-type$(w, b_i) = \Pi'$, we claim that $T^w_\theta, w, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi', i, f')] \models \lambda$ where $f' = f + 1 \mod 3$. Suppose not, then $\wedge$ and $\vee$ can be broken down and we get some module such that $M, w, [x \mapsto a^w_\Lambda, y \mapsto b_i] \models \Delta_z\lambda'$ and $T^w_\theta, w, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi', i, f')] \not\models \Delta_z\lambda'$ where $\Delta \in \{\Box, \Diamond\}$ and $z \in \{x, y\}$. Assume $\Delta = \Box$ and $z = x$ (other cases are analogous). This implies $T^w_\theta, w, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi', i, f')] \models \Diamond_x\neg\lambda'$ and hence there is some $w \xrightarrow{(\Lambda,e,f)} u^{(\Lambda,e,f)}$ such that $T^w_\theta, u^{(\Lambda,e,f)}, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi', i, f')] \models \neg\lambda'(*)$. By construction, there is a corresponding $w \xrightarrow{a^w_\Lambda} u$ in M. Now since $M, w, [x \mapsto a^w_\Lambda, y \mapsto b_i] \models \Box_x\lambda'$, we have $M, u, [x \mapsto a^w_\Lambda, y \mapsto b_i] \models \lambda'$. Let $b'_i \in A^u$ such that 1-type(u, b_i) =1-type(u, b'_i). Since a^w_Λ is the incoming edge to u by Π'_2 component, we have 2-type(u, b_i, a^w_Λ) =2-type(u, b'_i, a^w_Λ) and $a^w_\Lambda \in A^u$. Thus, $M, u, [x \mapsto a^w_\Lambda, y \mapsto b'_i] \models \lambda'$ and by induction hypothesis, $T^u_\theta, u, [x \mapsto (\Lambda, e, f), y \mapsto ($1-type$(u, b'_i), e, f)] \models \lambda'$. Again by construction, at u we have $\Omega((\Lambda, e, f)) = (\Lambda, e, f)$ and $\Omega((\Pi', i, f')) = ($1-type$(u, b'_i), e, f)$ and hence by Lemma 12, $T^w_\theta, u^{(\Lambda,e,f)}, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi', i, f')] \models \lambda'$ which is a contradiction to (*). The case of $\exists y\ \lambda$ is analogous.

For the case of $\forall x\ \lambda$ (where y is free at the outer most level), suppose $M, w, [y \mapsto a^w_\Pi] \models \forall x\ \lambda$. We need to prove that $T^w_\theta, w, [y \mapsto (\Pi, e, f)] \models \forall x\ \lambda$. Pick any $(\Lambda', e', f') \in \delta^w_\theta(w)$, now we claim $T^w_\theta, w, [x \mapsto (\Lambda', e', f'), y \mapsto (\Pi, e, f)] \models \lambda$ (otherwise, like in the previous case, since λ is quantifier-safe, we can reach a module where they differ and obtain a contradiction). The case $\forall y\ \lambda$ is analogous.

Finally we come to sentences which are relevant for claim (1). Note that in the normal form, at the outermost level, a sentence will have only literals or formulas of the form $\forall x \exists y\ \psi_l$ or $\forall x \forall y\ \varphi$.

For the case $M, w \models \forall x \exists y\ \psi_l$, let $\forall x \exists y\ \psi_l$ be i^{th} formula in E_θ. We need to prove $T^w_\theta, w \models \forall x \exists y\ \psi_l$. Pick any $(\Lambda, e, f) \in \delta^w_\theta(w)$ and we have $a^w_\Lambda \in A^w$. Let the i^{th} witness for a^w_Λ be b_i. Thus we have $M, w, [x \mapsto a_\Gamma, y \mapsto b_i] \models \psi_l$. Let 1-type$(w, b_i) = \Pi'$. Again we claim that $T^w_\theta, w, [x \mapsto (\Gamma, e, f), y \mapsto [\Pi', e, f')] \models \psi_l$ where $f' = f + 1 \mod 3$. Suppose not, again $\wedge$ and $\vee$ can be broken down and we get some module such that $M, w, [x \mapsto a^w_\Lambda, y \mapsto b_i] \models \Delta_z\lambda'$ and $T^w_\theta, w, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi', i, f')] \not\models \Delta_z\lambda'$ where $\Delta \in \{\Box, \Diamond\}$ and $z \in \{x, y\}$. Assume $\Delta = \Diamond$ and $z = y$ (other cases are analogous). This implies $T^w_\theta, w, [x \mapsto$

$(\Lambda, e, f), y \mapsto (\Pi', i, f')] \models \Box_y \neg\lambda'$ (*). Now let $a^w_{\Pi'} \in A^w$ such that 1-type$(w, a^w_{\Pi'})$ = 1-type$(w, b_i) = \Pi'$. Thus by Π'_3 component, there is some $d \in \delta^w_\theta$ such that 2-type$(w, a^w_{\Pi'}, d)$ = 2-type(w, b_i, a^w_Λ) and hence $M, w, [x \mapsto d, y \mapsto a^w_{\Pi'}] \models \Diamond_y \lambda'$. Hence there is some $w \xrightarrow{a^w_{\Pi'}} u$ such that $M, u, [x \mapsto d, y \mapsto a^w_{\Pi'}] \models \lambda'$. Now let 1-type$(u, d)$ = 1-type(u, d') such that $d' \in A^u$ and since $a^w_{\Pi'}$ is the incoming edge, we have $M, u, [x \mapsto d', y \mapsto a^w_{\Pi'}] \models \lambda'$ and by induction hypothesis, $T^u_\theta, u, [x \mapsto$ (1-type$(u, d'), i, f'), y \mapsto$ (1-type$(u, a^w_{\Pi'}), i, f')] \models \lambda'$ and while constructing $u^{(\Pi', i, f')}$ (case 3 applies for a^w_Λ since its i^{th} witness has same 1-type as $a^w_{\Pi'}$) we have $\Omega((\Lambda, e, f' - 1))$ = (1-type$(u, d'), i, f')$. Thus by Lemma 12 (since $f' - 1 = f$), $T^w_\theta, u^{(\Pi', i, f')}, [x \mapsto (\Lambda, e, f), y \mapsto (\Pi', i, f')] \models \lambda'$ which contradicts (*).

Finally, for the case $\forall x \forall y\ \varphi$ suppose $M, w \models \forall x \forall y\ \varphi$, then for any (Γ, e, f), $(\Delta, e', f') \in \delta^w_\theta(w)$ we claim that $T^w_\theta, w, [x \mapsto (\Gamma, e, f), y \mapsto (\Delta, e', f')] \models \varphi$ (else again, go to the smallest module and prove contradiction). ◀

Note that in the type based model, at any world w we have $|\delta^w_\theta| = 2^{2^{O(|\mathsf{SF}(\theta)|)}}$. Now if we start with a PTML^2 formula φ, then though its corresponding equi-satisfiable formula θ is exponentially larger, the number of distinct subformulas in θ is still linear in the size of φ.

▶ **Corollary 14.** TML^2 *satisfiability is in 2-EXPSPACE.*

Proof. Any TML^2 formula α is satisfiable iff (by Prop.6) its corresponding PTML^2 translation φ is satisfiable iff (by Theorem 13) the corresponding normal form θ of φ is satisfiable over agent set D of size $2^{2^{O(|\varphi|)}}$ iff (by Prop. 4) $\hat{\theta} \in$ PTML^2 is satisfiable in a constant domain model over D.

Thus we can expand the quantifiers of $\hat{\theta}$ by corresponding $\bigwedge$ and $\bigvee$ for $\forall$ and $\exists$ respectively and we get a propositional multi-modal formula. This satisfiability is in PSPACE. But in terms of the size of the formulas, $|\hat{\theta}| = 2^{2^{|\alpha|^2}}$. Thus we have a 2-EXPSPACE algorithm. ◀

3.3 Example

We illustrate the construction of *type based models* with an example. Consider the PTML^2 sentence $\theta := \forall x\ \Box_x \Box_x \bot \wedge \forall x \exists y\ (\Box_x(\Diamond_y(\neg p) \wedge \exists y\ \Diamond_y p))$ which is in FSNF DNF. Let M be the model described in Fig. 1 where

- $W = \{r\} \cup \{u^i, v^i, w^i \mid i \in \mathcal{N}\}$
- $D = \mathcal{N}$
- $\delta(r) = \{2i \mid i \in \mathcal{N}\}$ (all even numbers) and
  $\delta(w^i) = \delta(u^i) = \delta(v^i) = \mathcal{N}$
- $R = \{(r, 2i, w^i), (w^i, 2i + 1, u^i), (w^i, 2i + 2, v^i) \mid i \in \mathcal{N}\}$
- $\rho(r) = \rho(w^i) = \rho(v^i) = \emptyset$ and $\rho(u^i) = p$ for all $i \in \mathcal{N}$.

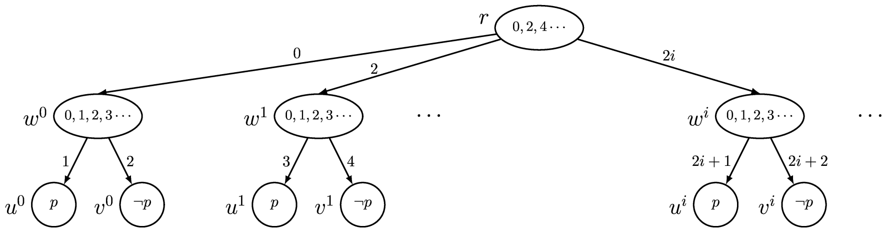


Figure 1 Given model such that $M, r \models \theta$.

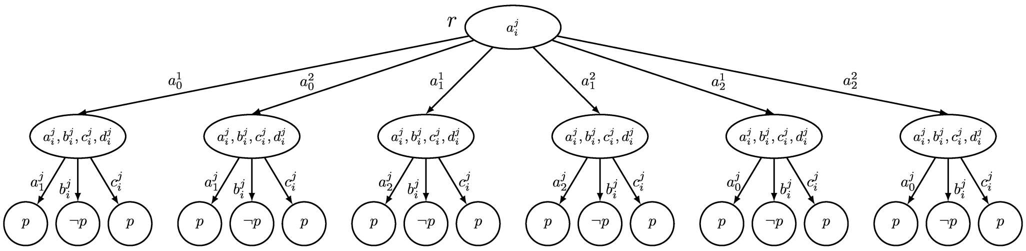


Figure 2 Corresponding bounded agent model with $M', r \models \theta$. a_i^j, b_i^j, c_i^j corresponds to agents with $1 \leq j \leq 2$ and $i \in \{0, 1, 2\}$. The edge a_i^j, b_i^j, c_i^j indicate one successor for every $1 \leq j \leq 2$ and $i \in \{0, 1, 2\}$.

Clearly, $M, r \models \theta$. Let $f^r : D_r \mapsto D_r$ be defined by $f^r(2i) = 2i + 2$ and at all w^i, $g^i(j) = 2i + 1$ for all $i \in \mathcal{N}$ be the two (relevant) witness functions. The one and two types at every world are described as follows:

At leaf nodes u^i and v^i there is only one distinct one type and two types. At w^i, note that $r \xrightarrow{2i} w_i$ is the incoming edge and only $2i + 1$ and $2i + 2$ have outgoing edges. Thus, there are 3 distinct 1-type members at w^i, each for $(2i + 1), (2i + 2)$ and [the rest]. Let b, c, d be the respective types. Finally at the root again we have only a single distinct type (call it a).

Since there are 2 existential formulas, the root of the type based model has $(1 \times 2 \times 3) = 6$ agents let it be $\{a_f^e \mid 1 \leq e \leq 2,\ 0 \leq f \leq 2\}$ and 0 be the representative. At w^0 we have $(3 \times 2 \times 3) = 18$ agents. Let the representatives be $1, 2, 0$ for b, c, d respectively. Note that we cannot pick any other representative for [the rest] other than 0 since 0 is the incoming edge to w^0. Let the bounded agent set be $\{b_f^e,\ c_f^e,\ d_f^e \mid 1 \leq e \leq 2,\ 0 \leq f \leq 2\}$. The corresponding bounded model M' is described in Figure 2. It can be verified that $M', r \models \theta$.

4 Discussion

We have proved that the two variable fragment of PTML^2 (and hence TML^2) is decidable. The upper bound shown is in 2-EXPSPACE. A NEXPTIME lower bound follows since FO^2 satisfiability can be reduced to PTML^2 satisfiability. We believe that by careful management of the normal form, space can be reused and the upper bound can in fact be brought down by one exponent. That would still leave a significant gap between lower and upper bounds to be addressed in future work.

We can also prove that addition of constants makes PTML^2 undecidable. In fact, with the addition of a single constant $\mathbf{c}$ we can use $\Box_{\mathbf{c}}$ to simulate the "free" $\Box$ of FOML^2, thus yielding undecidability. When it comes to equality, the situation is more tricky: note that we can no longer use model extension (Def.11 and Lemma 12) since equality might restrict the number of agents at every world.

The most important issue is expressiveness. What kind of accessibility relations or model classes can be characterized by 2-variable TML? This is unclear, but there are sufficiently intriguing examples and applications making the issue an interesting challenge.

References

1 Hajnal Andréka, István Németi, and Johan van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of philosophical logic*, 27(3):217–274, 1998.

2 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic (Cambridge Tracts in Theoretical Computer Science)*. Cambridge University Press, 2001.

3 Giovanna Corsi. A unified completeness theorem for quantified modal logics. *The Journal of Symbolic Logic*, 67(4):1483–1510, 2002.

4 Kit Fine et al. Normal forms in modal logic. *Notre Dame journal of formal logic*, 16(2):229–237, 1975.

5 Melvin Fitting and Richard L. Mendelsohn. *First-Order Modal Logic (Synthese Library)*. Springer, 1999.

6 Melvin Fitting, Lars Thalmann, and Andrei Voronkov. Term-Modal Logics. *Studia Logica*, 69(1):133–169, 2001. doi:10.1023/A:1013842612702.

7 Erich Grädel, Phokion G Kolaitis, and Moshe Y Vardi. On the decision problem for two-variable first-order logic. *Bulletin of symbolic logic*, 3(1):53–69, 1997.

8 Erich Grädel and Martin Otto. On logics with two variables. *Theoretical computer science*, 224(1-2):73–113, 1999.

9 MJ Hughes and GE Cresswell. *A New Introduction to Modal Logic. Routledge. 1996*. Routledge, 1996.

10 Roman Kontchakov, Agi Kurucz, and Michael Zakharyaschev. Undecidability of first-order intuitionistic and modal logics with two variables. *Bulletin of Symbolic Logic*, 11(3):428–438, 2005.

11 Barteld Kooi. Dynamic term-modal logic. In *A Meeting of the Minds*, pages 173–186, 2007.

12 Saul A. Kripke. The Undecidability of Monadic Modal Quantification Theory. *Mathematical Logic Quarterly*, 8(2):113–116, 1962. doi:10.1002/malq.19620080204.

13 Michael Mortimer. On languages with two variables. *Mathematical Logic Quarterly*, 21(1):135–140, 1975.

14 Eugenio Orlandelli and Giovanna Corsi. Decidable Term-Modal Logics. In *15th European Conference on Multi-Agent Systems*, 2017.

15 Anantha Padmanabha and R Ramanujam. The Monodic Fragment of Propositional Term Modal Logic. *Studia Logica*, pages 1–25, 2018.

16 Anantha Padmanabha and R Ramanujam. Two variable fragment of Term Modal Logic. *arXiv preprint*, 2019. arXiv:1904.10260.

17 Anantha Padmanabha, R Ramanujam, and Yanjing Wang. Bundled Fragments of First-Order Modal Logic: (Un)Decidability. In Sumit Ganguly and Paritosh Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, volume 122 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:20. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.43.

18 Mikhail Rybakov and Dmitry Shkatov. Undecidability of first-order modal and intuitionistic logics with two variables and one monadic predicate letter. *Studia Logica*, pages 1–23, 2017.

19 Gennady Shtakser. Propositional Epistemic Logics with Quantification Over Agents of Knowledge. *Studia Logica*, 106(2):311–344, 2018.

20 Gennady Shtakser. Propositional Epistemic Logics with Quantification Over Agents of Knowledge (An Alternative Approach). *Studia Logica*, August 2018. doi:10.1007/s11225-018-9824-6.

21 Yanjing Wang. A New Modal Framework for Epistemic Logic. In *Proceedings Sixteenth Conference on Theoretical Aspects of Rationality and Knowledge, TARK 2017, Liverpool, UK, 24-26 July 2017.*, pages 515–534, 2017. doi:10.4204/EPTCS.251.38.

22 Yanjing Wang and Jeremy Seligman. When Names Are Not Commonly Known: Epistemic Logic with Assignments. In *Advances in Modal Logic Vol. 12 (2018): 611-628, College Publications*, 2018 .

23 Frank Wolter and Michael Zakharyaschev. Decidable fragments of first-order modal logics. *The Journal of Symbolic Logic*, 66(3):1415–1438, 2001.

Choiceless Logarithmic Space

Erich Grädel
RWTH Aachen University, Germany
graedel@logic.rwth-aachen.de

Svenja Schalthöfer
RWTH Aachen University, Germany
schalthoefer@logic.rwth-aachen.de

Abstract

One of the most important open problems in finite model theory is the question whether there is a logic characterising efficient computation. While this question usually concerns PTIME, it can also be applied to other complexity classes, and in particular to LOGSPACE which can be seen as a formalisation of efficient computation for big data. One of the strongest candidates for a logic capturing PTIME is Choiceless Polynomial Time (CPT). It is based on the idea of choiceless algorithms, a general model of symmetric computation over abstract structures (rather than their encodings by finite strings). However, there is currently neither a comparably strong candidate for a logic for LOGSPACE, nor a logic transferring the idea of choiceless computation to LOGSPACE.

We propose here a notion of Choiceless Logarithmic Space which overcomes some of the obstacles posed by LOGSPACE as a less robust complexity class. The resulting logic is contained in both LOGSPACE and CPT, and is strictly more expressive than all logics for LOGSPACE that have been known so far. Further, we address the question whether this logic can define all LOGSPACE-queries, and prove that this is not the case.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases Finite Model Theory, Logics for Logspace, Choiceless Computation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.31

1 Introduction

The probably most fundamental problem of finite model theory concerns the question whether there exist logics that precisely characterise the efficiently computable queries on finite structures. Usually this problem is called the *quest for a logic for polynomial time* [15], a problem originally posed by Chandra and Harel [5] and later made more precise by Gurevich [19]. But PTIME is just one, although certainly the most common, complexity class to capture the intuitive notion of efficient computability. Another important such class is LOGSPACE, which formalises efficient computation over large data sets with significantly smaller working memory, and may thus be viewed as a notion of efficient computability for big data. When Chandra and Harel [5] enquired about a logical characterization of the PTIME-queries, they already asked about LOGSPACE as well, and since then, a number of logics have been proposed that capture relevant parts of LOGSPACE. We briefly compare the state of the art on the quests for logics for PTIME and LOGSPACE, respectively.

Logics for polynomial time. For PTIME, the logic of reference is *fixed-point logic with counting* (FPC), which has first been proposed informally by Immerman [20] and then made precise in [11]. It extends first-order logic by fixed-point operators and counting terms and actually comes rather close to being a logic for polynomial time. It is strong enough to express many of the algorithmic techniques leading to polynomial-time procedures and it captures PTIME on many interesting classes of structures, including planar graphs, structures of bounded tree width, and actually all classes of graphs with an excluded minor [16]. For

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 31; pp. 31:1–31:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a recent survey on FPC, see [6]. Nevertheless there are important problems that separate PTIME from FPC. The first of these, which still is an interesting benchmark problem for logics in PTIME, has been the CFI-query due to [4]. Other examples are efficiently solvable variants of the graph isomorphism problem or problems from linear algebra such as solving linear equation systems over finite fields or rings [2]. To handle such examples, several more powerful logics than FPC have been proposed, including for instance rank logic [7, 12].

However, the most promising candidate for a logic for PTIME is Choiceless Polynomial Time (CPT), introduced by Blass, Gurevich, and Shelah [3]. There are several different presentations of CPT (see e.g. [9, 10, 22, 25]). The original intention was to explore a model for efficient computations on finite structures which preserve symmetries at every step in the computation. This prohibits the explicit introduction of an ordering or, equivalently, arbitrary choices between indistinguishable elements of the input structure or of the current state. Such choices appear in many algorithms of fundamental importance, including depth-first search, Gaussian elimination, and many more. CPT is based on a computation model that avoids symmetry breaking choices, but allows essentially everything else, including parallelism and "fancy data structures", as long as all operations can be carried out in polynomial time. It works, given a finite structure, on its extension by all hereditarily finite sets over its universe, which may be seen as a powerful higher-order data structure. Choiceless Polynomial Time is the restriction of this model to polynomial-time resources. It is known that CPT is strictly more powerful than fixed-point logic with counting. In particular, CPT can express several variants of the CFI-queries that are not expressible in FPC and, more generally, CPT captures polynomial time over interesting classes of structures over which FPC fails to do so [1, 8, 23]. Nevertheless it is still open whether CPT captures PTIME on arbitrary finite structures. We refer to [9, 22, 26] for more information on CPT.

Logics for logarithmic space. The state of the art concerning logics for LOGSPACE is less advanced than for PTIME. On ordered structures, LOGSPACE can be captured by DTC, the extension of first-order logic by deterministic transitive closures [21]. A more elegant variant is the symmetric transitive closure logic STC, whose evaluation in LOGSPACE however depends on the highly non-trivial algorithm by Reingold [24] for undirected graph reachability. But even with an added counting operation, STC does not capture LOGSPACE on arbitrary structures, and again this can be proved by the CFI-query. Some of the shortcomings of transitive closure logics are elegantly countered by the logic LREC (short for L-recursion), especially in its stronger variant, called $\text{LREC}_=$ in the literature [17, 18]. The core of LREC is an elaborate recursion operator, augmented by a mechanism defining symmetric transitive closures to obtain closure under interpretations. However, although LREC captures LOGSPACE on a larger class of structures than transitive closure logics, it is still contained in FPC and thus does not capture LOGSPACE on arbitrary finite structures.

There has, up to now, not really emerged a convincing logspace-analogue of Choiceless Polynomial Time, and also no other really serious candidate for a logic for LOGSPACE. So some natural questions arise: *"What is Choiceless Logspace? What expressive power does it have, and could it actually capture all of* LOGSPACE*?"* Addressing these questions, we propose a notion of Choiceless Logspace. We prove that this provides a logic that is indeed contained in both CPT and LOGSPACE. Moreover, it is strictly more powerful than all LOGSPACE-logics known so far, including (the strong variant of) LREC. However, it turns out that even this logic fails to capture all LOGSPACE queries, which leads to the more general question of what would be a suitable notion of efficient choiceless computation for big data.

Sets of logarithmic size. Towards such a definition, we first note that the obvious, but naïve attempt fails. Since CPT permits polynomially many computation steps using only sets whose transitive closures contain polynomially many objects, the naïve approach would simply discard the time bound and allow sets with a transitive closure of logarithmically many objects. But this would make it possible to define sets containing logarithmically many atoms, which admits no straightforward evaluation in LOGSPACE: to represent such a set, one would store every atom as a number with logarithmically many bits, so one would actually have to represent logarithmically many numbers of logarithmic size. Unless LOGSPACE = NC^2, this should not be possible.

The central question is thus what it means that a set is of logarithmic size. Even though choiceless computation is invariant under encodings, already the size of an atom depends on its encoding. On an ordered encoding of a structure, every atom can be referenced by a binary string with logarithmically many bits. On the basis of this straightforward encoding, a logspace algorithm could only store constantly many atoms at a time, so a set of logarithmic size would be a set whose transitive closure is bounded by a constant. In fact, many standard logspace algorithms operate with a bounded number of storage locations, each of which can store an object with logarithmically many bits, typically an element of the input structure or a number that is polynomially bounded by the size of the input structure. A previous approach to defining choiceless logarithmic space [13, 27] has operated on exactly these "bounded sets". It corresponds to a logic, denoted here BDTC, that provides an operator for deterministic transitive closures over bounded sets. In its original version, BDTC lacks the ability to count, but besides that it has more fundamental deficiencies. Indeed, the assumption that a set of logarithmic size can only contain constantly many atoms does not take into account that there are alternative ways to represent atoms, and logspace algorithms that operate in a more sophisticated manner, using an unbounded number of storage locations such as the algorithm for tree isomorphism or Reingold's algorithm for undirected graph reachability. It is, for instance, possible to store the unique root of a tree in constant space by just storing its defining property. In a structure with a unary predicate P, every atom in P can be represented as "the kth element of P", so the size of the atom representation is logarithmic in $|P|$ instead of the size of the whole structure. The size of an atom with respect to LOGSPACE-algorithms can hence vary depending on its representation.

Towards Choiceless Logspace. This observation leads to the main idea behind CLogspace, the logic we shall introduce: When evaluating formulae with logarithmic bounds, we do not assume a fixed size for every atom. Therefore, the sizes of atoms are part of the semantics and, consequently, the logic is evaluated over *size-annotated hereditarily finite sets.*

Atoms can be represented using logical formulae. We introduce terms "$\mathrm{Atoms}\,.\varphi$" in our logic to make sure that, whenever a set of atoms is defined, that definition is guarded by a formula. The size of the atoms in that set will then be defined assuming that every atom is represented as "the ith entry of the kth tuple satisfying φ". So the size of every hereditarily finite object will depend on the terms of the form $\mathrm{Atoms}\,.\varphi$ generating the atoms in its transitive closure. However, this way of defining atoms does not allow unbounded recursion. We therefore add (a modified version of) the recursion operator from LREC to our logic. But, just like in CPT, iterated *creation* of sets is necessary to obtain a formalism that is stronger than common logics. Consequently, our logic also incorporates CPT-style iteration with some modifications mimicking the behaviour of LOGSPACE-algorithms.

In a nutshell, Choiceless LOGSPACE in our sense means iteration and recursion over hereditarily finite sets of logarithmic size, where the size of each atom is derived from its representation through a formula.

2 Choiceless Logarithmic Space and Choiceless Polynomial Time

In this section we present the precise definition of CLogspace. Readers familiar with Choiceless Polynomial Time will recall that it is evaluated with the data structure of all *hereditarily finite objects* over the given input structure.

The elements of the universe A of an input structure $\mathbf{A}$ are assumed to be *atoms*, i. e. not sets, and an *object* is an atom or a set. A set a is *transitive* if $\in$ is a transitive relation on a, which means that all elements of a are either subsets of a or atoms. The transitive closure $\mathrm{tc}(a)$ of an object a is the smallest transitive set b with $a \in b$. An object a is hereditarily finite if $\mathrm{tc}(a)$ is finite. Given a set A of atoms, $\mathrm{HF}(A)$ is the collection of hereditarily finite objects over A, i. e. the set of all objects x such that $x \in A$ or x is a hereditarily finite set such that all atoms in $\mathrm{tc}(x)$ are elements of A.

Choiceless Logspace is evaluated over *size-annotated hereditarily finite sets* where a size annotation is a function assigning (in a consistent way) a size to every element of the transitive closure of the given set. The definition of sizes guarantees that a set of logarithmic size can be represented in LOGSPACE in a straightforward way.

▶ **Definition 1.** *Let $\mathbf{A}$ be a σ-structure and $a \in \mathrm{HF}(A)$. A function $s\colon \mathrm{tc}(a) \to \mathbb{N}$ is a* size annotation of *a if, for each set $b \in \mathrm{tc}(a)$, $s(b) = 1+\sum_{c\in b} s(c)$. The set of* size-annotated hereditarily finite objects *is $\mathrm{SHF}(A) := \{(a, s_a) : a \in \mathrm{HF}(A)$ and s_a is a size annotation of $a\}$. For size annotations $(a, s_a), (b, s_b)$, we say that $(b, s_b) \in (a, s_a)$ if $b \in a$ and $s_b = s_a{\restriction}\mathrm{tc}(b)$.*

Note that a size annotation is completely determined by the values it assigns to the atoms. In particular, for a *pure set* $a \in \mathrm{HF}(A)$, with $\mathrm{tc}(a) \cap A = \emptyset$, there is a unique size annotation, which we denote by ann_a. Further, the size annotation yields an upper bound for the size of the transitive closure. As atoms may have size 0, conclude that $|\,\mathrm{tc}(a) \setminus A| \leq s(a)$ for any size annotation s of a.

We now formulate a high-level definition of the logic CLogspace, and then proceed to make precise, and explain, its ingredients step by step.

▶ **Definition 2** (Choiceless Logarithmic Space)**.** *The logic* CLogspace *is defined by rules for ordinary terms and formulae, iteration terms and recursion formulae as follows. If t is a term and φ is a formula according to these rules, and f is a function $f\colon n \mapsto c \log n$ for $c \in \mathbb{N}$, then (t, f) is a* CLogspace*-term, and (φ, f) is a* CLogspace*-formula.*

The function f only plays a role in defining the semantics of iteration terms. Otherwise the semantics is just given by the semantics of t and φ, respectively.

We first explain ordinary terms and formulae which provide the basic operations for constructing sets.

▶ **Definition 3** (Syntax)**.** *Ordinary terms and formulae over a vocabulary σ are defined inductively as follows:*

- *$\emptyset$ is a term, and every variable x is a term,*
- *if $\varphi \in \mathrm{FO}[\sigma]$ with at least one free variable, then $\mathrm{Atoms}\,.\varphi$ is a term,*
- *if t_1, t_2 are terms, then $\mathrm{Union}(t_1)$, $\mathrm{Unique}(t_1)$, $\mathrm{Pair}(t_1, t_2)$, and $\mathrm{Card}(t_1)$ are terms,*
- *for $R \in \sigma$ and terms $t_1, \dots, t_k$, the expressions $t_1 = t_2$, $t_1 \in t_2$, and $Rt_1 \dots t_k$ are formulae,*
- *Boolean combinations of formulae are formulae,*
- *if p and q are terms and φ is a formula, then $\{p : x \in q : \varphi\}$ is a (comprehension) term.*

The free variables $\mathrm{free}(t)$ of a term or formula t are defined as usual, where, in the term $t := \{p : x \in q : \varphi\}$, x occurs free in p and φ and bound in t. We therefore write $\{p(x) : x \in q : \varphi(x)\}$.

Counting. The terms $\mathrm{Card}(t)$ define cardinalities of sets. Since the cardinality of sets of polynomial size can be computed in LOGSPACE, von Neumann ordinals, which represent cardinalities in Choiceless Polynomial Time, are in general too large for our purposes. Therefore cardinalities will be denoted by a set encoding of their binary representation. For this purpose, we assume the short form $\langle a, b\rangle = \{a, \{a, b\}\}$ of Kuratowski pairs. This will make sure that every word of length two is represented by a set with two elements and can thus be distinguished from $\mathrm{bitset}(0)$ and $\mathrm{bitset}(1)$.

▶ **Definition 4.** *We define the function* $\mathrm{bitset}\colon \{0,1\}^+ \to \mathrm{HF}(\emptyset)$ *by setting* $\mathrm{bitset}(0) := \emptyset$, $\mathrm{bitset}(1) := \{\emptyset\}$, *and* $\mathrm{bitset}(bw) = \langle \mathrm{bitset}(b), \mathrm{bitset}(w)\rangle$ *for* $b \in \{0,1\}$ *and* $w \in \{0,1\}^+$.

If $n \in \mathbb{N}$, *let* $\mathrm{bitset}(n)$ *denote* $\mathrm{bitset}(\mathrm{bin}(n))$, *where* $\mathrm{bin}(n)$ *is the binary representation of* n, *with* 2^0 *as the right-most bit and without leading zeroes.*

It follows from the definition that the function bitset is injective. This representation of numbers keeps their size logarithmic with respect to their unique size annotation. More precisely, a simple induction on $|w|$ shows:

▶ **Lemma 5.** *Let* $s\colon \mathrm{tc}(\mathrm{bitset}(w)) \to \mathbb{N}$ *be a size annotation for* $w \in \{0,1\}^+$. *Then* $s(\mathrm{bitset}(w)) \leq 6|w|$.

Semantics of ordinary terms and formulae. Evaluation over size-annotated objects means that the value of every term is equipped with a size annotation. Hence, also the values of free variables have to incorporate size annotations. Recall that a size annotation defines a size for every element of the transitive closure of an object, which is uniquely determined by the sizes associated with the atoms. Unless the size of an atom is given externally through a free variable, its size originates from the term defining it. The only way to define atoms directly will be with terms of the form $\mathrm{Atoms}\,.\varphi$.

We assume every atom to be represented as "the ith entry of the kth tuple satisfying φ". Both φ and i are of constant size, since φ is a part of the term. So we can assume that, up to an additive constant, such an atom can be written as a number with $\log k$ bits where k is bounded by the number of tuples satisfying φ. This number is the size we assign to every atom defined that way. If an atom occurs in the values of multiple subterms, it may have multiple sizes that contribute to the value of the term. In that case, the minimal size is picked, which is formalised as follows:

▶ **Definition 6.** *Let* $S \subseteq \mathrm{SHF}(A)$ *be a set of size-annotated hereditarily finite sets. We denote by* S_{SHF} *the pair* (s, ann_s) *where* $s = \{s' : (s', \mathrm{ann}_{s'}) \in S\}$ *and* ann_s *is the unique size annotation of* s *with* $\mathrm{ann}_s(a) = \min\{\mathrm{ann}_{s'}(a) : (s', \mathrm{ann}_{s'}) \in S\}$ *for every atom* $a \in \mathrm{tc}(s)$.

▶ **Definition 7** (Semantics of ordinary terms). *Let* $\mathbf{A}$ *be a* σ*-structure and let* $\beta\colon X \to \mathrm{SHF}(A)$ *be a variable assignment. For* $\alpha \in \mathrm{SHF}(A)$, *we write* $\beta[x \mapsto \alpha]\colon X \cup \{x\} \to \mathrm{SHF}(A)$ *to denote the function that behaves like* β *on* $X \setminus \{x\}$ *and maps* x *to* α. *For an ordinary* σ*-term* t *(with* $\mathrm{free}(t) \subseteq X$*), we define the value* $t^{\mathbf{A},\beta} = ([\![t]\!]^{\mathbf{A},\beta}, \mathrm{ann}_t^{\mathbf{A},\beta}) \in \mathrm{SHF}(A)$ *as follows:*

- $\emptyset^{\mathbf{A},\beta} = (\emptyset, \emptyset \mapsto 1)$, *and* $x^{\mathbf{A},\beta} = \beta(x)$ *for any variable* x.
- *For* $t = \mathrm{Atoms}\,.\varphi$ *with* $\varphi \in \mathrm{FO}[\sigma]$, *let* $[\![t]\!]^{\mathbf{A},\beta} = \{a : \mathbf{A} \models \varphi(a)\}$ *if* $|\mathrm{free}(\varphi)| = 1$, *and* $[\![t]\!]^{\mathbf{A},\beta} = \{\{\langle \mathrm{bitset}(1), a_1\rangle, \dots, \langle \mathrm{bitset}(k), a_k\rangle\} : \mathbf{A} \models \varphi(a_1, \dots, a_k)\}$ *otherwise. In both cases* $\mathrm{ann}_t^{\mathbf{A},\beta}$ *is generated by mapping every occurring atom* a *to* $\log|[\![t]\!]^{\mathbf{A},\beta}|$.
- *If* $t = \mathrm{Pair}(t_1, t_2)$, *then* $t^{\mathbf{A},\beta} = \{{t_1}^{\mathbf{A},\beta}, {t_2}^{\mathbf{A},\beta}\}_{\mathrm{SHF}}$.
- *If* $t = \mathrm{Unique}(t')$, *then* $t^{\mathbf{A},\beta} = (a, \mathrm{ann}_{t'}^{\mathbf{A},\beta} \restriction \mathrm{tc}(a))$ *if* $[\![t']\!]^{\mathbf{A},\beta} = \{a\}$ *and* $t^{\mathbf{A},\beta} = (\emptyset, \emptyset \mapsto 1)$ *otherwise.*

- *If* $t = \mathrm{Union}(t')$, *then* $[\![t]\!]^{\mathbf{A},\beta} = \bigcup_{b\in[\![t']\!]^{\mathbf{A},\beta}} b$ *and* $\mathrm{ann}_t^{\mathbf{A},\beta} = \mathrm{ann}_{t'}^{\mathbf{A},\beta} \upharpoonright [\![t]\!]^{\mathbf{A},\beta}$.
- *If* $t = \mathrm{Card}(t')$, *then* $[\![t]\!]^{\mathbf{A},\beta} = \mathrm{bitset}\left(\left|[\![t']\!]^{\mathbf{A},\beta}\right|\right)$, *and* $\mathrm{ann}_t^{\mathbf{A},\beta} = \mathrm{ann}_{[\![t]\!]^{\mathbf{A},\beta}}$.
- *If* $t = \{p : x \in q : \varphi\}$, *then* $t^{\mathbf{A},\beta} = \{p^{\mathbf{A},\beta[x\mapsto a]} : a \in q^{\mathbf{A},\beta} : \mathbf{A},\beta[x \mapsto a] \models \varphi\}_{\mathrm{SHF}}$.

For a σ*-formula* φ*, the truth value of* $\mathbf{A},\beta \models \varphi$ *is defined in the obvious way.*

We denote the size $\mathrm{ann}_t^{\mathbf{A},\beta}([\![t]\!]^{\mathbf{A},\beta})$ assigned to the value of a term t by $\|t\|^{\mathbf{A},\beta}$. Further, we write true^k to denote a tautology with k free variables, true for true^1 and Atoms instead of Atoms . true. Then Atoms . true^k defines the set of all k-tuples over the input structure.

Iteration terms. Ordinary terms and formulae cannot define arbitrarily nested sets. This is achieved by constructing sets using fixed-point iteration.

▶ **Definition 8** (Syntax of iteration terms). *If* u *and* x *are variables and* t *is a term, then* $[t_u]^*(x)$ *is an* iteration term *with free variables* $\{x\} \cup \mathrm{free}(t) \setminus \{u\}$.

The fixed-point iteration starts with the object given by the free variable x, and the variable u is usually a free variable of t, whose value is the previous stage of the iteration. To ensure Logspace-computability, the fixed point is computed only if the logarithmic bound given as part of the term is satisfied by all intermediate stages. The value of an iteration term is not the fixed point itself, but the set of all stages up to that point.

▶ **Definition 9** (Semantics of iteration terms). *The* i*-fold iteration of a term* s *with free variable* u *is defined by induction as* $s^0 = s[u/x]$ *and* $s^{i+1} = s[u/s^i]$*, where* $s[u/t]$ *results from replacing every occurrence of* u *in* s *by the term* t*.*

Now let $t = [s_u]^*(x)$ *be an iteration term, let* $\mathbf{A}$ *be a structure and* $\beta\colon \mathrm{free}(t) \to \mathrm{SHF}(A)$ *a variable assignment. Then*

$$[\![t]\!]^{\mathbf{A},\beta} = \begin{cases} \{(s^i)^{\mathbf{A},\beta} : i \leq \ell\}_{\mathrm{SHF}}, & \text{for the least } \ell \text{ with } s^\ell = s^{\ell+1} \\ & \text{and } \|s^i\|^{\mathbf{A},\beta} \leq f(|A|) \text{ for all } i \leq \ell \\ & \text{if such an } \ell \text{ exists,} \\ (\emptyset, \emptyset \mapsto 1), & \text{otherwise.} \end{cases}$$

The differences between this definition and iteration in CPT are explained by two important properties of Logspace-algorithms: Firstly, a Logspace-algorithm can be sequentially executed for different input values, e.g. in a `forall`-loop. Iteration terms allow to do this in parallel for different values of the free variable x using suitable comprehension terms. Secondly, defining the value of the iteration term as the aggregation of all intermediate stages allows to generate outputs of polynomial size, which can be the input for other iteration terms.

Recursion formulae. With recursion formulae, we basically add to our logic the lrec-operator from [17, 18]. This operator permits a restricted kind of recursion over a graph interpretable in the input structure. Since our logic operates on hereditarily finite sets, we modify the interpretation such that the domain of the interpreted structure can be any definable set. Recursion formulae contain a term t_δ that restricts the domain to finitely many objects and defines their size annotations. Further, the vertices of the graph are labelled by sets of natural numbers, which we represent as bitsets instead of elements of a distinct number sort.

▶ **Definition 10** (Syntax of recursion formulae). *If* t_δ *and* t_C *are terms and* φ_E *and* $\varphi_=$ *are formulae, then* $[\mathsf{lrec}_{u,v} t_\delta, \varphi_=, \varphi_E, t_C](x, y)$ *is a* recursion formula*.*

For the definition of the semantics, fix a σ-structure $\mathbf{A}$, a variable assignment β, and a recursion formula $\varphi = [\mathsf{lrec}_{u,v} t_\delta, \varphi_=, \varphi_E, t_C](x,y)$. We first describe how t_δ, $\varphi_=$, φ_E and t_C define a labelled graph $G = (V, E, C)$ interpreted in the input structure. The formulae $\varphi_=$ and φ_E will be evaluated for pairs of elements from $[\![t_\delta]\!]^{\mathbf{A},\beta}$. For that purpose, we define the extension $\beta_{a,b} := \beta[u \mapsto (a, \mathrm{ann}^{\mathbf{A},\beta}_{t_\delta} \restriction \mathrm{tc}(a)), v \mapsto (b, \mathrm{ann}^{\mathbf{A},\beta}_{t_\delta} \restriction \mathrm{tc}(b))]$ of β for $a, b \in [\![t_\delta]\!]^{\mathbf{A},\beta}$.

The vertex set V consists of the equivalence classes of the reflexive, symmetric, transitive closure of the relation $\{(a,b) \in [\![t_\delta]\!]^{\mathbf{A},\beta} \times [\![t_\delta]\!]^{\mathbf{A},\beta} : \mathbf{A}, \beta_{a,b} \models \varphi_=\}$. Let $[a]$ denote the equivalence class of a. The edge set of G is $E := \{([a],[b]) : \mathbf{A}, \beta_{a,b} \models \varphi_E\}$. Further, the label of $[a] \in V$ is the set $C([a]) = \{c :$ there exists $a' \in [a]$ with $\mathrm{bitset}(c) \in [\![t_C]\!]^{\mathbf{A},\beta_{a'}}\}$, where $\beta_{a'} = \beta[u \mapsto (a', \mathrm{ann}^{\mathbf{A},\beta}_{t_\delta} \restriction \mathrm{tc}(a'))]$.

The truth value of the recursion formula depends on a recursive property X of vertices of a DAG unfolding of G. This unfolding is obtained by augmenting every vertex $[a] \in V$ with a resource $\ell \in \mathbb{N}$ restricting the space necessary to process paths through the DAG (for a detailed explanation, see [17, 18]). Then $([a], \ell) \in X \subseteq V \times \mathbb{N}$ if, and only if,

$$\ell > 0 \text{ and } \mathrm{bitset}\left(\left|\left\{[b] \in [a]E : \left([b], \left\lfloor \frac{\ell - 1}{|E[b]|} \right\rfloor\right) \in X\right\}\right|\right) \in C([a]),$$

If the values of the free variables x, y define a pair in $V \times \mathbb{N}$ (i.e. a vertex of the DAG), then the membership of that pair in X determines the truth value of the recursion formula:

▶ **Definition 11** (Semantics of recursion formulae). *Let $\varphi, \mathbf{A}, \beta$ and X as above. Then $\mathbf{A}, \beta \models \varphi$, where $\beta(x) = (a, \mathrm{ann}^{\mathbf{A},\beta}_{t_\delta} \restriction \mathrm{tc}(a))$ and $\beta(y) = (\mathrm{bitset}(\ell), \mathrm{ann}_{\mathrm{bitset}(\ell)})$ for $\ell \in \mathbb{N}$, if, and only if, $([a], \ell) \in X$.*

This completes the definition of CLogspace. We illustrate it with two short examples.

▶ **Example 12** (Size annotations). To see that the size annotation of a term can map the atoms to different values, consider the term $\mathrm{Union}(\mathrm{Pair}(\mathrm{Atoms}\,.\,\mathrm{true}, \mathrm{Atoms}\,.Px))$. The value of that term in a structure $\mathbf{A}$ is its whole domain. But the size annotation maps every element of $P^{\mathbf{A}}$ to $\log |P^{\mathbf{A}}|$, and the other atoms to $\log |A|$.

To demonstrate the use of iteration terms, we construct a formula $\varphi_{\mathrm{dtc}}(x,y)$ defining that there is a deterministic path from x to y.

▶ **Example 13** (Deterministic paths). The core of the formula is an iteration term $[t_u]^*(x)$ progressing along the deterministic path starting from x:

$$t(u) = \mathrm{Unique}\left(\{z : z \in \mathrm{Atoms}\,.\,\mathrm{true} : Euz\}\right).$$

Note that the free variable x determines the value of the first stage. So, by definition of Unique, the ith stage of $[t_u]^*(x)$ is the ith element on the deterministic path starting at x. Every stage is added to the value of the term, so the desired formula is $\varphi_{\mathrm{dtc}}(x,y) = y \in [t_u]^*(x)$. Since every atom is initially defined with the term $\mathrm{Atoms}\,.\,\mathrm{true}$, it is mapped to $\log |A|$ by the size annotation for the structure $\mathbf{A}$. Thus $n \mapsto \log n$ is a suitable bound for the formula.

Choiceless Polynomial Time. The newly introduced features that distinguish Choiceless Logspace from Choiceless Polynomial Time are size annotations, the addition of recursion formulae and the semantics of iteration terms. Further, CPT-formulae clearly possess polynomial instead of logarithmic bounds.

Consequently, the definition of CPT-formulae can be recovered from our definition of CLogspace by the following modifications: The terms of the form "$\mathrm{Atoms}\,.\varphi$" are replaced by an atomic term "Atoms", defining the full set of atoms from the input structure. The rule for recursion formulae is omitted. In terms (t, f) and formulae (φ, f), the function f is a polynomial.

In the definition of the semantics, the size annotations are eliminated completely. The polynomial bound for iteration terms is applied to the cardinality of the transitive closure of the current stage. For the computation to remain within polynomial time, the number of stages is also bounded by the polynomial. The value of an iteration term is the stage inducing the fixed-point instead of the accumulation of all intermediate stages. For more detailed definitions of Choiceless Polynomial Time, we refer to, for instance, [25, 9].

3 Inclusion in Logspace and CPT

We next prove the two essential properties of our logic that justify it to be called Choiceless Logspace: Firstly, every formula in our logic can be evaluated in logarithmic space. Secondly, it is choiceless in the sense that every formula can be translated to Choiceless Polynomial Time, the prototypical model of choiceless computation.

▶ **Theorem 14.** *Every query definable in* CLogspace *is* LOGSPACE*-computable.*

The explicit logarithmic bound in CLogspace-sentences is intended to enable evaluation in LOGSPACE. It ensures that every value occurring during an iterated computation has a sufficiently small size annotation. The size annotations induced by the semantics of terms and formulae are defined with a certain encoding of hereditarily finite sets in mind. We first define this encoding, which is the core of our evaluation algorithm.

Recall that elements of the input structure are always defined by means of terms $\mathrm{Atoms}\,.\varphi$. Since the algorithm will take as input string encodings of structures, we now operate on ordered structures. Hence one can represent an atom as "the ith entry of the kth tuple satisfying φ" using the linear order on the input structure. We denote the set of k-tuples from a structure $\mathbf{A}$ satisfying an FO-formula φ with k free variables by $\varphi^{\mathbf{A}}$.

▶ **Definition 15** (Representations)**.** *Let $\Phi \subseteq \mathrm{FO}[\sigma]$ for some vocabulary σ. The alphabet Σ_Φ consists of 0,1, "(", ")", "{", "}", "," and an alphabet symbol for each $\varphi \in \Phi$, which we also denote by φ. A* Φ-representation *is a word in the alphabet Σ_Φ that is either a set representation or an atom representation:*

- *An* atom representation *is a word (φ, i, m), where $\varphi \in \Phi$ with $k \geq 1$ free variables, and i and m are binary encodings of natural numbers $\leq k$ and $\leq |\varphi^{\mathbf{A}}|$, respectively. If $|\varphi^{\mathbf{A}}| = 0$, then m is the empty string.*
- *If $r_1, \dots, r_k$ are representations, then the word $\{r_1, \dots, r_k\}$ is a* set representation*.*

Let $\mathbf{A}$ be the expansion of a σ-structure $\mathbf{A}'$ by a linear order $<^{\mathbf{A}}$. The value *$r^{\mathbf{A}}$ of a representation r is defined as*

- *$(\varphi, i, m)^{\mathbf{A}} = a_i$ where $(a_1, \dots, a_k)$ is the m-th tuple (w.r.t. the lexicographical order extending $<$) in $\varphi^{\mathbf{A}}$.*
- *$\{r_1, \dots, r_k\}^{\mathbf{A}} = \{r_1^{\mathbf{A}}, \dots, r_k^{\mathbf{A}}\}$.*

A representation r is $\mathbf{A}$-minimal if

1. *for every atom a in $\mathrm{tc}(r^{\mathbf{A}})$, there is a unique atom representation of a that occurs as a substring of r,*
2. *if $r = \{r_1, \dots, r_k\}$, then $r_1, \dots, r_k$ are minimal representations and the values $r_i^{\mathbf{A}}$ are pairwise distinct.*

For an $\mathbf{A}$*-minimal representation* r*, we define the size annotation* $\mathrm{ann}_r^{\mathbf{A}}$ *by induction on the representation* r*:*

- $\mathrm{ann}^{\mathbf{A}}_{(\varphi,i,m)} \colon (\varphi, i, m)^{\mathbf{A}} \mapsto \log |\varphi^{\mathbf{A}}|$,
- $\mathrm{ann}^{\mathbf{A}}_{\{r_1 \dots r_k\}}$ *is the size annotation of* $\{(r_1^{\mathbf{A}}, \mathrm{ann}^{\mathbf{A}}_{r_1}), \dots, (r_k^{\mathbf{A}}, \mathrm{ann}^{\mathbf{A}}_{r_k})\}_{\mathrm{SHF}}$.

We say that r *is a representation of* $(r^{\mathbf{A}}, \mathrm{ann}_r^{\mathbf{A}}) \in \mathrm{SHF}(A)$*.*

We omit the structure $\mathbf{A}$ and speak about minimal representations if $\mathbf{A}$ can be inferred from the context. Note that there are representations that are not minimal, since the same set may be listed twice or an atom may occur encoded by different atom representations throughout the transitive closure of the value. Thus minimal representations ensure that the size annotation is well-defined. Note further that the size annotation does not agree with the word length of the representation. It does, however, provide an upper bound on the word length up to constant factors. Since the size annotation does not depend on the specific numbers i and m, this upper bound is independent of the linear order. Moreover, the size annotation of any given minimal representation is Logspace-computable.

Towards the evaluation algorithm, we show how to check whether two representations have the same value and, using the former as a subroutine, how to compute minimal representations.

▶ **Lemma 16.** *There is an algorithm that, given a* $\sigma \cup \{<\}$*-structure* $\mathbf{A}$ *(where* $<^{\mathbf{A}}$ *is obtained from the string representation of* $\mathbf{A}$*) and* $\mathbf{A}$*-minimal representations* r_1, r_2*, decides in logarithmic space whether* $r_1^{\mathbf{A}} = r_2^{\mathbf{A}}$*.*

Proof. We reduce equality of representations to isomorphism of coloured trees, where the colours are numbers $1, \dots, |A|$. An atom representation is encoded by a single node coloured by its value, and a set representation is encoded by a tree where the subtrees below the root encode the elements of the set representation. Note that, since the representations are minimal, every node has at most one child of any isomorphism type, so equality of the representations indeed implies isomorphism of the corresponding trees. ◀

▶ **Lemma 17.** *There is an algorithm that, given a* $\sigma \cup \{<\}$*-structure* $\mathbf{A}$ *(where* $<^{\mathbf{A}}$ *is obtained from the encoding of* $\mathbf{A}$*) and* $\mathbf{A}$*-minimal representations* $r_1, \dots, r_k$*, computes in logarithmic space a minimal representation* r *of the size-annotated set* $\{(r_1^{\mathbf{A}}, \mathrm{ann}^{\mathbf{A}}_{r_1}), \dots, (r_k^{\mathbf{A}}, \mathrm{ann}^{\mathbf{A}}_{r_k})\}_{\mathrm{SHF}}$*.*

Proof. We sequentially execute two Logspace-algorithms. The first one copies those representations to the output tape whose value did not occur before (using the algorithm from the previous lemma). The second algorithm replaces every atom representation by the shortest representation of its value occurring in the original string. ◀

With these subroutines, the evaluation of terms and formulae becomes rather straightforward. To simplify the statement of the following lemma, we do not distinguish between terms and formulae, but assume formulae to be terms with values $\emptyset$ and $\{\emptyset\}$.

▶ **Lemma 18.** *For every term* t *with free variables* $x_1, \dots, x_k$ *and every function* $f \colon n \mapsto c \log n$*, there is a* Logspace *algorithm that, given a* σ*-structure* $\mathbf{A}$ *and representations* $r_1, \dots, r_k$ *of* $\alpha_1, \dots, \alpha_k \in \mathrm{SHF}(A)$*, computes a representation of* $(t, f)^{\mathbf{A}, \beta \colon x_i \mapsto \alpha_i}$*.*

Proof. We proceed by induction on the construction of terms. Evaluation of atomic terms $\emptyset$ and x is trivial. For terms of the form $\mathrm{Atoms}\,.\varphi$, it suffices to compute the number n of tuples satisfying the FO-formulae φ and enumerate all tuples with entries (φ, i, m) for $0 \le m \le n$.

In the induction step, most cases follow directly from the induction hypothesis, using the fact that LOGSPACE is closed under sequential execution and thus the output of previous computations can be processed further regardless of its size. For Pair, Union and comprehension, the results of the subcomputations can be converted to a minimal representation with the algorithm from Lemma 17. Formulae $t_1 = t_2$ and $t_1 \in t_2$ can be evaluated using the algorithm from Lemma 16. For recursion formulae, the lemma follows from inclusion of LREC in LOGSPACE.

Let $[t_u]^*$ be an iteration term with free variable x_j. In its ith iteration, the algorithm computes a representation of $(t^i)^{\mathbf{A},\beta}$ from $(t^{i-1})^{\mathbf{A},\beta}$ (which is possible by induction hypothesis) and checks whether it satisfies the size bound. This is again possible using sequential execution. The new stage is compared to the previous one. Whenever a new value has been produced, it is additionally written to the output tape. A postprocessing step converts the result to a minimal representation and replaces it by the empty set if the bound has been exceeded at some point.

The bound on the size of each stage induces a logarithmic bound on the word length of the representations that are written to the work tape. In particular, this implies that the number of possible representations occurring as stages is polynomial. This ensures termination, since a counter can track the number of values that have been produced. ◀

We have established that CLogspace is a fragment of LOGSPACE. To verify that this fragment is *choiceless*, we embed it in CPT.

▶ **Theorem 19.** CLogspace $\leq$ CPT.

The straightforward way to prove the theorem is to inductively translate CLogspace-formulae to CPT. But the translation of formulae with free variables bears two technical difficulties: Firstly, CLogspace is evaluated over size-annotated hereditarily finite set. Secondly, standard definitions of CPT do not allow free variables in iteration terms.

We can, however, assume a non-standard variant of CPT where iteration terms may have free variables. This does not increase its expressive power because iteration terms can be simulated for all possible values of the free variables at once without exceeding the polynomial bound.

We address the first difficulty by the key concept of the translation to CPT: Expressing size annotations as hereditarily finite sets.

▶ **Definition 20.** *Let $\mathbf{A}$ be a σ-structure, and let $(a, s) \in \mathrm{SHF}(A)$. The* set representation $\mathrm{set}((a, s)) \in \mathrm{HF}(A)$ *is the set $\langle a, \{\langle b, [s(b)]\rangle : b \in \mathrm{tc}(a) \cap A\}\rangle$, where $[s(b)]$ is the ordinal corresponding to $s(b)$. We say that $\{\langle b, s(b)\rangle : b \in \mathrm{tc}(a) \cap A\}$* represents the size annotation s.

Proof of Theorem 19. Set representations of size annotations induce a translation between variable assignments, and thus a meaningful definition of translation between formulae with free variables. It remains to show that every CLogspace-formula can be (inductively) translated to CPT in that sense.

Most cases follow directly from the induction hypothesis, using CPT-terms that define combinations of size annotations. Size annotations are combined by using the minimal known size for every atom, which is definable in CPT.

In case a size annotation is not derived from the subterms, it originates from a term of the form "Atoms $.\varphi$". So the corresponding CPT-term has to define the size annotation that assigns the value $\log |\varphi^{\mathbf{A}}|$ to every atom from the structure $\mathbf{A}$ occurring in the value of the term. This is possible since logarithms are CPT-definable.

Further, CPT can check the logarithmic bound of iteration terms using the size annotation at each stage. Finally, recursion formulae can be translated to CPT because LREC $\leq$ FPC $\leq$ CPT. This concludes the proof of Theorem 19. ◀

4 The expressive power of CLogspace

To substantiate the idea that the rich syntax of CLogspace leads to greater expressive power, we compare it to known logics below LOGSPACE. As classical benchmarks, we consider the logics DTC and STC, which augment FO by deterministic and symmetric transitive closure operators. For precise definitions, we refer to [14, Sect. 3.5.2]. These logics are known [17] to be included in LREC, which is the strongest logic inside LOGSPACE known so far. Hence our main theorem of this kind is:

▶ **Theorem 21.** CLogspace $\gneq$ LREC.

LREC is a logic with counting terms, so it features a linearly ordered number sort and terms defining the cardinality of definable sets of tuples. Its core is the recursion operator, a modified version of which appears in our logic. We describe the lrec-operator in terms of the differences to our version, and refer to [17] for the full definition. Recall that the recursion operator interprets a graph in the input structure. In the original version, the elements of that graph are k-tuples from the input structure instead of hereditarily finite sets. Consequently, the domain term t_δ does not occur, and the subformulae can have multiple free variables. Further, the labels of the graph are defined as tuples over the number sort instead of bitsets. Both for counting of tuples and defining the labels of the graph, LREC uses a LOGSPACE-computable encoding of numbers polynomial in the size of the input structure by fixed-size tuples over the number sort. To verify that the LOGSPACE-computable arithmetic operations used in LREC are CLogspace-definable, we show:

▶ **Proposition 22.** *Let $R \subseteq \{0, \dots, |A|^k\}^\ell$ be any* LOGSPACE*-computable relation. Then there is a* CLogspace*-term defining* $\{\langle \mathrm{bitset}(n_1), \dots, \mathrm{bitset}(n_\ell)\rangle : (n_1, \dots, n_\ell) \in R\}$.

Proof. By implementing basic bit operations in CLogspace, the successor of any natural number is definable in the bitset encoding. This makes it possible to define a linear order on all bitsets up to any definable number. The elements of the linear order are bitsets of logarithmic size, so DTC-formulae can be simulated on that order analogously to Example 13. As DTC captures LOGSPACE on ordered structures, the proposition follows. ◀

The different encodings of numbers also pose a difficulty when translating formulae with free variables, since these variables can be domain or number variables. Defining a translation between variable assignments, inclusion of LREC in CLogspace can be made precise on the level of formulae with free variables. The translation to CLogspace is then rather straightforward. In particular, the only necessary iteration terms are those defining the operations on the number sort, so there exists an appropriate logarithmic bound.

So LREC is included in CLogspace. The inclusion is strict because of a property that CLogspace shares with CPT but not with traditional logics such as FPC and LREC: It benefits from padding of the input structure, i.e. it can define all LOGSPACE-properties of sufficiently small, definable substructures.

▶ **Definition 23.** *A $\sigma \cup \{U\}$-structure* $\mathbf{A}$ *is a* padded σ-structure *if $U \notin \sigma$ is a unary predicate. The* underlying structure *of* $\mathbf{A}$ *is the σ-reduct of $U^{\mathbf{A}}$.*

▶ **Theorem 24.** *Let $\mathcal{C}$ be the class of padded structures such that $2^{u!(u^2(6\log u+2)+1)} \leq n$ for every structure in $\mathcal{C}$ of size n with underlying structure of size u. For every* Logspace*-computable property $\mathcal{P}$, there is a* CLogspace*-sentence (φ, f) such that any structure in $\mathcal{C}$ satisfies (φ, f) if, and only if, the underlying structure satisfies $\mathcal{P}$.*

Proof. We define the set of all linear orders on the underlying structure. Then, since DTC $\leq$ LREC $\leq$ CLogspace and DTC captures Logspace on ordered structures, every Logspace-computable property can be defined on the resulting ordered structures.

Initially, all linear orders on two elements are created, defining pairs of atoms with the term $\text{Atoms}\,.(Ux \wedge Uy \wedge x \neq y)$. Then an iteration term extends every linear order by every atom that does not occur yet, defined by $\text{Atoms}\,.Uy$. It remains to show that there is a logarithmic bound for the iteration term. Its value in the ith stage is the set of all orders of the form $\{\langle a, b\rangle : a, b \in V : a < b\}$ for subsets $V \subseteq U$ of size $i + 2$. All atoms are defined by the terms $\text{Atoms}\,.Ux$ and $\text{Atoms}\,.(Ux \wedge Uy \wedge x \neq y)$, so every atom is annotated with a size $\leq 2\log|U|$. Then the pair $\langle a, b\rangle = \{a, \{a, b\}\}$ is mapped to $6\log|U| + 2$. Every linear order contains $< |V|^2 \leq |U|^2$ many pairs of that form, and there are $|V|! \leq |U|!$ many linear orders in the ith iteration. So $\|t^i\|^{\mathbf{A},\beta} \leq (|U|^2 \cdot (6\log|U| + 2) + 1) \cdot |U|! + 1$. By assumption, this is logarithmic in the size of the full input structure. ◀

Since the CFI-query is in Logspace, it is CLogspace-definable on padded structures. But LREC is included in FPC [18], so it cannot define this query even in the presence of padding. Thus Theorem 24 implies that CLogspace is strictly more expressive than LREC. It follows that also DTC and STC, and their extensions with counting, are strictly included in CLogspace. The same holds for all models of choiceless computation with a constant bound on the number of objects in the transitive closure of a set.

Even though our results demonstrate that CLogspace is stronger than all previously studied logics for Logspace, we can show that it does not capture all of Logspace. To establish this result we use a technique for proving inexpressibility results for fragments of CPT that is based on *supports* of hereditarily finite objects (see e.g. [3, 8, 26]).

▶ **Definition 25.** *Let $\text{Aut}(\mathbf{A})$ be the automorphism group of a finite structure $\mathbf{A}$. For $a \in \text{HF}(A)$, let $\text{Stab}(a)$ be the stabiliser of a with respect to $\text{Aut}(\mathbf{A})$, and let $\text{Stab}^\bullet(a)$ be its point-wise stabiliser. An object $s \in \text{HF}(A)$ is a* support *of a if $\text{Stab}^\bullet(s) \subseteq \text{Stab}(a)$.*

Every CPT-formula can be translated to an FPC-formula over a substructure of $\text{HF}(\mathbf{A})$ containing the sets *activated* by the formula. For general CPT-formulae, this can be an arbitrary substructure of $\text{HF}(\mathbf{A})$, depending on both the CPT-formula and the input structure. But for some fragments of CPT, the substructure can be over-approximated by those hereditarily finite objects that have a sufficiently small support. In particular, CLogspace-formulae can be translated to FPC-formulae that are evaluated over hereditarily finite sets with a support of logarithmic size.

▶ **Lemma 26.** *For a structure $\mathbf{A}$ and $f : \mathbb{N} \to \mathbb{N}$, let $\text{HF}_f(\mathbf{A})$ be the substructure of $\text{HF}(\mathbf{A})$ containing exactly those sets that have a support of size $f(n)$, where n is the size of the domain of $\mathbf{A}$. For every* CLogspace*-formula (φ, f), there is an* FPC*-formula ψ such that, for every structure $\mathbf{A}$, $\mathbf{A} \models (\varphi, f)$ if, and only if, $\text{HF}_f(\mathbf{A}) \models \psi$.*

Proof. Since the stages of all iteration terms are of size at most $f(n)$ with respect to their size annotations, they are elements of $\text{HF}_f(\mathbf{A})$. Analogously to the translation of CPT-terms to interpretations in an extension of FO in [10], every ordinary term or formula in CLogspace can be translated to an LREC-interpretation. Iteration terms can be translated to FPC-interpretations that create new objects representing the value of the iteration term, i.e. the aggregation of all stages. The element relation between that new element and the stages can then be defined by a fixed-point formula, maintaining the size annotations as a separate relation. ◀

By the same arguments as used in the proof of Theorem 40 in [8], this implies that the Cai-Fürer-Immerman query cannot be defined in CLogspace. As a consequence we get

▶ **Theorem 27.** CLogspace *is a strict fragment of* LOGSPACE.

5 Discussion

We have introduced Choiceless Logarithmic Space formalising the notion of symmetric, choiceless computations using only logarithmic workspace. Through the notion of size-annotated hereditarily finite sets, our logic takes into account that sizes of objects in LOGSPACE are sensitive to their encoding. The logic is a fragment of both Choiceless Polynomial Time and LOGSPACE, and it captures LOGSPACE on certain classes of structures with padding. This demonstrates the similarity to CPT, and, more importantly, makes it stronger than all previously known logics in LOGSPACE, in particular LREC.

However, we have seen that CLogspace does not capture LOGSPACE because it can only define sets with a support of logarithmic size, which, as shown in [8], makes it impossible to define the Cai-Fürer-Immerman query. It remains open whether the concepts used in CLogspace can be used to obtain a stronger logic that could capture all queries in LOGSPACE.

Of course, our definition can be tuned in several ways, addressing issues such as closure under interpretations, or trying to avoid the use of two different kinds of recursion operators (iteration terms and recursion formulae). Even for the current definition of CLogspace, it is open whether the lrec-operator can be eliminated without losing expressive power. It can, however, be shown that iteration terms are necessary (see [26]). However, contrary to polynomial time, logspace complexity is a much less robust notion, and the conflicting goals of providing sufficient power for choiceless computation and remaining inside LOGSPACE necessarily seem to result in a rather involved construction of CLogspace. It seems difficult to overcome, with further tunings of the definition, the barrier that the definable sets in CLogspace have a (far too) small support. In the light of the research carried out here, we therefore consider it improbable that some variant of choiceless computation can capture precisely all (symmetry-invariant) LOGSPACE-queries.

We may take a step back, and reconsider our general objectives. One of the reasons why we are looking for a logic that captures logspace complexity on arbitrary finite structures is that LOGSPACE, other than being a well-studied complexity class for standard computation models on ordered objects, is a reasonable formalisation of efficient computation for big data. But there are also other, less restrictive, complexity classes that can serve a similar purpose, for instance on the basis of polylogarithmic space bounds. Such variants may be more robust and make it possible to assume a standard encoding of atoms with small space and might grant more freedom in defining tree-like recursion using only iteration terms. Rather than trying to tune the definition of CLogspace to find stronger notions of choiceless computation inside LOGSPACE, we may thus also go beyond the logspace bound and ask more generally: *"What is efficient choiceless computation for big data?"*.

References

1 F. Abu Zaid, E. Grädel, M. Grohe, and W. Pakusa. Choiceless Polynomial Time on structures with small Abelian colour classes. In *MFCS 2014*, volume 8634 of *Lecture Notes in Computer Science*, pages 50–62. Springer, 2014.

2 A. Atserias, A. Bulatov, and A. Dawar. Affine Systems of Equations and Counting Infinitary Logic. *Theoretical Computer Science*, 410:1666–1683, 2009.

3 A. Blass, Y. Gurevich, and S. Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100(1-3), 1999.

4 J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992.

5 A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25(1), 1982.

6 A. Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1):8–21, 2015.

7 A. Dawar, M. Grohe, B. Holm, and B. Laubner. Logics with Rank Operators. In *Proc. 24th IEEE Symp. on Logic in Computer Science (LICS 09)*, pages 113–122, 2009.

8 A. Dawar, D. Richerby, and B. Rossman. Choiceless Polynomial Time, Counting and the Cai-Fürer-Immerman Graphs. *Annals of Pure and Applied Logic*, 152:31–50, 2009.

9 E. Grädel and M. Grohe. Is Polynomial Time Choiceless? In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, pages 193–209, 2015.

10 E. Grädel, Ł. Kaiser, W. Pakusa, and S. Schalthöfer. Characterising Choiceless Polynomial Time with First-Order Interpretations. In *LICS*, 2015.

11 E. Grädel and M. Otto. Inductive Definability with Counting on Finite Structures. In *Computer Science Logic, CSL 92*, volume 702 of *Lecture Notes in Computer Science*, pages 231–247. Springer, 1992.

12 E. Grädel and W. Pakusa. Rank logic is dead, long live rank logic! *Journal of Symbolic Logic*, To appear, 2019. http://arxiv.org/abs/1503.05423.

13 E. Grädel and M. Spielmann. Logspace Reducibility via Abstract State Machines. In J. Wing, J. Woodcock, and J. Davies, editors, *World Congress on Formal Methods (FM '99)*, volume 1709 of *LNCS*. Springer, 1999. URL: `http://www.logic.rwth-aachen.de/pub/graedel/GrSp-fm99.ps`.

14 E. Grädel et al. *Finite Model Theory and Its Applications*. Springer-Verlag, 2007.

15 M. Grohe. The quest for a logic capturing PTIME. In *Proceedings of the 23rd IEEE Symposium on Logic in Computer Science (LICS'08)*, pages 267–271, 2008.

16 M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Cambridge University Press, 2017.

17 M. Grohe, B. Grußien, A. Hernich, and B. Laubner. L-Recursion and a new Logic for Logarithmic Space. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 12. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2011.

18 B. Grußien. *Capturing Polynomial Time and Logarithmic Space using Modular Decompositions and Limited Recursion*. PhD thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät, 2017. `doi:10.18452/18548`.

19 Y. Gurevich. Logic and the challenge of computer science. In E. Börger, editor, *Current Trends in Theoretical Computer Science*, pages 1–57. Computer Science Press, 1988.

20 N. Immerman. Expressibility as a complexity measure: results and directions. In *Structure in Complexity Theory*, pages 194–202, 1987.

21 N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, 1987.

22 W. Pakusa. *Linear Equation Systems and the Search for a Logical Characterisation of Polynomial Time*. PhD thesis, RWTH Aachen University, 2016.

23 W. Pakusa, S. Schalthöfer, and E. Selman. Definability of Cai-Fürer-Immerman Problems in Choiceless Polynomial Time. In *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, pages 19:1–19:17, 2016.

24 O. Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008.

25 B. Rossman. Choiceless Computation and Symmetry. In A. Blass, N. Dershowitz, and W. Reisig, editors, *Fields of Logic and Computation: Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume 6300 of *LNCS*, pages 565–580. Springer, 2010.

26 S. Schalthöfer. *Choiceless Computation and Logic.* PhD thesis, RWTH Aachen University, 2018.

27 M. Spielmann. *Abstract state machines: Verification problems and complexity.* PhD thesis, RWTH Aachen University, 2000.

Faster FPT Algorithm for 5-Path Vertex Cover

Radovan Červený
Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Prague, Czech Republic
radovan.cerveny@fit.cvut.cz

Ondřej Suchý
Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Prague, Czech Republic
ondrej.suchy@fit.cvut.cz

Abstract

The problem of d-Path Vertex Cover, d-PVC lies in determining a subset F of vertices of a given graph $G = (V, E)$ such that $G \setminus F$ does not contain a path on d vertices. The paths we aim to cover need not to be induced. It is known that the d-PVC problem is NP-complete for any $d \geq 2$. When parameterized by the size of the solution k, 5-PVC has direct trivial algorithm with $\mathcal{O}(5^k n^{\mathcal{O}(1)})$ running time and, since d-PVC is a special case of d-Hitting Set, an algorithm running in $\mathcal{O}(4.0755^k n^{\mathcal{O}(1)})$ time is known. In this paper we present an iterative compression algorithm that solves the 5-PVC problem in $\mathcal{O}(4^k n^{\mathcal{O}(1)})$ time.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases graph algorithms, Hitting Set, iterative compression, parameterized complexity, d-Path Vertex Cover

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.32

Related Version `https://arxiv.org/abs/1906.09213`

Funding *Radovan Červený*: Supported by grant 17-20065S of the Czech Science Foundation.
Ondřej Suchý: The author acknowledges the support of the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

1 Introduction

The problem of d-Path Vertex Cover, d-PVC lies in determining a subset F of vertices of a given graph $G = (V, E)$ such that $G \setminus F$ does not contain a path on d vertices (even not a non-induced one). The problem was first introduced by Brešar et al. [1], but its NP-completeness for any $d \geq 2$ follows already from the meta-theorem of Lewis and Yannakakis [11]. The 2-PVC problem corresponds to the well known Vertex Cover problem and the 3-PVC problem is also known as Maximum Dissociation Set. The d-PVC problem is motivated by the field of designing secure wireless communication protocols [12] or in route planning and speeding up shortest path queries [9].

Since the problem is NP-hard, any algorithm solving the problem exactly is expected to have exponential running time. If one measures the running time solely in terms of the input size, then several efficient (faster than trivial enumeration) exact algorithms are known for 2-PVC and 3-PVC. In particular, 2-PVC (Vertex Cover) can be solved in $\mathcal{O}(1.1996^n)$ time and polynomial space due to Xiao and Nagamochi [20] and 3-PVC can be solved in $\mathcal{O}(1.4656^n)$ time and polynomial space due to Xiao and Kou [18].

In this paper we aim on the parameterized analysis of the problem, that is, to confine the exponential part of the running time to a specific parameter of the input, presumably much smaller than the input size. Algorithms achieving running time $f(k)n^{O(1)}$ are called *parameterized*, *fixed-parameter tractable*, or *FPT*. See Cygan et al. [3] for a broader introduction to parameterized algorithms.

When parameterized by the size of the solution k, the d-PVC problem is directly solvable by a trivial FPT algorithm that runs in $\mathcal{O}^*(d^k)$ time.[1] However, since d-PVC is a special case of d-Hitting Set, it was shown by Fomin et al. [6] that for any $d \geq 4$ we have an algorithm solving d-PVC in $\mathcal{O}^*((d-0.9245)^k)$. In order to find more efficient solutions, the problem has been extensively studied in a setting where d is a small constant. For the 2-PVC (Vertex Cover) problem, the algorithm of Chen, Kanj, and Xia [2] has the currently best known running time of $\mathcal{O}^*(1.2738^k)$. For 3-PVC, Tu [16] used iterative compression to achieve a running time $\mathcal{O}^*(2^k)$. This was later improved by Katrenič [10] to $\mathcal{O}^*(1.8127^k)$, by Xiao and Kou [19] to $\mathcal{O}^*(1.7485^k)$ by using a branch-and-reduce approach and it was further improved by Tsur [15] to $\mathcal{O}^*(1.713^k)$. For the 4-PVC problem, Tu and Jin [17] again used iterative compression and achieved a running time $\mathcal{O}^*(3^k)$ and Tsur [14] claims to have an algorithm with running time $\mathcal{O}^*(2.619^k)$.

We present an algorithm that solves the 5-PVC problem parameterized by the size of the solution k in $\mathcal{O}^*(4^k)$ time by employing the iterative compression technique. Using the result of Fomin et al. [7] this also yields $\mathcal{O}(1.7501^n)$ time algorithm improving upon previously known $\mathcal{O}(1.7547^n)$ time algorithm.

Organization of this paper. We introduce the notation and define the 5-PVC problem in section 2. Our disjoint compression routine for iterative compression is exposed in section 3. We conclude this paper with a few open questions. Due to space constraints most proofs were deferred to the full version of this paper. However, to showcase our technique of proving the correctness of our algorithm, we included the proofs of correctness for Rules (R10) and (R13.1), and proofs of Lemmata 18 and 21.

2 Preliminaries

We use the $\mathcal{O}^*$ *notation* as described by Fomin and Kratsch [8], which is a modification of the big-$\mathcal{O}$ notation suppressing all polynomially bounded factors. We use the notation of parameterized complexity as described by Cygan et al. [3]. We use standard graph notation and consider simple and undirected graphs unless otherwise stated. Vertices of graph G are denoted by $V(G)$, edges by $E(G)$. By $G[X]$ we denote the subgraph of G induced by vertices of $X \subseteq V(G)$. By $N(v)$ we denote the set of neighbors of $v \in V(G)$ in G. Analogically, $N(X) = \bigcup_{x \in X} N(x)$ denotes the set of neighbors of vertices in $X \subseteq V(G)$. The degree of vertex v is denoted by $deg(v) = |N(v)|$. For simplicity, we write $G \setminus v$ for $v \in V(G)$ and $G \setminus X$ for $X \subseteq V(G)$ as shorthands for $G[V(G) \setminus \{v\}]$ and $G[V(G) \setminus X]$, respectively.

A *k-path*, denoted as an ordered k-tuple $P_k = (p_1, p_2, \ldots, p_k)$, is a path on k vertices $\{p_1, p_2, \ldots, p_k\}$. A path P_k *starts* at vertex x when $p_1 = x$. A *k-cycle* is a cycle on k vertices. A *triangle* is a 3-cycle. A P_5*-free* graph is a graph that does not contain a P_5 as a subgraph (the P_5 need not to be induced). The 5-Path Vertex Cover problem is formally defined as follows:

5-Path Vertex Cover, 5-PVC	
Input:	A graph $G = (V, E)$, an integer $k \in Z_0^+$.
Output:	A set $F \subseteq V$, such that $\lvert F \rvert \leq k$ and $G \setminus F$ is a P_5-free graph.

[1] The $\mathcal{O}^*()$ notation suppresses all factors polynomial in the input size.

▶ **Definition 1.** *A star is a graph S with vertices $V(S) = \{s\} \cup \{l_1, \ldots, l_k\}$, $k \geq 3$ and edges $E(S) = \{\{s, l_i\} \mid i \in \{1, \ldots, k\}\}$. Vertex s is called a center, vertices $L = \{l_1, \ldots, l_k\}$ are called leaves.*

▶ **Definition 2.** *A star with a triangle is a graph $S^{\triangle}$ with vertices $V(S^{\triangle}) = \{s, t_1, t_2\} \cup \{l_1, \ldots, l_k\}$, $k \geq 1$ and edges $E(S^{\triangle}) = \{\{s, t_1\}, \{s, t_2\}, \{t_1, t_2\}\} \cup \{\{s, l_i\} \mid i \in \{1, \ldots, k\}\}$. Vertex s is called a center, vertices $T = \{t_1, t_2\}$ are called triangle vertices and vertices $L = \{l_1, \ldots, l_k\}$ are called leaves.*

▶ **Definition 3.** *A di-star is a graph D with vertices $V(D) = \{s, s'\} \cup \{l_1, \ldots, l_k\} \cup \{l'_1, \ldots, l'_m\}$, $k \geq 1, m \geq 1$ and edges $E(D) = \{\{s, s'\}\} \cup \{\{s, l_i\} \mid i \in \{1, \ldots, k\}\} \cup \{\{s', l'_j\} \mid j \in \{1, \ldots, m\}\}$. Vertices s, s' are called centers, vertices $L = \{l_1, \ldots, l_k\}$ and $L' = \{l'_1, \ldots, l'_m\}$ are called leaves.*

▶ **Lemma 4.** *If a connected graph is P_5-free and has more than 5 vertices, then it is a star, a star with a triangle, or a di-star.*

3 5-PVC with P_5-free bipartition

We employ the generic iterative compression framework as described by Cygan et al. [3, pages 80–81]. We skip the generic steps and only present the disjoint compression routine (see the full version of the paper for a brief discussion of the whole iterative compression algorithm). That is, we assume that we are given a solution to the problem and search for another solution which is strictly smaller than and disjoint from the given one. Moreover, if the graph induced by the given solution contains a P_5, then we can directly answer no. Hence our routine DISJOINT_R restricts itself to a problem called 5-PVC WITH P_5-FREE BIPARTITION and we need it to run in $\mathcal{O}^*(3^k)$ time.

A *P_5-free bipartition* of graph $G = (V, E)$ is a pair (V_1, V_2) such that $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$ and $G[V_1], G[V_2]$ are P_5-free. The 5-PVC WITH P_5-FREE BIPARTITION problem is formally defined as follows:

5-PVC WITH P_5-FREE BIPARTITION, 5-PVCWB	
INPUT:	A graph $G = (V, E)$ with P_5-free bipartition (V_1, V_2), an integer $k \in Z_0^+$.
OUTPUT:	A set $F \subseteq V_2$, such that $\lvert F \rvert \leq k$ and $G \setminus F$ is a P_5-free graph.

Throughout this paper the vertices from V_1 will be also referred to as “red” vertices and vertices from V_2 will be also refereed to as “blue” vertices.

3.1 Algorithm

Our algorithm is a recursive procedure DISJOINT_R(G, V_1, V_2, F, k), where G is the input graph, V_1, V_2 are the partitions of the P_5-free bipartition of G, F is the solution being constructed, and k is the maximum number of vertices we can still add to F. The procedure repeatedly tries to apply a series of rules with a condition that a rule (RI) can be applied only if all rules that come before (RI) cannot be applied. It is paramount that in every call of DISJOINT_R at least one rule can be applied. The main work is done in rules of two types: *reduction rules* and *branching rules*. To make it easier for the reader we also use rules called *context rules*, which only describe the configuration we are currently in and serve as some sort of a parent rules for their subrules.

A *reduction rule* is used to simplify a problem instance, i.e. remove some vertices or edges from G and possibly add some vertices to a solution, or to halt the algorithm. A *branching rule* splits the problem instance into at least two subinstances. The branching is based on subsets of vertices that we try to add to a solution and by adding them to the solution we also remove them from G.

The notation we use to denote the individual branches of a branching rule is as follows: $\langle\, X_1 \mid X_2 \mid \ldots \mid X_l \,\rangle$. Such a rule has l branches and $X_1, X_2, \ldots, X_l$ are subsets of V_2 which we try to add to the solution. This rule is translated into the following l calls of the procedure:

$$\text{DISJOINT\_R}(G \setminus X_i, V_1, V_2 \setminus X_i, F \cup X_i, k - |X_i|) \text{ for } i \in \{1, \ldots, l\}$$

A rule is *applicable* if the conditions of the rule are satisfied and none of the previous rules is applicable. If a context rule is not applicable, it means that none of its subrules is applicable.

A reduction rule is *correct* if it satisfies that the problem instance has a solution if and only if the simplified problem instance has a solution. A branching rule is *correct* if it satisfies that the problem instance has a solution if and only if at least one of the branches of the rule will return a solution.

When we say we *delete* a vertex, we mean that we remove it from G and also add it to the solution F. When we say we *remove* a vertex, we mean that we remove it from G and *do not* add it to the solution F.

For the rest of this paper assume that the parameters of the current call of DISJOINT_R are G, V_1, V_2, F, k.

3.2 Preprocessing

▶ **Reduction rule (R0).** This rule stops the recursion of DISJOINT_R. It has three stopping conditions:

1. If $k < 0$, return *no solution*;
2. else if G is P_5-free, return F;
3. else if $k = 0$, return *no solution.*

▶ **Reduction rule (R1).** Let $v \in V(G)$ be a vertex such that there is no P_5 in G that uses v. Then remove v from G.

▶ **Branching rule (R2).** Let P be a P_5 in G with $X = V(P) \cap V_2$ such that $|X| \le 3$. Then branch on $\langle\, x_1 \mid x_2 \mid \ldots \rangle, x_i \in X$, i.e. branch on the blue vertices of P.

▶ **Lemma 5.** *Assume that Rules (R0) – (R2) are not applicable. Then for each vertex $v \in V(G)$ there exists a P_5 in G that uses v; every P_5 in G uses exactly one red vertex; and there are only isolated vertices in $G[V_1]$.*

3.3 Dealing with isolated vertices in $G[V_2]$

▶ **Lemma 6.** *Assume that Rules (R0) – (R2) are not applicable. Let v be an isolated vertex in $G[V_2]$ and let F be a solution to* 5-PVCwB *which uses vertex v. Then there exists a solution F' that does not use vertex v and $|F'| \le |F|$.*

▶ **Branching rule (R3).** Let v be an isolated vertex in $G[V_2]$ and let $P = (v, w, x, y, z)$ be a P_5 where w is a red vertex. Then branch on $\langle\, x \mid y \mid z \,\rangle$.

▶ **Lemma 7.** *Assume that Rules (R0) – (R3) are not applicable. Then there are no isolated vertices in $G[V_2]$.*

3.4 Dealing with isolated edges in $G[V_2]$

▶ **Lemma 8.** *Assume that Rules (R0) – (R3) are not applicable. Let v be a blue vertex to which at least two red vertices are connected and let C_v be a connected component of $G[V_2]$ which contains v. Then for each red vertex w connected to v we have that $N(w) \subseteq V(C_v)$.*

▶ **Lemma 9.** *Assume that Rules (R0) – (R3) are not applicable. Let $e = \{u, v\}$ be a blue edge to which at least two red vertices are connected in a way that to both u and v there is at least one red vertex connected. Let C_e be a connected component of $G[V_2]$ which contains e. Then for each red vertex w connected to e we have that $N(w) \subseteq V(C_e)$.*

▶ **Lemma 10.** *Let X be a subset of V_2 such that $N(X) \cap V_1 = \emptyset$ and $|N(X) \cap V_2| = 1$. If there exists a solution F such that $F \cap X \neq \emptyset$, then there exists a solution F' such that $F' \cap X = \emptyset$ and $|F'| \leq |F|$.*

▶ **Definition 11.** *We say that two nodes x, y are twins if $N(x) \setminus \{y\} = N(y) \setminus \{x\}$.*

▶ **Lemma 12.** *Let x, y be blue vertices that are twins. Let F be a solution and $x \in F$. Then at least one of the following holds:*
(1) $y \in F$,
(2) $F' = (F \setminus \{x\}) \cup \{y\}$ is a solution.

▶ **Branching rule (R4).** Let $e = \{u, v\}$ be an isolated edge in $G[V_2]$. We know from Lemmata 8 and 9 that there is only one red vertex w connected to e, because if there were at least two red vertices connected to e, then there would be no P_5 that uses vertices from e. Let there be a red vertex w connected to at least one vertex in e. If w is connected only to one vertex in e, let that vertex be v. Assume that x is some vertex to which w connects outside e and let y be a neighbor of x in $G[V_2]$. Then branch on $\langle\, v \mid x \mid y \,\rangle$.

▶ **Lemma 13.** *Assume that Rules (R0) – (R4) are not applicable. Then there are no isolated edges in $G[V_2]$.*

3.5 Dealing with isolated P_3 paths in $G[V_2]$

▶ **Context rule (R5).** Let P be a $P_3 = (t, u, v)$ that forms a connected component in $G[V_2]$. From Lemmata 5, 8 and 9 we know that there is only one red vertex w connected to P. We further know that w must be connected to some component of $G[V_2]$ other than P, otherwise no P_5 could be formed. Assume that x is some vertex to which w connects outside P and let y be a neighbor of x in $G[V_2]$. This rule is split into four subrules (R5.1), (R5.2), (R5.3) and (R5.4) based on how w is connected to P.

▶ **Branching rule (R5.1).** Vertex w is connected only to v in P. Then branch on $\langle\, v \mid x \,\rangle$.

▶ **Branching rule (R5.2).** Vertex w is connected only to u, v in P. Then branch on $\langle\, u \mid v \mid x \,\rangle$.

▶ **Branching rule (R5.3).** Vertex w is connected only to u in P. Then branch on $\langle\, u \mid x \mid y \,\rangle$.

▶ **Branching rule (R5.4).** Vertex w is connected to t, v in P and w can be also connected to u in P. Then branch on $\langle\, u \mid v \mid x \,\rangle$.

▶ **Lemma 14.** *Assume that Rules (R0) – (R5) are not applicable. Then there are no isolated P_3 paths in $G[V_2]$.*

3.6 Dealing with isolated triangles in $G[V_2]$

▶ **Context rule (R6).** Let T be a $K_3 = \{t, u, v\}$ that forms a connected component in $G[V_2]$. From Lemmata 5 and 8 we know that there is only one red vertex w connected to T. We further know that w must be connected to some component of $G[V_2]$ other than T, otherwise no P_5 could be formed. Assume that x is some vertex to which w connects outside T and let y be a neighbor of x in $G[V_2]$. This rule is split into three subrules (R6.1), (R6.2) and (R6.3) based on how w is connected to T.

▶ **Branching rule (R6.1).** Vertex w is connected only to one vertex in T, let that vertex be v. Then branch on $\langle\, v \mid x \,\rangle$.

▶ **Branching rule (R6.2).** Vertex w is connected to exactly two vertices in T, let those vertices be u, v. Then branch on $\langle\, t \mid v \mid x \,\rangle$.

▶ **Branching rule (R6.3).** Vertex w is connected to all vertices in T. Then branch on $\langle\, v \mid x \,\rangle$.

▶ **Lemma 15.** *Assume that Rules (R0) – (R6) are not applicable. Then there are no isolated triangles in $G[V_2]$.*

3.7 Dealing with 4-cycles in $G[V_2]$

▶ **Lemma 16.** *Let C be a connected component of $G[V_2]$ and $X = V(C) \cap N(V_1)$. Let F be a solution that deletes at least $|X|$ vertices in C. Then $F' = (F \setminus V(C)) \cup X$ is also a solution and $|F'| \leq |F|$.*

▶ **Context rule (R7).** Let Q be a connected component in $G[V_2]$ such that Q is a subgraph of K_4 and a 4-cycle is a subgraph of Q, label the vertices of the 4-cycle (v_1, v_2, v_3, v_4). We will call pairs of vertices $\{v_1, v_3\}$ and $\{v_2, v_4\}$ *diagonal*, all other pairs will be called *non-diagonal*. Edges corresponding to diagonal (non-diagonal) pairs are called diagonal (non-diagonal) edges, respectively. This rule is split into two subrules (R7.1), (R7.2) based on the number of red vertices connected to Q.

▶ **Reduction rule (R7.1).** Assume that there are at least two red vertices connected to Q. Then delete any vertex v_i in Q and add it to the solution F.

▶ **Context rule (R7.2).** Assume that there is only one red vertex w connected to Q and $X = V(Q) \cap N(w)$. This rule is split into five subrules (R7.2a), (R7.2b), (R7.2c), (R7.2d) and (R7.2e) based on how w is connected to Q and whether w is connected to other components.

▶ **Reduction rule (R7.2a).** Vertex w is connected only to one vertex in Q, let it be v_1. Then delete v_1 and add it to the solution F.

▶ **Branching rule (R7.2b).** Set X contains at least one diagonal pair, let that pair be $\{v_1, v_3\}$. Then branch on $\langle\, v_1 \mid v_2 \mid v_4 \,\rangle$.

▶ **Branching rule (R7.2c).** Set X contains exactly one non-diagonal pair, let that pair be $\{v_1, v_2\}$, and case *(a)* either both diagonal edges are in Q , or case *(b)* none of them is. Then branch on $\langle\, v_1 \mid v_3 \mid v_4 \,\rangle$

▶ **Reduction rule (R7.2d).** Set X contains exactly one non-diagonal pair, let that pair be $\{v_1, v_2\}$ and exactly one diagonal edge is in Q, let that edge be $\{v_1, v_3\}$. Furthermore, w is connected only to Q, i.e. $N(w) \subseteq V(Q)$. Then delete any vertex v_i in Q and add it to the solution F.

▶ **Branching rule (R7.2e).** Set X contains exactly one non-diagonal pair, let that pair be $\{v_1, v_2\}$ and exactly one diagonal edge is in Q, let that edge be $\{v_1, v_3\}$. Furthermore, w is connected to at least one more component of $G[V_2]$ other than Q, label the vertex to which w connects outside Q as x and let y be a neighbor of x in $G[V_2]$. Then branch on $\langle\, \{v_1, v_2\} \mid x \mid y \,\rangle$.

▶ **Lemma 17.** *Assume that Rules (R0) – (R7) are not applicable. Then there is no component of $G[V_2]$ that contains a 4-cycle as a subgraph.*

3.8 Dealing with stars in $G[V_2]$

In this subsection we consider a connected component of $G[V_2]$ which is isomorphic to a star. Through this subsection we denote it by S and label its vertices as in Definition 1. We remark that, as none of the Rules (R0) – (R7) is applicable, there is exactly one red vertex connected to S.

▶ **Context rule (R8).** Let S be a star in $G[V_2]$ and let w be a red vertex connected to S. This rule is divided into three subrules (R8.1), (R8.2) and (R8.3) based on how w is connected to S.

▶ **Branching rule (R8.1).** A red vertex w is connected to at least two leaves of S, let those two leaves be l_1, l_2. Then branch on $\langle\, l_1 \mid s \mid L \setminus \{l_1, l_2\} \,\rangle$.

▶ **Branching rule (R8.2).** A red vertex w is connected only to s in S and w is connected to some other vertex x in $G[V_2]$ outside S and y is a neighbor of x in $G[V_2]$. Then branch on $\langle\, s \mid x \mid y \,\rangle$.

▶ **Branching rule (R8.3).** A red vertex w is connected to l_1 in S, w can be connected also to s in S, and w is connected to some other vertex x in $G[V_2]$ outside S. Then branch on $\langle\, s \mid l_1 \mid x \,\rangle$.

▶ **Lemma 18.** *Assume that Rules (R0) – (R8) are not applicable. Then there are no stars in $G[V_2]$.*

Proof. For contradiction assume that Rules (R0) – (R8) are not applicable and there is a star S in $G[V_2]$.

If there is no P_5 that uses vertices from S, then Rule (R1) is applicable on S. Suppose there are at least two red vertices connected to S. If the red vertices are not connected to a single vertex or a single edge in S, then Rule (R2) is applicable, since there is a P_5 that uses at least two red vertices. So suppose the red vertices are connected to a single vertex or a single edge in S. Then from Lemmata 8 and 9 we know that those red vertices are not connected to any other vertices outside S. Consequently, there cannot be a P_5 that uses vertices from S and again Rule (R1) is applicable on S.

So suppose that there is a P_5 that uses vertices from S and there is only one red vertex w connected to S. If w is connected to two leaves, then Rule (R8.1) is applicable. So suppose that w is not connected to two leaves. There are three not mutually isomorphic possibilities how w can be connected to S: $\{l_1\}$, $\{s\}$, and $\{l_1, s\}$. In those cases we apply Rules (R8.3), (R8.2), and (R8.3), respectively. ◀

3.9 Dealing with stars with a triangle in $G[V_2]$

In this subsection we consider a connected component of $G[V_2]$ which is isomorphic to a star with a triangle. Through this subsection we denote it by $S^{\triangle}$ and label its vertices as in Definition 2.

▶ **Context rule (R9).** Let $S^{\triangle}$ be a star with a triangle in $G[V_2]$ and let w be a red vertex connected to $S^{\triangle}$. This rule is divided into four subrules (R9.1), (R9.2), (R9.3) and (R9.4) based on how w is connected to $S^{\triangle}$.

▶ **Branching rule (R9.1).** There is a red vertex w such that $\{t_1, t_2\} \subseteq N(w)$. Then branch on $\langle\, t_1 \mid s \mid L \,\rangle$.

▶ **Branching rule (R9.2).** There is a red vertex w such that $|\{t_1, t_2\} \cap N(w)| = 1$, assume that w is connected to t_1. Then branch on $\langle\, t_1 \mid s \mid L \,\rangle$.

▶ **Lemma 19.** *Assume that Rules (R0) – (R9.1) are not applicable and the assumptions of Rule (R9.2) are satisfied. If F is a solution that contains t_2, then at least one of the following holds:*
(1) $t_1 \in F$,
(2) $F' = (F \setminus \{t_2\}) \cup \{t_1\}$ *is a solution.*

▶ **Branching rule (R9.3).** There is a red vertex w connected to a leaf of $S^{\triangle}$, let that leaf be l_1. Then branch on $\langle\, l_1 \mid s \,\rangle$.

▶ **Branching rule (R9.4).** A red vertex w is connected only to s in $S^{\triangle}$. Then w must be also connected to some component of $G[V_2]$ other than $S^{\triangle}$, otherwise no P_5 would occur in the component containing $S^{\triangle}$. Label the vertex to which w connects outside $S^{\triangle}$ as x. Then branch on $\langle\, s \mid x \,\rangle$.

▶ **Lemma 20.** *Assume that Rules (R0) – (R9) are not applicable. Then there are no stars with a triangle in $G[V_2]$.*

3.10 Dealing with di-stars in $G[V_2]$

In this subsection we consider a connected component of $G[V_2]$ which is isomorphic to a di-star. Through this subsection we denote it by D and label its vertices as in Definition 3.

▶ **Branching rule (R10).** Let D be a di-star in $G[V_2]$ and let there be a red vertex w connected to at least two leaves on the same side of the di-star, i.e. $|N(w) \cap L| \geq 2$ or $|N(w) \cap L'| \geq 2$. Assume that those leaves are from L and l_1, l_2 are among them. Observe that there is no other red vertex connected to l_1, l_2. Then branch on $\langle\, l_1 \mid s \mid s' \,\rangle$.

Proof of correctness. We have to delete something in $\{l_1, l_2, s, s'\}$ and since l_1, l_2 are twins, from Lemma 12 we know that we have to try only one of them, thus branching on $\langle\, l_1 \mid s \mid s' \,\rangle$ is correct. ◀

▶ **Context rule (R11).** Let D be a di-star in $G[V_2]$ and let w be the only red vertex connected to D such that $\{s, s'\} \subseteq N(w)$. This rule is split into three subrules (R11.1), (R11.2), and (R11.3) based on the degrees of s and s'.

▶ **Context rule (R11.1).** Assume that both s, s' have degree two in $G[V_2]$, i.e. the di-star D is actually a P_4. This rule is split into four subrules (R11.1a), (R11.1b), (R11.1c), and (R11.1d) based on how w is connected to D and whether w is connected to other components.

▶ **Branching rule (R11.1a).** Vertex w is connected only to s, s' in D. Then branch on $\langle\, s \mid s' \,\rangle$.

▶ **Branching rule (R11.1b).** Vertex w is connected to s, s' and to one leaf in D, let that leaf be l_1. Then branch on $\langle\, l_1 \mid s \mid s' \,\rangle$.

▶ **Branching rule (R11.1c).** Vertex w is connected to l_1, l'_1, s, s' in D and to at least one other component of $G[V_2]$, label the vertex w connects to outside D as x and the neighbor of x in $G[V_2]$ as y. Then branch on $\langle\, x \mid y \mid \{l_1, s'\} \mid \{s, l'_1\} \mid \{s, s'\}\,\rangle$.

▶ **Reduction rule (R11.1d).** Vertex w is connected only to l_1, l'_1, s, s' in D and to no other component of $G[V_2]$. Then delete any vertex v in D and add it to the solution F.

▶ **Context rule (R11.2).** Assume that exactly one of s, s' has degree at least 3 in $G[V_2]$, let it be s. This rule is split into four subrules (R11.2a), (R11.2b), (R11.2c), and (R11.2d) based on how w is connected to D.

▶ **Branching rule (R11.2a).** Vertex w is connected only to s, s' in D. Then branch on $\langle\, s \mid s'\,\rangle$.

▶ **Branching rule (R11.2b).** Vertex w is connected to s, s' and exactly one leaf from L in D, let that leaf be l_1. Then branch on $\langle\, l_1 \mid s \mid s'\,\rangle$.

▶ **Branching rule (R11.2c).** Vertex w is connected to s, s', l'_1 in D. Then branch on $\langle\, l'_1 \mid s \mid s'\,\rangle$.

▶ **Branching rule (R11.2d).** Vertex w is connected to s, s', l'_1 and exactly one leaf from L in D, let that leaf be l_1. Then branch on $\langle\, l_1 \mid s \mid s'\,\rangle$.

▶ **Branching rule (R11.3).** Assume that both s, s' in D have degree at least 3 in $G[V_2]$. Then branch on $\langle\, L \mid s \mid s' \mid L'\,\rangle$.

▶ **Context rule (R12).** Let D be a di-star in $G[V_2]$ and let w be the only red vertex connected to D such that w is connected to D by exactly two edges. We know that w is connected to a subset of $\{l_1, s, s', l'_1\}$, but not to both s and s'. This rule is split into three subrules (R12.1), (R12.2), and (R12.3) based on how w is connected to D.

▶ **Branching rule (R12.1).** Vertex w is connected to a center and its leaf in D, let them be s and l_1. Then branch on $\langle\, l_1 \mid s\,\rangle$.

▶ **Branching rule (R12.2).** Vertex w is connected to a center and to a leaf of the other center in D, let them be s' and l_1. Then branch on $\langle\, l_1 \mid s \mid s'\,\rangle$.

▶ **Context rule (R12.3).** Vertex w is connected to two opposite leaves in D, let them be l_1 and l'_1. This rule is split into four subrules (R12.3a), (R12.3b), (R12.3c), and (R12.3d) based on the degrees of s and s' and whether w is connected to other components.

▶ **Branching rule (R12.3a).** Both s, s' in D have degree 2 in $G[V_2]$ and w is connected to a component of $G[V_2]$ other than D, let x be the vertex w connects to outside D and let y be a neighbor of x in $G[V_2]$. Then branch on $\langle\, x \mid y \mid \{l_1, l'_1\}\,\rangle$.

▶ **Reduction rule (R12.3b).** Both s, s' in D have degree 2 in $G[V_2]$ and w is not connected to a component of $G[V_2]$ other than D. Then delete any vertex v in D and add it to the solution F.

▶ **Branching rule (R12.3c).** Exactly one of s, s' in D has degree at least 3 in $G[V_2]$, let it be s. Then branch on $\langle\, l_1 \mid s \mid l'_1\,\rangle$.

▶ **Branching rule (R12.3d).** Both s, s' in D have degree at least 3 in $G[V_2]$. Then branch on $\langle\, s \mid s' \mid \{l_1, l'_1\}\,\rangle$.

▶ **Context rule (R13).** Let D be a di-star in $G[V_2]$ and let w be the only red vertex connected to D such that w is connected to D by exactly three edges. We know that w is connected to a subset of $\{l_1, s, s', l'_1\}$, but not to both s and s'. Assume that w is connected to l_1, s, l'_1. This rule is split into four subrules (R13.1), (R13.2), (R13.3) and (R13.4) based on the degrees of s and s' and whether w is connected to other components.

▶ **Branching rule (R13.1).** Both s, s' in D have degree 2 in $G[V_2]$ and w is connected to at least one other component of $G[V_2]$, label the vertex w connects to outside D as x and the neighbor of x in $G[V_2]$ as y. Then branch on $\langle\, x \mid y \mid \{l_1, s'\} \mid \{s, l'_1\} \,\rangle$.

Proof of correctness. If none of the vertices x, y is deleted, then we have to delete at least two and, by Lemma 16, at most three vertices in D. Suppose we wanted to delete exactly two vertices. Out of six possible pairs, only $\{l_1, s'\}, \{s, s'\}, \{s, l'_1\}$ lead to a solution. We do not have to try $\{s, s'\}$, since if we delete s, then Lemma 10 becomes applicable and we may delete l'_1 instead of s'. Finally, if we wanted to delete three vertices, then by Lemma 16 those vertices would be $\{l_1, s, l'_1\}$, but this is already covered by branching on $\{s, l'_1\}$. Thus branching on $\langle\, x \mid y \mid \{l_1, s'\} \mid \{s, l'_1\} \,\rangle$ is correct. ◀

▶ **Reduction rule (R13.2).** Both s, s' in D have degree 2 in $G[V_2]$ and w is not connected to other component of $G[V_2]$. Then delete any vertex v in D and add it to the solution F.

▶ **Branching rule (R13.3).** Vertex s in D has degree at least 3 in $G[V_2]$. Then branch on $\langle\, l_1 \mid s \mid l'_1 \,\rangle$.

▶ **Branching rule (R13.4).** Vertex s' in D has degree at least 3 in $G[V_2]$. Then branch on $\langle\, l_1 \mid s' \mid l'_1 \,\rangle$.

▶ **Context rule (R14).** There is exactly one red vertex w connected to D by one edge. This rule is split into two subrules (R14.1) and (R14.2) based on how w is connected to D.

▶ **Reduction rule (R14.1).** Vertex w is connected to a leaf in D, let it be l_1. Then delete l_1 and add it to the solution F.

▶ **Branching rule (R14.2).** Vertex w is connected to a center in D, let it be s, and w is connected to at least one component of $G[V_2]$ other than D, label the vertex w connects to outside D as x. Then branch on $\langle\, s \mid x \,\rangle$.

▶ **Reduction rule (R15).** There are at least two red vertices connected to D by exactly one edge and they are connected to a single vertex. From Lemma 8 we know, that the red vertices are not connected to a component of $G[V_2]$ other than D and hence the single vertex must be a leaf, let it be l_1, otherwise no P_5 would be formed and Rule (R1) would be applicable. Then delete l_1 and add it to the solution F.

▶ **Branching rule (R16).** There are at least two red vertices connected to D by exactly one edge and they are connected to two opposite leaves, let those leaves be l_1, l'_1. Assume that there is at least one red vertex connected to each one of them. Further assume that the red vertices connected to l_1 are not connected to a component of $G[V_2]$ other than D. Then branch on $\langle\, s' \mid l'_1 \,\rangle$.

▶ **Branching rule (R17).** Let there be a di-star D and the two red vertices w, w' connected to D are connected to leaves l_1, l'_1, respectively, and at least one of the centers has degree at least three, let it be s. Then branch on $\langle\, s \mid s' \mid l'_1 \,\rangle$.

▶ **Branching rule (R18).** Let there be a di-star D and the two red vertices w, w' connected to D are connected to leaves l_1, l_1', respectively, and both centers have degree exactly two. Then branch on $\langle\, l_1 \mid l_1' \,\rangle$.

▶ **Lemma 21.** *Assume that Rules (R0) – (R9) are not applicable. Then at least one of Rules (R10) – (R18) is applicable.*

Proof. From Lemma 4 together with Lemmata 7, 13, 14, 15, 17, 18 and 20 we are now in the situation in which all components of $G[V_2]$ are di-stars and there must be a di-star D in $G[V_2]$ such that there is a P_5 that uses the vertices of D which implies there is at least one red vertex connected to D. For contradiction assume that Rules (R10) – (R18) are not applicable, i.e. no rules are applicable.

Let w be some red vertex connected to D. If $|N(w) \cap L| \geq 2$ or $|N(w) \cap L'| \geq 2$, then Rule (R10) is applicable. If there there is a red vertex w connected to l_i and a red vertex w' connected to l_j, $j \neq i$, then Rule (R2) applies. Similarly with vertices connected to l_i' and l_j'. So for the rest of this proof assume that each red vertex can be connected only to vertices l_1, s, s', or l_1'.

Firstly, assume that there is only one red vertex w connected to D. In Table 1 we list all possibilities (omitting several isomorphic cases) based on how w is connected to D, on the degrees of s and s', and whether w is connected only to D ($N(w) \subseteq V(D)$) or w is also connected outside D ($N(w) \nsubseteq V(D)$).

Observe that if there were at least two red vertices connected to D and w was connected to D by at least two edges, then Rule (R2) would be applicable with the only exception in case where w is connected to $\{l_1, s\}$ or $\{s', l_1'\}$ and the other red vertices to s or s', respectively. But this exception is resolved by Rule (R1) since vertices connected only to s or s' in this configuration are not used by any P_5. With this in mind, if there are at least two red vertices connected to D, then they are connected to D by only one edge.

Secondly, assume that there are at least two red vertices connected to D by exactly one edge. Let $X \subseteq V(D)$ be the vertices to which the red vertices are connected in D. Again, if $|X \cap L| \geq 2$ or $|X \cap L'| \geq 2$, then Rule (R2) applies. So suppose that $X \subseteq \{l_1, s, s', l_1'\}$. If $\{l_1, s'\} \subseteq X$ or $\{s, l_1'\} \subseteq X$ (which covers also cases where $|X| \geq 3$), then again Rule (R2) is applicable. If the vertices are connected to a single edge, then at least one of the vertices of such edge is a center, the vertices connected to it are not used by any P_5, and Rule (R1) applies. We conclude that the red vertices may be connected only to a single vertex or to two opposite leaves in D.

Thirdly, assume that the red vertices are connected to a single vertex. If that vertex is a leaf, then Rule (R15) is applicable, otherwise Rule (R1) is applicable.

Fourthly, assume that the red vertices are connected to two opposite leaves, let them be l_1 and l_1', and let W be the set of red vertices connected to l_1 and W' be the set of red vertices connected to l_1'. If the vertices in W or in W' (or both) are not connected to any component other than D, then Rule (R16) is applicable. This is the case whenever $|W| \geq 2$ or $|W'| \geq 2$ by Lemma 8.

Observe that now we are in situation in which there are exactly two red vertices w and w' connected to D by exactly one edge and these vertices are connected to l_1 and l_1', assume that w is connected to l_1 and w' is connected to l_1'. Furthermore, vertices w and w' are connected to at least one other di-star in $G[V_2]$. If at least one of L, L' has size at least two, then Rule (R17) is applicable, otherwise all di-stars in $G[V_2]$ are actually a P_4 paths and Rule (R18) is applicable.

Finally, there is no di-star remaining in $G[V_2]$ which together with Lemmata 4, 7, 13, 14, 15, 17, 18 and 20 implies that $G[V_2] = \emptyset$ and since V_1, V_2 is a P_5-free bipartition, there is no P_5 path remaining in G and Rule (R0) is applicable. ◀

Table 1 Possible configurations of single red vertex w and D in Lemma 21.

	$N(w) \not\subseteq V(D)$			$N(w) \subseteq V(D)$		
$N(w) \cap V(D)$	$\lvert L\rvert = 1$, $\lvert L'\rvert = 1$	$\lvert L\rvert > 1$, $\lvert L'\rvert = 1$	$\lvert L\rvert > 1$, $\lvert L'\rvert > 1$	$\lvert L\rvert = 1$, $\lvert L'\rvert = 1$	$\lvert L\rvert > 1$, $\lvert L'\rvert = 1$	$\lvert L\rvert > 1$, $\lvert L'\rvert > 1$
$\{l_1\}$	(R14.1)	(R14.1)	(R14.1)	(R14.1)	(R14.1)	(R14.1)
$\{s\}$	(R14.2)	(R14.2)	(R14.2)	(R1)	(R1)	(R1)
$\{s'\}$	(R14.2)	(R14.2)	(R14.2)	(R1)	(R1)	(R1)
$\{l'_1\}$	(R14.1)	(R14.1)	(R14.1)	(R14.1)	(R14.1)	(R14.1)
$\{l_1, s\}$	(R12.1)	(R12.1)	(R12.1)	(R12.1)	(R12.1)	(R12.1)
$\{l_1, s'\}$	(R12.2)	(R12.2)	(R12.2)	(R12.2)	(R12.2)	(R12.2)
$\{l_1, l'_1\}$	(R12.3a)	(R12.3c)	(R12.3d)	(R12.3b)	(R12.3c)	(R12.3d)
$\{s, s'\}$	(R11.1a)	(R11.2a)	(R11.3)	(R11.1a)	(R11.2a)	(R11.3)
$\{s, l'_1\}$	(R12.2)	(R12.2)	(R12.2)	(R12.2)	(R12.2)	(R12.2)
$\{s', l'_1\}$	(R12.1)	(R12.1)	(R12.1)	(R12.1)	(R12.1)	(R12.1)
$\{l_1, s, s'\}$	(R11.1b)	(R11.2b)	(R11.3)	(R11.1b)	(R11.2b)	(R11.3)
$\{l_1, s, l'_1\}$	(R13.1)	(R13.3)	(R13.3)	(R13.2)	(R13.3)	(R13.3)
$\{l_1, s', l'_1\}$	(R13.1)	(R13.4)	(R13.3)	(R13.2)	(R13.4)	(R13.3)
$\{s, s', l'_1\}$	(R11.1b)	(R11.2c)	(R11.3)	(R11.1b)	(R11.2c)	(R11.3)
$\{l_1, s, s', l'_1\}$	(R11.1c)	(R11.2d)	(R11.3)	(R11.1d)	(R11.2d)	(R11.3)

3.11 Final remarks

From Lemma 21 we know that there is always at least one rule applicable. It remains to analyze the running time of the disjoint compression routine DISJOINT_R.

▶ **Theorem 22.** *The* DISJOINT_R *procedure solves the* 5-PVCWB *problem in* $\mathcal{O}^*(3^k)$ *time.*

By standard arguments (see Cygan et al. [3, pages 80–81]) we get the following corollary.

▶ **Corollary 23.** *The iterative compression algorithm solves the* 5-PVC *problem and runs in* $\mathcal{O}^*(4^k)$ *time.*

4 Conclusion

We conclude this paper with a few open questions.

Firstly, we see the trend of solving 3-PVC, 4-PVC and now 5-PVC with the iterative compression technique, so it is natural to ask whether this approach can be further used for 6-PVC or even to d-PVC in general. However, given the complexity (number of rules) of the algorithm presented in this paper, it seems more reasonable to first try to find a simpler algorithm for 5-PVC.

Secondly, motivated by the work of Orenstein et al. [13], we ask whether known algorithms for 3-PVC, 4-PVC, 5-PVC can be generalized to work with directed graphs.

Finally, due to Fafianie and Kratsch [5] we know that d-PVC problem has a kernel with $\mathcal{O}(k^d)$ vertices and edges. Dell and van Melkebeek [4] showed that there is no $\mathcal{O}(k^{d-\epsilon})$ kernel for any $\epsilon > 0$ for general d-HITTING SET unless coNP is in NP/poly, which would imply a collapse of the polynomial-time hierarchy. However, for 3-PVC problem, Xiao and Kou [19] presented a kernel with $5k$ vertices. To our knowledge, it is not known whether there exists a linear kernel for 4-PVC or any d-PVC with $d \geq 5$.

References

1 Boštjan Brešar, František Kardoš, Ján Katrenič, and Gabriel Semanišin. Minimum k-path vertex cover. *Discrete Applied Mathematics*, 159(12):1189–1195, 2011. doi:10.1016/j.dam.2011.04.008.

2 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.

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 Holger Dell and Dieter van Melkebeek. Satisfiability Allows No Nontrivial Sparsification unless the Polynomial-Time Hierarchy Collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.

5 Stefan Fafianie and Stefan Kratsch. A Shortcut to (Sun)Flowers: Kernels in Logarithmic Space or Linear Time. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, pages 299–310, 2015. doi:10.1007/978-3-662-48054-0_25.

6 Fedor V. Fomin, Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Saket Saurabh. Iterative compression and exact algorithms. *Theor. Comput. Sci.*, 411(7-9):1045–1053, 2010. doi:10.1016/j.tcs.2009.11.012.

7 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact Algorithms via Monotone Local Search. *J. ACM*, 66(2):8:1–8:23, 2019. doi:10.1145/3284176.

8 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010. doi:10.1007/978-3-642-16533-7.

9 Stefan Funke, André Nusser, and Sabine Storandt. On k-Path Covers and their applications. *VLDB J.*, 25(1):103–123, 2016. doi:10.1007/s00778-015-0392-3.

10 Ján Katrenič. A faster FPT algorithm for 3-path vertex cover. *Inf. Process. Lett.*, 116(4):273–278, 2016. doi:10.1016/j.ipl.2015.12.002.

11 John M. Lewis and Mihalis Yannakakis. The Node-Deletion Problem for Hereditary Properties is NP-Complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

12 Marián Novotný. Design and Analysis of a Generalized Canvas Protocol. In *Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices, 4th IFIP WG 11.2 International Workshop, WISTP 2010, Passau, Germany, April 12-14, 2010. Proceedings*, pages 106–121, 2010. doi:10.1007/978-3-642-12368-9_8.

13 Yaron Orenstein, David Pellow, Guillaume Marçais, Ron Shamir, and Carl Kingsford. Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing. *PLoS Computational Biology*, 13(10), 2017. doi:10.1371/journal.pcbi.1005777.

14 Dekel Tsur. An $O^*(2.619^k)$ algorithm for 4-path vertex cover. *CoRR*, abs/1811.03592, 2018. arXiv:1811.03592.

15 Dekel Tsur. Parameterized algorithm for 3-path vertex cover. *Theoretical Computer Science*, 783:1–8, 2019. doi:10.1016/j.tcs.2019.03.013.

16 Jianhua Tu. A fixed-parameter algorithm for the vertex cover P_3 problem. *Inf. Process. Lett.*, 115(2):96–99, 2015. doi:10.1016/j.ipl.2014.06.018.

17 Jianhua Tu and Zemin Jin. An FPT algorithm for the vertex cover P_4 problem. *Discrete Applied Mathematics*, 200:186–190, 2016. doi:10.1016/j.dam.2015.06.032.

18 Mingyu Xiao and Shaowei Kou. Exact algorithms for the maximum dissociation set and minimum 3-path vertex cover problems. *Theor. Comput. Sci.*, 657:86–97, 2017. doi:10.1016/j.tcs.2016.04.043.

19 Mingyu Xiao and Shaowei Kou. Kernelization and Parameterized Algorithms for 3-Path Vertex Cover. In *Theory and Applications of Models of Computation - 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, pages 654–668, 2017 . doi:10.1007/978-3-319-55911-7_47.

20 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017. doi:10.1016/j.ic.2017.06.001.

Parameterized Complexity of Fair Vertex Evaluation Problems

Dušan Knop
Algorithmics and Computational Complexity, Faculty IV, TU Berlin
Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Prague, Czech Republic
dusan.knop@tu-berlin.de

Tomáš Masařík
Department of Applied Mathematics, Charles University, Prague, Czech Republic
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
masarik@kam.mff.cuni.cz

Tomáš Toufar
Computer Science Institute, Charles University, Prague, Czech Republic
toufi@iuuk.mff.cuni.cz

Abstract

A prototypical graph problem is centered around a graph-theoretic property for a set of vertices and a solution to it is a set of vertices for which the desired property holds. The task is to decide whether, in the given graph, there exists a solution of a certain quality, where we use size as a quality measure. In this work, we are changing the measure to the fair measure (cf. Lin and Sahni [27]). The fair measure of a set of vertices S is (at most) k if the number of neighbors in the set S of any vertex (in the input graph) does not exceed k. One possible way to study graph problems is by defining the property in a certain logic. For a given objective, an evaluation problem is to find a set (of vertices) that simultaneously minimizes the assumed measure and satisfies an appropriate formula. More formally, we study the MSO Fair Vertex Evaluation, where the graph-theoretic property is described by an MSO formula.

In the presented paper we show that there is an FPT algorithm for the MSO Fair Vertex Evaluation problem for formulas with one free variable parameterized by the twin cover number of the input graph and the size of the formula. One may define an extended variant of MSO Fair Vertex Evaluation for formulas with ℓ free variables; here we measure a maximum number of neighbors in each of the ℓ sets. However, such variant is W[1]-hard for parameter ℓ even on graphs with twin cover one.

Furthermore, we study the Fair Vertex Cover (Fair VC) problem. Fair VC is among the simplest problems with respect to the demanded property (i.e., the rest forms an edgeless graph). On the negative side, Fair VC is W[1]-hard when parameterized by both treedepth and feedback vertex set of the input graph. On the positive side, we provide an FPT algorithm for the parameter modular width.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Logic; Theory of computation → Graph algorithms analysis

Keywords and phrases Fair objective, metatheorem, fair vertex cover, twin cover, modular width

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.33

Related Version `http://arxiv.org/abs/1803.06878`

Funding *Dušan Knop*: Supported by DFG under project "MaMu", NI 369/19 and partly supported by the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16 019/0000765 "Research Center for Informatics".
Tomáš Masařík : Author was supported by SVV–2017–260452 and by GAUK 1514217 of Charles University and by the CE-ITI grant project P202/12/G061 of GA ČR.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 33; pp. 33:1–33:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A prototypical graph problem is centered around a fixed property for a set of vertices. A solution is any set of vertices for which the given property holds. In a similar manner, one can define the solution as a set of vertices such that the given property holds when we remove this set of vertices from the input graph. This leads to the introduction of deletion problems – a standard reformulation of some classical problems in combinatorial optimization introduced by Yannakakis [37]. Formally, for a graph property π we formulate a *vertex deletion problem*. That is, given a graph $G = (V, E)$, find a smallest possible set of vertices W such that $G \setminus W$ satisfies the property π. Many classical problems can be formulated in this way such as Minimum Vertex Cover (π describes an edgeless graph) or Minimum Feedback Vertex Set (π is valid for forests).

Clearly, the complexity of a graph problem is governed by the associated property π. We can either study one particular problem or a broader class of problems with related graph-theoretic properties. One such relation comes from logic, for example, two properties are related if it is possible to express both by a first order (FO) formula. Then, it is possible to design a *model checking algorithm* that for any property π expressible in the fixed logic decides whether the input graph with the vertices from W removed satisfies π or not.

Undoubtedly, Courcelle's Theorem [4] for graph properties expressible in the monadic second-order logic (MSO) on graphs of bounded treewidth plays a prime role among model checking algorithms. In particular, Courcelle's Theorem provides for an MSO sentence φ an algorithm that given an n-vertex graph G with treewidth k decides whether φ holds in G in time $f(k, |\varphi|)n$, where f is some computable function and $|\varphi|$ is the quantifier depth of φ. In terms of parameterized complexity, such an algorithm is called *fixed-parameter tractable* (the problem belongs to the class FPT for the combined parameter $k + |\varphi|$). We refer the reader to monographs [7, 8] for background on parameterized complexity theory and algorithm design. There are many more FPT model-checking algorithms, e.g., an algorithm for (existential counting) modal logic model checking on graphs of bounded treewidth [33], MSO model checking on graphs of bounded neighborhood diversity [24], or MSO model checking on graphs of bounded shrubdepth [14] (generalizing the previous result). First order logic (FO) model checking received recently quite some attention as well and algorithms for graphs with bounded degree [34], nowhere dense graphs [16], and some dense graph classes [11] were given.

Fair Objective. Fair deletion problems, introduced by Lin and Sahni [27], are such modifications of deletion problems where instead of minimizing the size of the deleted set we change the objective. The Fair Vertex Deletion problem is defined as follows. For a given graph $G = (V, E)$ and a property π, the task is to find a set $W \subseteq V$ which minimizes the maximum number of neighbors in the set W over all vertices, such that the property π holds in $G \setminus W$. Intuitively, the solution should not be concentrated in a neighborhood of any vertex. In order to classify (fair) vertex deletion problems we study the associated decision version, that is, we are interested in finding a set W of size at most k, for a given number k. Note that this can introduce only polynomial slowdown, as the value of our objective is bounded by 0 from below and by the number of vertices of the input graph from above (provided a solution exists). Since we are about to use a formula with a free variable X to express the desired property π, we naturally use X to represent the set of deleted vertices in the formula. The *fair cost of a set* $W \subseteq V$ is defined as $\max_{v \in V} |N(v) \cap W|$. We refer to the function that assigns each set $W \subseteq V$ its fair cost as to the *fair objective function*. Here, we give a formal definition of the computational problem when the property π is defined in some logic L.

Fair L Vertex Deletion
Input: An undirected graph $G = (V, E)$, an L sentence φ, and a positive integer k.
Question: Is there a set $W \subseteq V$ of fair cost at most k such that $G \setminus W \models \varphi$?

Let $\varphi(X)$ be a formula with one free variable and let $G = (V, E)$ be a graph. For a set $W \subseteq V$ by $\varphi(W)$ we mean that we substitute W for X in φ. The definition below can be naturally generalized to contain ℓ free variables. We would like to point out one crucial difference between deletion and evaluation problems, namely that in evaluation problems we have access to the variable that represents the solution. This enables evaluation problems to impose some conditions on the solution, e.g., we can ensure that the graph induced by the solution has diameter at most 2 or is triangle-free.

Fair L Vertex Evaluation[1]
Input: An undirected graph $G = (V, E)$, an L formula $\varphi(X)$ with one free variable, and a positive integer k.
Question: Is there a set $W \subseteq V$ of fair cost at most k such that $G \models \varphi(W)$?

Minimizing the fair cost arises naturally for edge problems in many situations as well, e.g., in defective coloring [6], which has been substantialy studied from the practical network communication point of view [17, 19]. A graph $G = (V, E)$ is (k, d)-colorable if every vertex can be assigned a color from the set $[k]$ in such a way that every vertex has at most d neighbors of the same color. This problem can be reformulated in terms of fair deletion; we aim to find a set of edges F such that graph $G' = (V, F)$ has maximum degree d and $G \setminus F$ can be partitioned into k independent sets.

Related Results. There are several possible research directions once a model checking algorithm is known. One possibility is to broaden the result either by enlarging the class of graphs it works for or by extending the expressive power of the concerned logic, e.g., by introducing a new predicate [23]. Another obvious possibility is to find the exact complexity of the general model checking problem by providing better algorithms (e.g., for subclasses [24]) and/or lower-bounds for the problem [9, 25]. Finally, one may instead of deciding a sentence turn attention to finding a set satisfying a formula with a free variable that is optimal with respect to some objective function [1, 5, 15]. In this work, we take the last presented approach – extending a previous work on MSO model checking for the fair objective function.

When extending a model checking result to incorporate an objective function or a predicate, we face two substantial difficulties. On the one hand, we are trying to introduce as strong extension as possible, while on the other, we try not to worsen the running time too much. Of course, these two are in a clash. One evident possibility is to sacrifice the running time and obtain an XP algorithm, that is, an algorithm running in time $f(k)|G|^{g(k)}$. For example there is an XP algorithm for the Fair MSO_2 Vertex Evaluation problems parameterized by treewidth (and the size of the formula) by Kolman et al. [21] running in time $f(|\varphi|, \mathrm{tw}(G))|G|^{g(\mathrm{tw}(G))}$. An orthogonal extension is due to Szeider [35] for the so-called local cardinality constraints (MSO-LCC) who gave an XP algorithm parameterized by treewidth. If we decide to keep the FPT running time, such a result is not possible for treedepth (consequently for treewidth), as we give W[1]-hardness result for a very basic

[1] This problem is called Generalized Fair L Vertex-Deletion in [29] and in the respective conference version [28]. However, we believe that evaluation is a more suitable expression and coincides with standard terminology in logic.

Fair $\mathsf{L}_\emptyset$ Vertex Deletion problem[2] in this paper. A weaker form of this hardness was already known for FO logic [29]. Ganian and Obdržálek [15] study CardMSO and provide an FPT algorithm parameterized by neighborhood diversity. Recently, Masařík and Toufar [29] examined fair objective and provide an FPT algorithm for the Fair MSO_1 Vertex Evaluation problem parameterized by neighborhood diversity. See also [20] for a discussion of various extensions of the MSO.

We want to turn a particular attention to the Fair Vertex Cover (Fair VC) problem which, besides its natural connection with Vertex Cover, has some further interesting properties. For example, we can think about classical vertex cover as having several crossroads (vertices) and roads (edges) that we need to monitor. However, people could get uneasy if they will see too many policemen from one single crossroad. In contrast, if the vertex cover has low fair cost, it covers all roads while keeping a low number of policemen in the neighborhood of every single crossroad.

1.1 Our Results

We give a metatheorem for graphs having bounded twin cover. Twin cover (introduced by Ganian [12]; see also [13]) is one possible generalization of the vertex cover number. Here, we measure the number of vertices needed to cover all edges that are not twin-edges; an edge $\{u, v\}$ is a *twin-edge* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. Ganian introduced twin cover in the hope that it should be possible to extend algorithms designed for parameterization by the vertex cover number to a broader class of graphs.

▶ **Theorem 1.** *The* Fair MSO_1 Vertex Evaluation *problem parameterized by the twin cover number and the quantifier depth of the formula admits an* FPT *algorithm.*

We want to point out here that in order to obtain this result we have to reprove the original result of Ganian [12] for MSO_1 model checking on graphs of bounded twin cover. For this, we extend arguments given by Lampis [24] in the design of an FPT algorithm for MSO_1 model checking on graphs of bounded neighborhood diversity. We do this to obtain better insight into the structure of the graph (a kernel) on which model checking is performed (its size is bounded by a function of the parameter). This, in turn, allows us to find a solution minimizing the fair cost and satisfying the MSO_1 formula. The result by Ganian in version [12] is based on the fact that graphs of bounded twin cover have bounded shrubdepth and so MSO_1 model checking algorithm on shrubdepth ([14, 10]) can be used.

When proving hardness results it is convenient to show the hardness result for a concrete problem that is expressible by an MSO_1 formula, yet as simple as possible. Therefore, we introduce a key problem for Fair Vertex Deletion – the Fair VC problem.

Fair Vertex Cover (Fair VC)	
Input:	An undirected graph $G = (V, E)$, and a positive integer k.
Question:	Is there a set $W \subseteq V$ of fair cost at most k such that $G \setminus W$ is an edgeless graph?

The Fair VC problem can be expressed in any logic that can express an edgeless graph (we denote such logic $\mathsf{L}_\emptyset$) which is way weaker than FO. Therefore, we propose a systematic study of the Fair VC problem which, up to our knowledge, have not been considered before.

▶ **Theorem 2.** *The* Fair VC *problem parameterized by treedepth* $\operatorname{td}(G)$ *and feedback vertex set* $\operatorname{fvs}(G)$ *combined is* W[1]*-hard.*

[2] Here, $\mathsf{L}_\emptyset$ stands for any logic that can express an edgeless graph.

Table 1 The table summarize some related (with a citation) and all the presented (with a reference) results on the studied parameters. Green cells denote FPT results, and red cells represent hardness results. Logic L in metatheorems is specified by a logic used in the respective theorem. Symbol $*$ denotes implied results from previous research and symbol ✓denotes new implied results. A question mark (?) indicates that the complexity is unknown.

	vc	fvs + td	tc	nd	cvd	mw
Fair VC	$*$	T2	✓	$*$	?	T3
FV L Del	MSO$_2$ $*$	L$_\emptyset$ ✓	MSO$_1$ ✓	MSO$_1$ $*$	?	?
FV L Eval	MSO$_2$ [29]	L$_\emptyset$ ✓	MSO$_1$ T1	MSO$_1$ $*$	?	?
ℓ-FV L Eval	MSO$_1$ $*$	L$_\emptyset$ ✓	MSO$_1$ T5	MSO$_1$ [20]	MSO$_1$ ✓	MSO$_1$ ✓

Note that this immediately yields W[1]-hardness and $f(w)n^{o(\sqrt{w})}$ lower bound for Fair L$_\emptyset$ Vertex Evaluation. Previously, an $f(w)n^{o(w^{1/3})}$ lower bound was given for FO logic by Masařík and Toufar [29]. Thus our result is stronger in both directions, i.e., the lower bound is stronger, and the logic is less powerful. On the other hand, we show that Fair VC can be solved efficiently in dense graph models.

▶ **Theorem 3.** *The* Fair VC *problem parameterized by modular width* $\mathrm{mw}(G)$ *admits an* FPT *algorithm with running time* $2^{\mathrm{mw}(G)} \cdot \mathrm{mw}(G) \cdot n^3$*, where n is the number of vertices in G.*

We point out that the Fair VC problem is (rather trivially) AND-compositional and thus it does not admit a polynomial kernel for parameterization by modular width.

▶ **Lemma 4.** *The* Fair VC *problem parameterized by the modular width of the input graph does not admit a polynomial kernel, unless* NP $\subseteq$ coNP/ poly.

Moreover, an analog to Theorem 3 cannot hold for parameterization by shrubdepth of the input graph. This is a consequence of Theorem 2 and the fact that if a class of graphs has bounded treedepth, then it has bounded shrubdepth (cf. [14, Proposition 3.4]).

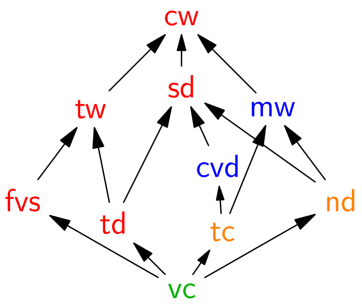
Another limitation in a rush for extensions of Theorem 1 is given when aiming for more free variables. More formally, the problem ℓ-Fair L Vertex Evaluation has formula $\varphi(X_1, \ldots, X_\ell)$ with ℓ free variables as an input and $W_1, \ldots, W_\ell$ are the corresponding sets in G of fair cost at most k. The *fair cost* of $W_1, \ldots, W_\ell$ is defined as $\max_{v \in V} \max_{i \in [\ell]} |N(v) \cap W_i|$. It is very surprising that such a generalization is not possible for parameterization by twin cover, since the same extension is possible for parameterization by neighborhood diversity [20]. In fact, they prove something even stronger, i.e., an FPT algorithm parameterized by neighborhood diversity in the context of $\mathsf{MSO}^{\mathsf{L}}_{\mathsf{lin}}$ is given in [20]. In $\mathsf{MSO}^{\mathsf{L}}_{\mathsf{lin}}$ one can specify both lower- and upper-bound for each vertex and each free variable (i.e., a feasibility interval is given for every vertex).

▶ **Theorem 5** (★[3]). *The* ℓ-Fair FO Vertex Evaluation *problem is* W[1]*-hard for parameter ℓ even on graphs with twin cover of size one.*

The reduction is done from the Unary ℓ-Bin Packing problem; the lower-bound and W[1]-hardness of Unary ℓ-Bin Packing was given by Jansen et al. [18].

For an overview of the results, please refer to Table 1 and to Figure 1 for the hierarchy of classes.

[3] We mark a result by ★ if the proof is omitted and deferred to the full version (available on arXiv).



🟨 **Figure 1** Hierarchy of graph parameters with depicted complexity of the Fair L Vertex Evaluation problem. An arrow indicates that a graph parameter upper-bounds the other. Thus, hardness results are implied in the direction of arrows, and FPT algorithms are implied in the reverse direction. Green colors indicate FPT results for MSO_2, orange are FPT for MSO_1, blue are open, and red are hardness results. We denote treewidth by tw, shrubdepth by sd, and clique width by cw. We refer to book [7] for definitions. Other parameters and their respective abbreviations are defined in Subsection 1.2.

1.2 Preliminaries

For a positive integer n we denote $[n]$ the set $\{1, \ldots, n\}$. We deal with simple undirected graphs, for further standard notation we refer to monographs: graph theory [30] and parameterized complexity [7]. For a vertex v by $N(v)$ we denote the neighborhood of v and by $N[v]$ we denote the closed neighborhood of vertex v, i.e., $N(v) \cup \{v\}$.

A parameter closely related to twin cover is *cluster vertex deletion* ($\mathrm{cvd}(G)$), that is, the smallest number of vertices one has to delete from a graph in order to get a collection of (disjoint) cliques. *Treedepth* of a graph G ($\mathrm{td}(G)$) is the minimum height of a rooted forest whose transitive closure contains the graph G [31]. *Feedback vertex set* ($\mathrm{fvs}(G)$) is the minimum number of vertices of a graph G whose removal leaves the graph without cycles. *Neighborhood diversity* ($\mathrm{nd}(G)$) is the smallest integer r such that the the vertex set of G can be partitioned into r sets $V_1, \ldots, V_r$ in such a way that the graph $G[V_i]$ is either a clique or an edgeless graph for all $i \in [r]$ and the bipartite graph $G[V_i, V_j]$ is either a complete bipartite graph or an edgeless graph for all distinct $i, j \in [r]$. *Modular width* of a graph G ($\mathrm{mw}(G)$), is the smallest positive integer r such that G can be obtained from an algebraic expression of width at most r, defined as follows. The *width of an expression* A is the maximum number of operands used by any occurrence of the substitution operation in A, where A is an algebraic expression that uses the following operations:

1. Create an isolated vertex.
2. The *substitution operation* with respect to a template graph T with vertex set $[r]$ and graphs $G_1, \ldots, G_r$ created by algebraic expression. The substitution operation, denoted by $T(G_1, \ldots, G_r)$, results in the graph on vertex set $V = V_1 \cup \cdots \cup V_r$ and edge set $E = E_1 \cup \cdots \cup E_r \cup \bigcup_{\{i,j\} \in E(T)} \big\{\{u, v\} : u \in V_i, v \in V_j\big\}$, where $G_i = (V_i, E_i)$ for all $i \in [r]$.

An algebraic expression of width $\mathrm{mw}(G)$ can be computed in linear time [36].

We stick with standard definitions and notation in logic. For a comprehensive summary, please consult a book by Libkin [26].

2 Metatheorems for Fair Evaluation

We show an FPT algorithm as it is stated in Theorem 1. We give a more detailed statement that implies the promised result straigthforwardly.

We split the proof into two parts. First, we show an algorithm for MSO_1 model checking parameterized by twin cover of the graph (Proposition 8). In the second part, we prove that we can even compute the optimal fair cost (Proposition 12) and so derive the promised result.

Overview of the Algorithm. For the model checking algorithm, we closely follow the approach of Lampis [24]. The idea is that if there is a large set of vertices with the same closed neighborhood, then some of them are irrelevant, i.e., we can delete them without affecting the truthfulness of the given formula φ. For graphs of bounded neighborhood diversity using this rule alone can reduce the number of vertices below a bound that depends on $\mathrm{nd}(G)$ and $|\varphi|$ only, thus providing an FPT model checking algorithm. For the graphs of bounded twin cover, this approach can be used to reduce the size of all (twin) cliques, yet their number can still be large. We take the approach one step further and describe the deletion of irrelevant cliques in a similar manner; these rules together yield a model checking algorithm for graphs of bounded twin cover.

The reduction rules also lead to a notion of *shape* of a set $W \subseteq V$. The motivation behind shapes is to partition all subsets of V such that if two sets W, W' have the same shape, then $G \models \varphi(W)$ if and only if $G \models \varphi(W')$. This allows us to consider only one set of each shape for the purposes of model checking. Since the number of all distinct shapes is bounded by some function of parameters, we can essentially brute force through all possible shapes.

A final ingredient is an algorithm that for a given shape outputs a subset of vertices with this shape that minimizes the fair cost. This algorithm uses ILP techniques, in particular minimizing quasiconvex function subject to linear constraints.

Notation. In what follows $G = (V, E)$ is a graph and K is its twin cover of size k. An MSO_1 formula φ contains q_S set quantifiers and q_v element (vertex) quantifiers. Given a twin cover K and $A \subseteq K$, we say that A is the *cover set* of a set $S \subseteq V \setminus K$ if every $v \in S$ has $N(v) \cap K = A$. Note that, by the definition of twin cover, for all $u, v \in V \setminus K$ with $\{u, v\} \in E$ we have that A is a cover set for u if and only if A is a cover set for v. We say that two cliques have the same *type* if they have the same size and the same cover set. Clearly, if the cover set is fixed, two cliques agree on type if and only if their sizes are the same. We define a *labeled graph*, that is, a graph and a collection of labels on the vertices. We say that two cliques have the *same labeled type* if all of them have the same size, the same cover set and the same labels on vertices.

2.1 Model checking

▶ **Proposition 6** ([24, Lemma 5 and Theorem 4]). *Let ϕ be an MSO_1 formula and let G be a labeled graph. If there is a set S of more than $2^{q_S} q_v$ vertices having the same closed neighborhood and the same labels, then for any $v \in S$ we have $G \models \varphi$ if and only if $G \setminus v \models \varphi$.*

In particular, if G is a graph with just one label, then for any clique C where each vertex has exactly the same closed neighborhood in G the following holds. Either there is a vertex $v \in C$ such that $G \models \phi$ if and only if $G \setminus v \models \phi$ or the size of C is bounded by $2^{q_S+1} q_v$.

Proposition 6 bounds the size of a maximum clique in $G \setminus K$ because we can apply it repeatedly for each clique that is bigger than the threshold $2^{q_S+1} q_v$. Now, we need to bound the number of cliques of each type. For this, we establish the following technical lemma.

▶ **Lemma 7** (★)**.** *Let G be a labeled graph with twin cover K. Let φ be an MSO_1 formula with q_v element quantifiers and q_S set quantifiers. Suppose the size of a maximum clique in $G \setminus K$ is bounded by r. If there are strictly more than*

$$\alpha(q_S, q_v) = 2^{rq_S}(q_v + 1)$$

cliques of the same labeled type $\mathcal{T}$, then there exists a clique C of the labeled type $\mathcal{T}$ such that $G \models \varphi$ if and only if $G \setminus C \models \varphi$.

From this, we can derive a model checking algorithm.

▶ **Proposition 8** (★ Model checking on graphs of bounded twin cover)**.** *Let G be a graph with twin cover K of size k and the size of the maximum clique in $G \setminus K$ bounded by $2^{q_S}q_v$ and φ is an MSO_1 sentence then either there exists a clique $C \in G \setminus K$ such that $G \models \phi$ if and only if $G \setminus C \models \varphi$ or the size of G is bounded by*

$$k + (q_v + 1)q_v^2 2^{k+2q_S+2^{q_S}q_Sq_v} = 2^{\mathcal{O}(k+2^{q_S}q_Sq_v)}\,.$$

2.2 Finding a Fair Solution

In the upcoming proof we follow the ideas of Masařík and Toufar [29]. They define, for a given formula $\varphi(X)$, a so-called shape of a set $W \subseteq V$ in G. The idea behind a shape is that in order to do the model checking we have deleted some vertices from G that cannot change the outcome of $\varphi(X)$, however, we have to derive a solution of minimal cost in the whole graph G. Thus the shape characterizes a set under which $\varphi(X)$ holds and we have to be able to find a set $W \subseteq V(G)$ for which $\varphi(W)$ holds and W minimizes the fair cost among sets having this shape.

Shape. Let $G = (V, E)$ be a graph, $\varphi(X)$ an MSO formula, $K \subseteq V$ a twin cover of G, $A \subseteq K$, and let $r = 2^{q_S+2}q_v$ and $\alpha = 2^{r(q_S+1)}(q_v + 1)$. Let $\mathcal{C}$ be the collection of all cliques in G such that A is their cover set. We define an A-*shape.* An A-shape of size r is a two dimensional table S_A of dimension $(r + 2) \times (r + 2)$ indexed by $\{0, 1, \dots, r + 1\} \times \{0, 1, \dots, r + 1\}$. Each entry $S_A(i, j) \in \{0, \dots, \alpha + 1\}$. The interpretation of $S_A(i, j)$ is the minimum of $\alpha + 1$ and of the number of cliques C with $N(C) = A$ such that

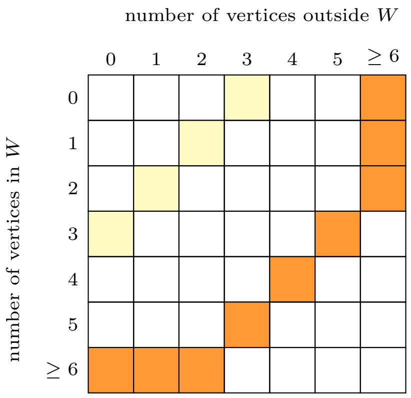
$$\min(\alpha + 1, |C \cap W|) = i \text{ and } \min(\alpha + 1, |C \setminus W|) = j.$$

Finally, the shape of X in G is a collection of A-shapes for all $A \subseteq K$.

A solution for $\mathcal{C}$ with cover set A can be formally described by a function sol: $\mathcal{C} \to \mathbb{N} \times \mathbb{N}$. The solution sol is *valid* if for every $C \in \mathcal{C}$ with $\mathrm{sol}(C) = (i, j)$ either $i + j = |C|$ or $|C| \geq r$, $i = r + 1$ (or equivalently $j = r + 1$), and $i + j < |C|$. For an illustration of a valid assignment please refer to Figure 2. We say that a valid solution sol is *compatible* with the shape S_A if $S(i, j) = \left|\mathrm{sol}^{-1}(i, j)\right|$, whenever $S(i, j) \leq \alpha$ and $\left|\mathrm{sol}^{-1}(i, j)\right| \geq \alpha + 1$ if $S(i, j) = \alpha + 1$. The A-shape S_A is said to be valid if there exists a valid solution for S_A. Note that such a solution does not exist if the shape specifies too many (or too few) cliques of certain sizes. The shape S is *valid* if all its A-shapes are valid.

The following lemma is a key observation about shapes.

▶ **Lemma 9.** *Let φ be an MSO_1 formula with one free variable, G a graph and W, W' two subsets of vertices having the same shape. Then $G \models \varphi(W)$ if and only if $G \models \varphi(W')$.*

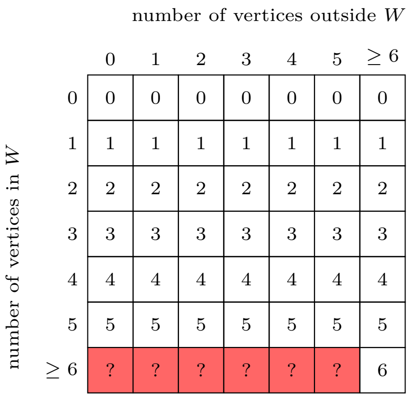


number of vertices outside W

number of vertices in W

	0	1	2	3	4	5	≥ 6
0				🟨			🟧
1			🟨				🟧
2		🟨					🟧
3	🟨					🟧	
4					🟧		
5				🟧			
≥ 6	🟧	🟧	🟧				

Figure 2 Example of a 7×7 A-shape. All cliques of size 3 will be assigned to yellow (light gray) fields, while cliques of size 8 will be assigned to orange (darker gray) fields.



number of vertices outside W

number of vertices in W

	0	1	2	3	4	5	≥ 6
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
≥ 6	?	?	?	?	?	?	6

Figure 3 An example of uncertainty in computation of objective function. The value in the last row depends on the size of the clique we are assigning to those cells. The value in the cell is how much we pay for any compatible clique assigned to this cell.

Proof. The proof follows using Proposition 6 and Lemma 7. Indeed, if we take the graph G with one label corresponding to set W and apply the reduction rules given by Proposition 6 and Lemma 7 and repeat the same process with W', we obtain two isomorphic labeled graphs. ◀

Lemma 9 allows us to say that a formula with one free variable holds under a shape since it is irrelevant which subset of vertices of this particular shape is assigned to the free variable. Also note that deciding whether the formula holds under the shape can be done in FPT time by simply picking arbitrary assignment of the given shape and running a model checking algorithm.

Lemma 11 computes a solution of minimal cost for an A-shape. We do this by reducing the task to integer linear programming (ILP) while using non-linear objective. A fuction $f : \mathbb{R}^p \to \mathbb{R}$ is *separable convex* if there exist convex functions $f_i : \mathbb{R} \to \mathbb{R}$ for $i \in [p]$ such that $f(x_1, \dots, x_p) = \sum_{i=1}^{p} f_i(x_i)$.

▶ **Theorem 10** ([32] – simplified). *Integer linear programming with objective minimization of a separable convex function in dimension p is* FPT *with respect to p and space exponential in L the length of encoding of the ILP instance.*

▶ **Lemma 11.** *Let $G = (V, E)$ be a graph, K be a twin cover of G, and $\emptyset \neq A \subseteq K$. There is an algorithm that given an A-shape S_A of size r computes a valid solution for S_A of minimal cost in time $f(|K|, r) \cdot |G|^{O(1)}$ or reports that S_A is not valid.*

Proof. Let $\mathcal{C}$ be the collection of all cliques such that A is their cover set. We split the task of finding a minimal solution to S_A into two independent parts depending on the size of cliques assigned in the phase.

The first phase is for cliques in $\mathcal{C}$ with sizes at most r. Observe that these can be assigned deterministically in a greedy way. This is because no cell of S_A is shared by two sizes and we can see that if there are more cells with value α on the corresponding diagonal we can always prefer the top one as this minimizes the cost (see Figure 3). However, this is not possible for larger cliques as they may in general share some cells of S_A and thus we defer them to the second phase.

Now observe that the most important vertices for computing the cost are the vertices constituting the set A. To see this just note that all other vertices see only A and their neighborhood (a clique) which is at least as large as for the vertices in A. It follows that we should only care about the number of selected vertices such that A is their cover set. Thus if the size of all cliques in $\mathcal{C}$ is bounded in terms of k we are done. Alas, this is not the case.

We split the set $\mathcal{C}$ into sets $\mathcal{C}_1, \dots, \mathcal{C}_{2r}$, and $\mathcal{C}_{\max}$. A clique $C \in \mathcal{C}$ belongs to $\mathcal{C}_{|C|}$ if $1 \le |C| \le 2r$ and belongs to $\mathcal{C}_{\max}$ otherwise. Note that cliques from $\mathcal{C}_{\max}$ may be assigned only to cells having at least one index $r+1$. As mentioned we are about to design an ILP with a non-linear objective function. This ILP has variables $x_{i,j}^q$ that express the number of cliques from the set $\mathcal{C}_q$ assigned to the cell (i,j) of S_A (that is, $1 \le i,j \le r+1$ and $q \in Q = \{1, \dots, 2r\} \cup \{\max\}$). The obvious conditions are the following (the $\trianglerighteq$ symbol translates to $=$ if $S(i,j) \le \alpha$ while it translates to $\ge$ if $S(i,j) = \alpha + 1$).

$$\sum_{q \in Q} x_{i,j}^q \trianglerighteq S_A(i,j) \qquad 0 \le i,j \le r+1$$

$$\sum_{0 \le i,j \le r+1} x_{i,j}^q = |\mathcal{C}_q| \qquad \forall q \in Q$$

$$x_{i,j}^q \ge 0 \qquad 0 \le i,j \le r+1, \forall q \in Q$$

We are about to minimize the following objective

$$\sum_{0 \le i \le r+1; 0 \le j \le r} \sum_{1 \le q \le 2r} (q-j) x_{i,j}^q + \sum_{0 \le i \le r+1} \sum_{\forall q} i \cdot x_{i,r+1}^q + g\left(x_{r+1,0}^{\max}, \dots, x_{r+1,r}^{\max}\right),$$

where $g \colon \mathbb{N}^r \to \mathbb{N}$ is a function that has access to sizes of all cliques in $\mathcal{C}_{\max}$ and computes the minimum possible assignment. We claim that the function g is a separable convex function in variables $x_{r+1,0}^{\max}, \dots, x_{r+1,r}^{\max}$. The first summand of the objective function describes the cliques of size at most $2r$. Their price corresponds to the number of vertices in the clique q minus the number of vertices that are not selected j. The second summand corresponds to the last row, where the cheapest price is always the number of selected vertices i. The last summand, discussed in the following paragraph, describes the assignment to the last row. The result then follows from Theorem 10 as the number of integral variables is $O(r^3)$.

Observe that the value of $g\left(x_{r+1,0}^{\max}, \dots, x_{r+1,r}^{\max}\right)$ is equal to sum of sizes of cliques "assigned to the last row" minus $\sum_{j=0}^r j \cdot x_{r+1,j}^{\max}$. Now, $g\left(x_{r+1,0}^{\max}, \dots, x_{r+1,r}^{\max}\right) = g'\left(\sum_{j=0}^r x_{r+1,j}^{\max}\right) - \sum_{j=0}^r j \cdot x_{r+1,j}^{\max}$. Since all cliques in $\mathcal{C}_{\max}$ are eligible candidates to be assigned to the last row and since it is always cheaper to assign there those of the smallest size among them, we can define g' based only on the number of cliques that are assigned to the last row. This finishes the proof since g' is a convex function. We defer the details on polynomial space to the full version of the paper. ◀

Now we are ready to prove the main result of this section. It essentially follows by the exhaustive search among all possible shapes S such that φ is true under S and application of Lemma 11.

▶ **Proposition 12.** *Let $G = (V,E)$ be a graph with twin cover K of size k. For an* MSO$_1$ *formula $\varphi(X)$ with one free variable that represents the set to be deleted it is possible to find a set $W \subseteq V$ such that*

- *$\varphi(W)$ holds in G and*
- *the cost of W is minimized among all subset of V satisfying $\varphi(X)$*

in time $f(k, |\varphi|)|V|^{O(1)}$ for some computable function f.

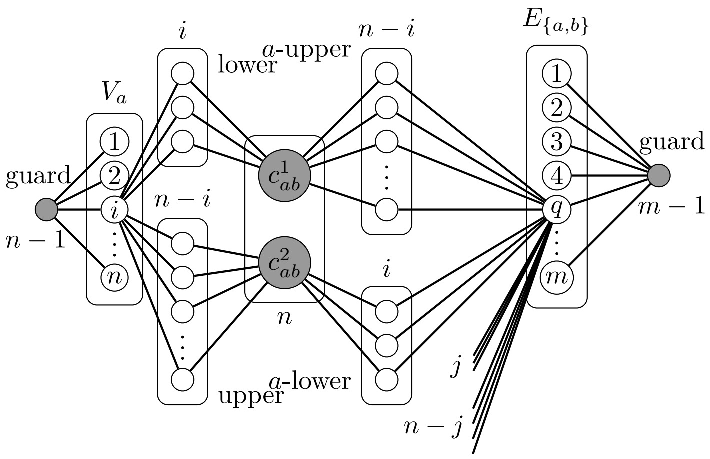


Figure 4 An overview of the reduction in the proof of Theorem 2. The gray vertices are enforced to be a part the fair vertex cover. If a vertex fair objective was lowered, then the resulting threshold is beneath the vertex (the group of vertices).

Proof sketch. We proceed as follows. For every possible selection of $K \cap W$ we generate all possible shapes and check whether $\varphi(X)$ evaluates to true under shape S and if so, we compute W for S having the minimal fair-cost. ◀

3 The Fair VC problem

3.1 Hardness for Treedepth and Feedback Vertex Set

We observe substantial connection between Fair VC and Target Set Selection (TSS). It is worth mentioning that Vertex Cover can be formulated in the language of TSS by setting the threshold to $\deg(v)$ for every vertex v. As a result, our reduction given here is, in certain sense, dual to the one given by Chopin et al. [3] for the TSS problem. However, we will show that the structure of the solution for Fair VC is, in fact, the complement of the structure of the solution for TSS given therein. The archetypal W[1]-hard problem is the ℓ-Multicolored Clique problem [7]:

ℓ-Multicolored Clique *Parameter:* ℓ
Input: An ℓ-partite graph $G = (V_1 \cup \cdots \cup V_\ell, E)$, where V_c is an independent set for every $c \in [\ell]$ and they are pairwise disjoint.
Question: Is there a clique of the size ℓ in G?

Proof sketch of Theorem 2. Refer to figure 4. Observe that we can enforce a vertex v to be a part of the fair vertex cover by attaching $k+1$ degree 1 vertices to v. Notice further that we may adjust (lower) the global budget k for individual vertex v by attaching vertices to v and then attaching k new leaves to the newly added vertices.

There are three types of gadgets in our reduction, namely the vertex selection gadget, the edge selection gadget, and the incidence check gadget. We start by enumerating the vertices in each color class by numbers from $[n]$ and edges by numbers in $[m]$. Now, we construct a graph H in which we are going to look for a vertex cover of small fair cost. Throughout the proof a, b are distinct numbers from $[\ell]$.

A selection gadget consists of z *choice* vertices (representing either the color class V_a with $z = n$ or $E_{\{a,b\}}$ in which case $z = m$), a special vertex called *guard*, and a group of n^2 *enumeration* vertices. The guard vertex is connected to all choice vertices, it is enforced to be a part of the fair vertex cover, and its budget is lowered so that at most $z - 1$ choice vertices can be in any fair vertex cover. The i-th choice vertex is connected to n private enumeration vertices. We further divide these vertices into two parts – the *lower part* consists of q vertices and the *upper part* consists of $n - q$ vertices, where q refers to the vertex number. That is, $q = i$ in case of a vertex choice gadget and for an edge choice gadget, we let $q = v_i$, where v_i is the number of the vertex incident to i-th edge in the corresponding colorclass.

The ab-incidence check gadget consist of two vertices c^1_{ab} and c^2_{ab}. Both c^1_{ab} and c^2_{ab} are enforced to be a part of the solution and with a lowered budget in a way that at most n vertices in the neighborhood of each of them can be part of any fair vertex cover. The vertex c^1_{ab} is connected to every lower part vertex in the selection gadget for V_a and to every upper a-part vertex in the selection gadget for $E_{\{a,b\}}$. For c^2_{ab} we exchange the role of upper- and lower- parts.

▷ Claim 13. Suppose (G, ℓ) is a yes-instance then there is a vertex cover in H with fair cost at most $k = \max(m - 1, 2n)$.

Let $K \subseteq V_1 \times \cdots \times V_\ell$ be a clique in G. We now construct a vertex cover C_K of H having $|N(w) \cap C_K| \le k$ for all $w \in W$. The set C_K contains the following:

1. all enforced vertices (including all guard and check vertices),
2. if $v \in V_a \cap K$ is the i-th vertex of V_a, then all selection vertices of V_a but the vertex i are in C_K and lower and upper enumeration vertices of i are in C_K, and
3. if $v \in V_a \cap K$ and $u \in V_b \cap K$ are adjacent through q-th edge of $E_{\{a,b\}}$, then all selection vertices of $E_{\{a,b\}}$ but the vertex q are in C_K and q's enumeration vertices are in C_K.

The proof of the reverse direction is deferred to the full version due to space limitations.

It remains to discuss the ETH based lower-bound. This follows immediately from our reduction and the result of Chen et al. [2] who proved that there is no $f(k)n^{o(\ell)}$ algorithm for ℓ-Multicolored Clique unless ETH fails. Since we have $\mathrm{td}(G) + \mathrm{fvs}(G) = O(\ell^2)$ in our reduction, the lower-bound follows. ◀

3.2 FPT algorithm for Modular Width

Since an algebraic expression A of width $\mathrm{mw}(G)$ can be computed in linear time [36], we can assume that we have A on the input. We construct the rooted ordered tree $\mathcal{T}$ corresponding to A. Each node $t \in \mathcal{T}$ is assigned a graph $G^t \subseteq G$, that is, the graph constructed by the subexpression of A rooted at t. Suppose we are performing substitution operation at node t with respect to template graph T and graphs $G_1, \dots, G_r$. Denote the resulting graph G^t and denote by n_i the size of $V(G_i)$.

Proof sketch of Theorem 3. The computation will be carried out recursively from the bottom of the tree $\mathcal{T}$. We first describe the structure of all vertex covers C in G^t. Observe that if $ij \in E(T)$, then at least one of $V(G_i), V(G_j)$ must be a subset of C. Thus, the set $C_T := \{i : V(G_i) \subseteq C\}$ is a vertex cover of the template graph T. We call the C_T the *type of the vertex cover* C. Furthermore, every set $C \cap V(G_i)$ must be a vertex cover of G_i. Since there are at most 2^r vertex covers of T, we try all of these. Furthermore, every set $C \cap V(G_i)$ must be a vertex cover of G_i.

We now describe the fair cost of the cover C in terms of fair costs and sizes of the sets $C \cap V(G_i)$. Let $c_i = |C \cap V(G_i)|$ and let f_i denote the fair cost of $C \cap V(G_i)$ in G_i. The *fair cost of* C *in* $W \subseteq V(G)$ is defined as $\max_{v \in W} |C \cap N(v)|$. For $i \in [r]$ the fair cost of C in $V(G_i)$ can be expressed as $f_i + \sum_{j : ij \in E(T)} c_j$. If we know the type C_T of the cover C, this expression can be simplified based on whether i lies in C_T or not. If $i \in C_T$, then f_i is $\Delta(G_i)$ (the maximal degree of G_i). If, on the other hand, $i \notin C_T$, then all its neighbors j are in C_T and in this case $c_j = n_j$. Combining these observations together we get

$$\text{fair cost of } C \text{ in } G_i = \begin{cases} \Delta(G_i) + \sum_{j \notin C_T : ij \in E(T)} c_j + \sum_{j \in C_T : ij \in E(T)} n_j & i \in C_T, \\ f_i + \sum_{j : ij \in E(T)} n_j & i \notin C_T. \end{cases}$$

From this we can design a dynamic program that computes the solution. ◀

4 Conclusions

Fair Edge L Deletion problems. One can define *edge deletion problems* in a similar way as vertex deletion problems.

FAIR L EDGE DELETION	
Input:	An undirected graph $G = (V, E)$, an L sentence φ, and a positive integer k.
Question:	Is there a set $F \subseteq E$ such that $G \setminus F \models \varphi$ and for every vertex v of G, it holds that $|\{e \in F : e \ni v\}| \le k$?

Recall, in dense graph classes one cannot obtain an MSO_2 model checking algorithm running in FPT-time [25]. This is the reason why evaluation problems do not make sense in this context. In sparse graph classes, this problem was studied in [29] where W[1]-hardness was obtained for Fair FO Edge Deletion on graphs of bounded treedepth and FPT algorithm was derived for Fair MSO_2 Edge Evaluation on graphs of bounded vertex cover.

The crucial open problem is to resolve the parameterized complexity of the FAIR FO EDGE DELETION problems for parameterization by neighborhood diversity and twin cover. To motivate the study we prove the following hardness result for parameterization by the cluster vertex deletion number. Recall that for any graph its cluster vertex deletion number is upper-bounded by the size of its twin cover.

▶ **Theorem 14** (★). *The* FAIR FO EDGE DELETION *problem is* W[1]*-hard when parameterized by the cluster vertex deletion number of the input graph.*

Generalization of parameters. Another open problem is to resolve the parameterized complexity of the FAIR MSO_1 VERTEX EVALUATION problems with respect to graph parameters generalizing neighborhood diversity or twin cover (e.g., modular width or cluster vertex deletion number respectively).

MSO with Local Linear Constraints. Previously, an FPT algorithm for evaluation of a fair objective was given for parameter neighborhood diversity [29]. That algorithm was extended [20] to a so-called *local linear constraints* again for a formula $\varphi(\cdot)$ with one free variable that is defined as follows. Every vertex $v \in V(G)$ is accompanied with two positive integers $\ell(v), u(v)$, the lower and the upper bound, and the task is to find a set X that not only $G \models \varphi(X)$ but for each $v \in V(G)$ it holds that $\ell(v) \le |N(v) \cap X| \le u(v)$. Note that this is a generalization as fair objective of value t can be tested in this framework by setting $\ell(v) = 0$ and $u(v) = t$ for every $v \in V(G)$. Is this extension possible for parameterization by the twin cover number?

Towards new fair problems. As we proposed the examination of Fair VC already, we would like to turn an attention to exploring fair versions of other classical and well-studied Vertex Deletion problems. In contrast, certain Fair Edge Deletion problems have got some attention before, namely Fair Feedback Edge Set [27] or Fair Edge Odd Cycle Transversal [22]. Besides Fair VC we propose a study of Fair Dominating Set and Fair Feedback Vertex Set. In particular, it would be really interesting to know whether fair variants of Vertex Cover and Dominating Set admit a similar behavior as in the classical setting.

Furthemore, We would like to ask whether there is an NP-hard Fair Vertex Deletion problem that admits an FPT algorithm for parameterization by treedepth (and feedback vertex set) of the input graph.

References

1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy Problems for Tree-Decomposable Graphs. *Journal of Algorithms*, 12(2):308–340, June 1991. doi:10.1016/0196-6774(91)90006-k.

2 Jianer Chen, Benny Chor, Michael R. Fellows, Xiuzhen Huang, David Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005. doi:10.1016/j.ic.2005.05.001.

3 Morgan Chopin, André Nichterlein, Rolf Niedermeier, and Mathias Weller. Constant Thresholds Can Make Target Set Selection Tractable. *Theory Comput. Syst.*, 55(1):61–83, 2014. doi:10.1007/s00224-013-9499-3.

4 Bruno Courcelle. The Monadic Second-order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1):12–75, March 1990. doi:10.1016/0890-5401(90)90043-h.

5 Bruno Courcelle and Mohamed Mosbah. Monadic Second-Order Evaluations on Tree-Decomposable Graphs. *Theor. Comput. Sci.*, 109(1&2):49–82, 1993. doi:10.1016/0304-3975(93)90064-z.

6 Lenore J. Cowen, Robert Cowen, and Douglas R. Woodall. Defective colorings of graphs in surfaces: Partitions into subgraphs of bounded valency. *Journal of Graph Theory*, 10(2):187–195, 1986. doi:10.1002/jgt.3190100207.

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

8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

9 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1):3–31, 2004. Papers presented at the 2002 IEEE Symposium on Logic in Computer Science (LICS). doi:10.1016/j.apal.2004.01.007.

10 Jakub Gajarský and Petr Hliněný. Kernelizing MSO Properties of Trees of Fixed Height, and Some Consequences. *Logical Methods in Computer Science*, Volume 11, Issue 1, April 2015. doi:10.2168/LMCS-11(1:19)2015.

11 Jakub Gajarský, Petr Hliněný, Jan Obdržálek, Daniel Lokshtanov, and M. S. Ramanujan. A New Perspective on FO Model Checking of Dense Graph Classes. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5–8, 2016*, pages 176–184. ACM, 2016. doi:10.1145/2933575.2935314.

12 Robert Ganian. Twin-Cover: Beyond Vertex Cover in Parameterized Algorithmics. In Dániel Marx and Peter Rossmanith, editors, *Parameterized and Exact Computation - 6th International Symposium IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised*

Selected Papers, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011. doi:10.1007/978-3-642-28050-4_21.

13 Robert Ganian. Improving Vertex Cover as a Graph Parameter. *Discrete Mathematics & Theoretical Computer Science*, 17(2):77–100, 2015. URL: http://dmtcs.episciences.org/2136.

14 Robert Ganian, Petr Hliněný, Jaroslav Nešetřil, Jan Obdržálek, and Patrice Ossona de Mendez. Shrub-depth: Capturing Height of Dense Graphs. *Logical Methods in Computer Science*, 15(1), 2019. URL: https://lmcs.episciences.org/5149.

15 Robert Ganian and Jan Obdržálek. Expanding the Expressive Power of Monadic Second-Order Logic on Restricted Graph Classes. In Thierry Lecroq and Laurent Mouchard, editors, *Combinatorial Algorithms—24th International Workshop, IWOCA 2013, Revised Selected Papers*, volume 8288 of *Lecture Notes in Computer Science*, pages 164–177. Springer, 2013. doi:10.1007/978-3-642-45278-9_15.

16 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding First-Order Properties of Nowhere Dense Graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

17 Frédéric Havet, Ross J. Kang, and Jean-Sébastien Sereni. Improper coloring of unit disk graphs. *Networks*, 54(3):150–164, 2009. doi:10.1002/net.20318.

18 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *J. Comput. Syst. Sci.*, 79(1):39–49, 2013. doi:10.1016/j.jcss.2012.04.004.

19 Ross J. Kang, Tobias Müller, and Jean-Sébastien Sereni. Improper colouring of (random) unit disk graphs. *Discrete Mathematics*, 308(8):1438–1454, 2008. Third European Conference on Combinatorics. doi:10.1016/j.disc.2007.07.070.

20 Dušan Knop, Martin Koutecký, Tomáš Masařík, and Tomáš Toufar. Simplified Algorithmic Metatheorems Beyond MSO: Treewidth and Neighborhood Diversity. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Graph-Theoretic Concepts in Computer Science: 43rd International Workshop, WG 2017, Eindhoven, The Netherlands, June 21-23, 2017, Revised Selected Papers*, pages 344–357, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-68705-6_26.

21 Petr Kolman, Bernard Lidický, and Jean-Sébastien Sereni. Fair Edge Deletion Problems on TreeDecomposable Graphs and Improper Colorings, 2010. URL: http://orion.math.iastate.edu/lidicky/pub/kls10.pdf.

22 Petr Kolman, Bernard Lidický, and Jean-Sébastien Sereni. On Fair Edge Deletion Problems, 2009. URL: https://kam.mff.cuni.cz/~kolman/papers/kls09.pdf.

23 Juha Kontinen and Hannu Niemistö. Extensions of MSO and the monadic counting hierarchy. *Information and Computation*, 209(1):1–19, 2011. doi:10.1016/j.ic.2010.09.002.

24 Michael Lampis. Algorithmic Meta-theorems for Restrictions of Treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.

25 Michael Lampis. Model Checking Lower Bounds for Simple Graphs. *Logical Methods in Computer Science*, 10(1):1–21, 2014. doi:10.2168/LMCS-10(1:18)2014.

26 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

27 Li-Shin Lin and Sartaj Sahni. Fair Edge Deletion Problems. *IEEE Trans. Comput.*, 38(5):756–761, 1989. doi:10.1109/12.24280.

28 Tomáš Masařík and Tomáš Toufar. Parameterized Complexity of Fair Deletion Problems. In T.V. Gopal, Gerhard Jäger, and Silvia Steila, editors, *Theory and Applications of Models of Computation: 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, pages 628–642, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-55911-7_45.

29 Tomáš Masařík and Tomáš Toufar. Parameterized complexity of fair deletion problems. *Discrete Applied Mathematics*, 2019. doi:10.1016/j.dam.2019.06.001.

30 Jiří Matoušek and Jaroslav Nešetřil. *Invitation to Discrete Mathematics (2. ed.)*. Oxford University Press, 2009.

31 Jaroslav Nešetřil and Patrice Ossona De Mendez. *Sparsity: graphs, structures, and algorithms*, volume 28. Springer Science & Business Media, 2012. doi:10.1007/978-3-642-27875-4.

32 Timm Oertel, Christian Wagner, and Robert Weismantel. Integer convex minimization by mixed integer linear optimization. *Operations Research Letters*, 42(6):424–428, 2014. doi:10.1016/j.orl.2014.07.005.

33 Michał Pilipczuk. Problems Parameterized by Treewidth Tractable in Single Exponential Time: A Logical Approach. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011. doi:10.1007/978-3-642-22993-0_47.

34 Detlef Seese. Linear Time Computable Problems and First-Order Descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.

35 Stefan Szeider. Monadic second order logic on graphs with local cardinality constraints. *ACM Trans. Comput. Log*, 12(2):1–21, 2011. doi:10.1145/1877714.1877718.

36 Marc Tedder, Dereck G. Corneil, Michel Habib, and Christophe Paul. Simpler Linear-Time Modular Decomposition Via Recursive Factorizing Permutations. In *ICALP 2008*, pages 634–645, 2008. doi:10.1007/978-3-540-70575-8_52.

37 Mihalis Yannakakis. Edge-Deletion Problems. *SIAM J. Comput.*, 10(2):297–309, 1981. doi:10.1137/0210021.

A Complexity Dichotomy for Critical Values of the b-Chromatic Number of Graphs

Lars Jaffke
Department of Informatics, University of Bergen, Norway
lars.jaffke@uib.no

Paloma T. Lima
Department of Informatics, University of Bergen, Norway
paloma.lima@uib.no

Abstract

A *b-coloring* of a graph G is a proper coloring of its vertices such that each color class contains a vertex that has at least one neighbor in all the other color classes. The b-Coloring problem asks whether a graph G has a b-coloring with k colors. The *b-chromatic number* of a graph G, denoted by $\chi_b(G)$, is the maximum number k such that G admits a b-coloring with k colors. We consider the complexity of the b-Coloring problem, whenever the value of k is close to one of two upper bounds on $\chi_b(G)$: The maximum degree $\Delta(G)$ plus one, and the m-degree, denoted by $m(G)$, which is defined as the maximum number i such that G has i vertices of degree at least $i-1$. We obtain a dichotomy result for all fixed $k \in \mathbb{N}$ when k is close to one of the two above mentioned upper bounds. Concretely, we show that if $k \in \{\Delta(G)+1-p, m(G)-p\}$, the problem is polynomial-time solvable whenever $p \in \{0,1\}$ and, even when $k = 3$, it is NP-complete whenever $p \geq 2$. We furthermore consider parameterizations of the b-Coloring problem that involve the maximum degree $\Delta(G)$ of the input graph G and give two FPT-algorithms. First, we show that deciding whether a graph G has a b-coloring with $m(G)$ colors is FPT parameterized by $\Delta(G)$. Second, we show that b-Coloring is FPT parameterized by $\Delta(G) + \ell_k(G)$, where $\ell_k(G)$ denotes the number of vertices of degree at least k.

2012 ACM Subject Classification Mathematics of computing → Graph coloring

Keywords and phrases b-Coloring, b-Chromatic Number

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.34

Related Version A full version is available at: `https://arxiv.org/abs/1811.03966` [13].

Funding *Lars Jaffke*: Supported by The Bergen Research Foundation (BFS).
Paloma T. Lima: Supported by Research Council of Norway via the project "CLASSIS".

Acknowledgements We would like to thank Fedor Fomin for useful advice with good timing and Petr Golovach for pointing out the reference [7].

1 Introduction

Given a set of colors, a *proper coloring* of a graph is an assignment of a color to each of its vertices in such a way that no pair of adjacent vertices receive the same color. In the deeply studied Graph Coloring problem, we are given a graph and the question is to determine the smallest set of colors with which we can properly color the input graph. This problem is among Karp's famous list of 21 NP-complete problems [14] and since it often arises in practice, heuristics to solve it are deployed in a wide range of applications. A very natural such heuristic is the following. We greedily find a proper coloring of the graph, and then try to *suppress* any of its colors in the following way: say we want to suppress color c. If there is a vertex v that has received color c, and there is another color $c' \neq c$ that does not appear in the neighborhood of v, then we can safely recolor the vertex v with color c' without

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 34; pp. 34:1–34:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

making the coloring improper. We terminate this process once we cannot suppress any color anymore.

To predict the worst-case behavior of the above heuristic, Irving and Manlove defined the notions of a *b-coloring* and the *b-chromatic number* of a graph [12]. A b-coloring of a graph G is a proper coloring such that in every color class there is a vertex that has a neighbor in all of the remaining color classes, and the b-chromatic number of G, denoted by $\chi_b(G)$, is the maximum integer k such that G admits a b-coloring with k colors. We observe that in a b-coloring with k colors, there is no color that can be suppressed to obtain a proper coloring with $k-1$ colors, hence $\chi_b(G)$ describes the worst-case behavior of the previously described heuristic on the graph G. We consider the following two computational problems associated with b-colorings of graphs.

b-Coloring

Input:	Graph G, integer k
Question:	Does G admit a b-coloring with k colors?

b-Chromatic Number

Input:	Graph G, integer k
Question:	Is $\chi_b(G) \geq k$?

We would like to point out an important distinction from the "standard" notion of proper colorings of graphs: If a graph G has a b-coloring with k colors, then this implies that $\chi_b(G) \geq k$. However, if $\chi_b(G) \geq k$ then we can in general not conclude that G has a b-coloring with k colors. A graph for which the latter implication holds as well is called *b-continuous*. This notion is mostly of structural interest, since the problem of determining if a graph is b-continous is NP-complete even if an optimal proper coloring and a b-coloring with $\chi_b(G)$ colors are given [2].

Besides observing that $\chi_b(G) \leq \Delta(G)+1$ where $\Delta(G)$ denotes the maximum degree of G, Irving and Manlove [12] defined the *m-degree* of G as the largest integer i such that G has i vertices of degree at least $i-1$. It follows that $\chi_b(G) \leq m(G)$. Since the definition of the b-chromatic number originated in the analysis of the worst-case behavior of graph coloring heuristics, graphs whose b-chromatic numbers take on *critical* values, i.e. values that are close to these upper bounds, are of special interest. In particular, identifying them can be helpful in structural investigations concerning the performance of graph coloring heuristics.

In terms of computational complexity, Irving and Manlove showed that both b-Coloring and b-Chromatic Number are NP-complete [12] and Sampaio observed that b-Coloring is NP-complete even for every fixed integer $k \geq 3$ [17]. Panolan et al. [16] gave an exact exponential algorithm for b-Chromatic Number running in time $\mathcal{O}(3^n n^4 \log n)$ and an algorithm that solves b-Coloring in time $\mathcal{O}(\binom{n}{k} 2^{n-k} n^4 \log n)$. From the perspective of parameterized complexity [6, 8], it has been shown that b-Chromatic Number is W[1]-hard parameterized by k [16] and that the dual problem of deciding whether $\chi_b(G) \geq n-k$, where n denotes the number of vertices in G, is FPT parameterized by k [11].

Since the above mentioned upper bounds $\Delta(G)+1$ and $m(G)$ on the b-chromatic number are trivial to compute, it is natural to ask whether there exist efficient algorithms that decide whether $\chi_b(G) = \Delta(G)+1$ or $\chi_b(G) = m(G)$. It turns out both these problems are NP-complete as well [10, 12, 15]. However, it is known that the problem of deciding whether a graph G admits a b-coloring with $k = \Delta(G)+1$ colors is FPT parameterized by k [16, 17].

The Dichotomy Result. One of the main contributions of this paper is a complexity dichotomy of the b-Coloring problem for fixed k, whenever k is close to either $\Delta(G)+1$ or $m(G)$. In particular, for fixed $k \in \{\Delta(G)+1-p, m(G)-p\}$, we show that the problem is polynomial-time solvable when $p \in \{0,1\}$ and, even in the case $k=3$, NP-complete for all fixed $p \geq 2$. More specifically, we give XP time algorithms for the cases $k=m(G)$, $k=\Delta(G)$, and $k=m(G)-1$ which together with the FPT algorithm for the case $k=\Delta(G)+1$ [16, 17] and the aforementioned NP-hardness result for $k=3$ complete the picture. We now formally state this result.

▶ **Theorem 1.** *Let G be a graph, $p \in \mathbb{N}$ and $k \in \{\Delta(G)+1-p, m(G)-p\}$. The problem of deciding whether G has a b-coloring with k colors is*

(i) NP*-complete if k is part of the input and $p \in \{0,1\}$,*
(ii) NP*-complete if $k=3$ and $p \geq 2$, and*
(iii) *polynomial-time solvable for any fixed positive k and $p \in \{0,1\}$.*

Maximum Degree Parameterizations. The positive results in our dichotomy theorem provide XP-algorithms to decide whether a graph has a b-coloring with a number of colors that either *precisely meets* or is *one below* one of two upper bounds on the b-chromatic number, with the parameter being the number of colors in each of the cases. Towards more "flexible" tractability results, we consider parameterized versions of b-Coloring that involve the maximum degree $\Delta(G)$ of the input graph G, but ask for the existence of b-colorings with a number of colors that in general is different from $\Delta(G)+1$ or $\Delta(G)$.

▶ **Theorem 2.** *Let G be a graph. The problem of deciding whether G has a b-coloring with $m(G)$ colors is* FPT *parameterized by $\Delta(G)$.*

One of the crucially used facts in the algorithm of the previous theorem is that if we ask whether a graph G has a b-coloring with $k=m(G)$ colors, then the number of vertices of degree at least k is at most k. We generalize this setting and parameterize b-Coloring by the maximum degree plus the number of vertices of degree at least k.

▶ **Theorem 3.** *Let G be a graph. The problem of deciding whether G has a b-coloring with k colors is* FPT *parameterized by $\Delta(G)+\ell_k(G)$, where $\ell_k(G)$ denotes the number of vertices of degree at least k in G.*

We now argue that parameterizing by only one of the two invariants used in Theorem 3 is not sufficient to obtain efficient parameterized algorithms. From the result of Kratochvíl et al. [15], stating that b-Coloring is NP-complete for $k=\Delta(G)+1$, it follows that b-Coloring is NP-complete when $\Delta(G)$ is unbounded and $\ell_k(G)=0$. On the other hand, Theorem 1(ii) implies that b-Coloring is already NP-complete when $k=3$ and $\Delta(G)=4$. Together, this rules out the possibility of FPT- and even of XP-algorithms for parameterizations by one of the two parameters alone, unless P $=$ NP.

Parameterizations of graph coloring problems by the number of high degree vertices have previously been considered for vertex coloring [1] and edge coloring [9]. Throughout the text, proofs of statements marked with "♣" are deferred to the full version [13].

2 Preliminaries

We use the following notation: For $k \in \mathbb{N}$, $[k] := \{1, \ldots, k\}$. For a function $f\colon X \to Y$ and $X' \subseteq X$, we denote by $f|_{X'}$ the restriction of f to X' and by $f(X')$ the set $\{f(x) \mid x \in X'\}$. For a set X and an integer n, we denote by $\binom{X}{n}$ the set of all size-n subsets of X.

Graphs. Throughout the paper a graph G with vertex set $V(G)$ and edge set $E(G) \subseteq \binom{V(G)}{2}$ is finite and simple. We often denote an edge $\{u, v\} \in E(G)$ by the shorthand uv. For graphs G and H we denote by $H \subseteq G$ that H is a subgraph of G, i.e. $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We often use the notation $n \coloneqq |V(G)|$. For a vertex $v \in V(G)$, we denote by $N_G(v)$ the *open neighborhood* of v in G, i.e. $N_G(v) = \{w \in V(G) \mid vw \in E(G)\}$, and by $N_G[v]$ the *closed neighborhood* of v in G, i.e. $N_G[v] \coloneqq \{v\} \cup N_G(v)$. For a set of vertices $X \subseteq V(G)$, we let $N_G(X) \coloneqq \bigcup_{v \in X} N_G(v) \setminus X$ and $N_G[X] \coloneqq X \cup N_G(X)$. When G is clear from the context, we abbreviate "N_G" to "N". The *degree* of a vertex $v \in V(G)$ is the size of its open neighborhood, and we denote it by $\deg_G(v) \coloneqq |N_G(v)|$ or simply by $\deg(v)$ if G is clear from the context. For an integer k, we denote by $\ell_k(G)$ the number of vertices of degree at least k in G.

For a vertex set $X \subseteq V(G)$, we denote by $G[X]$ the subgraph *induced* by X, i.e. $G[X] \coloneqq (X, E(G) \cap \binom{X}{2})$. We furthermore let $G - X \coloneqq G[V(G) \setminus X]$ be the subgraph of G obtained from removing the vertices in X and for a single vertex $x \in V(G)$, we use the shorthand "$G - x$" for "$G - \{x\}$".

A graph G is said to be *connected* if for any 2-partition (X, Y) of $V(G)$, there is an edge $xy \in E(G)$ such that $x \in X$ and $y \in Y$, and *disconnected* otherwise. A *connected component* of a graph G is a maximal connected subgraph of G. A *path* is a connected graph of maximum degree two, having precisely two vertices of degree one, called its *endpoints*. The *length* of a path is its number of edges. Given a graph G and two vertices u and v, the *distance* between u and v, denoted by $dist_G(u, v)$ (or simply $dist(u, v)$ if G is clear from the context), is the length of the shortest path in G that has u and v as endpoints.

A graph G is a *complete graph* if every pair of vertices of G is adjacent. A set $C \subseteq V(G)$ is a *clique* if $G[C]$ is a complete graph. A set $S \subseteq V(G)$ is an *independent set* if $G[S]$ has no edges. A graph G is a *bipartite graph* if its vertex set can be partitioned into two independent sets. A bipartite graph with bipartition (A, B) is a *complete bipartite graph* if all pairs consisting of one vertex from A and one vertex from B are adjacent, and with $a = |A|$ and $b = |B|$, we denote it by $K_{a,b}$. A *star* is the graph $K_{1,b}$, with $b \geq 2$, and we call *center* the unique vertex of degree b and *leaves* the vertices of degree one.

Colorings. Given a graph G, a map $\gamma \colon V(G) \to [k]$ is called a *coloring of G with k colors*. If for every pair of adjacent vertices, $uv \in E(G)$, we have that $\gamma(u) \neq \gamma(v)$, then the coloring γ is called *proper*. For $i \in [k]$, we call the set of vertices $u \in V(G)$ such that $\gamma(u) = i$ the *color class i*. If for all $i \in [k]$, there exists a vertex $x_i \in V(G)$ such that

(i) $\gamma(x_i) = i$, and

(ii) for each $j \in [k] \setminus \{i\}$, there is a neighbor $y \in N_G(x_i)$ of x_i such that $\gamma(y) = j$,

then γ is called a *b-coloring* of G. For $i \in [k]$, we call a vertex x_i satisfying the above two conditions a *b-vertex for color i*.

Parameterized Complexity. Let Σ be an alphabet. A *parameterized problem* is a set $\Pi \subseteq \Sigma^* \times \mathbb{N}$. A parameterized problem Π is said to be *fixed-parameter tractable*, or contained in the complexity class FPT, if there exists an algorithm that for each $(x, k) \in \Sigma^* \times \mathbb{N}$ decides whether $(x, k) \in \Pi$ in time $f(k) \cdot |x|^c$ for some computable function f and fixed integer $c \in \mathbb{N}$. A parameterized problem Π is said to be contained in the complexity class XP if there is an algorithm that for all $(x, k) \in \Sigma^* \times \mathbb{N}$ decides whether $(x, k) \in \Pi$ in time $f(k) \cdot n^{g(k)}$ for some computable functions f and g.

A *kernelization algorithm* for a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is a polynomial-time algorithm that takes as input an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ and either correctly decides whether

$(x,k) \in \Pi$ or outputs an instance $(x',k') \in \Sigma^* \times \mathbb{N}$ with $|x'| + k' \leq f(k)$ for some computable function f for which $(x,k) \in \Pi$ if and only if $(x',k') \in \Pi$. We say that Π *admits a kernel* if there is a kernelization algorithm for Π.

3 Hardness Results

In this section we prove the hardness results of our complexity dichotomy. First, we show that b-Chromatic Number and b-Coloring are NP-complete for $k = m(G) - 1 = \Delta(G)$, based on a reduction for the case $k = m(G)$ due to Havet et al. [10].

▶ **Theorem 4 (♣).** *b-Chromatic Number and b-Coloring are NP-complete, even when $k = m(G) - 1 = \Delta(G)$.*

The previous theorem, together with the result that b-Coloring is NP-complete when $k = \Delta(G) + 1$ [15] and when $k = m(G)$ [10], proves Theorem 1(i). We now turn to the proof of Theorem 1(ii), that is, we show that b-Coloring remains NP-complete for $k = 3$ if $k = \Delta(G) + 1 - p$ or $k = m(G) - p$ for any $p \geq 2$, based on a reduction due to Sampaio [17]. Note that the following proposition indeed proves Theorem 1(ii) as for fixed $p \geq 2$, we have that $3 \in \{\Delta(G) + 1 - p, m(G) - p\}$ if and only if $\Delta(G) = p + 2$ or $m(G) = p + 3$.

▶ **Proposition 5 (♣).** *For every fixed integer $p \geq 2$, the problem of deciding whether a graph G has a b-coloring with 3 colors is NP-complete when $\Delta(G) = p + 2$ or $m(G) = p + 3$.*

Since b-Chromatic Number and b-Coloring are known to be NP-complete when $k = \Delta(G) + 1$ [15], we make the following observation which is of relevance to us since in Section 5.2, we show that b-Coloring is FPT parameterized by $\Delta(G) + \ell_k(G)$.

▶ **Observation 6.** *b-Chromatic Number and b-Coloring are NP-complete on graphs with $\ell_k(G) = 0$, where k is the integer associated with the respective problem.*

4 Dichotomy Algorithms

In this section we give the algorithms in our dichotomy result, proving Theorem 1(iii). We show that for fixed $k \in \mathbb{N}$, the problem of deciding whether a graph G admits a b-coloring with k colors is polynomial-time solvable when $k = m(G)$ (Sect. 4.2), when $k = \Delta(G)$ (Sect. 4.3), and when $k = m(G) - 1$ (Sect. 4.4), by providing XP-algorithms for each case.

A natural way of solving the b-Coloring problem is to first try to identify a set of k b-vertices, color them bijectively with colors from $[k]$, and for each vertex in the set a set of $k - 1$ neighbors that can be colored in such a way that the vertex becomes a b-vertex for its color. Then try to extend the resulting coloring to the remainder of the graph. We enumerate all such sets and colorings, and show that the extension problem is solvable in polynomial time in each of the above cases.

The strategy of identifying the set of b-vertices and subsets of their neighbors that make them b-vertices was (for instance) also used to give polynomial-time algorithms to compute the b-chromatic number of trees [12] and graphs with large girth [4]. We capture it by defining the notion of a *b-precoloring* in the next subsection.

4.1 b-Precolorings

All algorithms in this section are based on guessing a proper coloring of several vertices in the graph, for which we now introduce the necessary terminology and establish some preliminary results.

▶ **Definition 7** (Precoloring). *Let G be a graph and $k \in \mathbb{N}$. A* precoloring *with k colors of a graph G is an assignment of colors to a subset of its vertices, i.e. for $X \subseteq V(G)$, it is a map $\gamma_X : X \to [k]$. We call γ_X* proper*, if it is a proper coloring of $G[X]$. We say that a coloring $\gamma : V(G) \to [k]$* extends *γ_X, if $\gamma|_X = \gamma_X$.*

We use the following notation. For two precolorings γ_X and γ_Y with $X \cap Y = \emptyset$, we denote by $\gamma_X \cup \gamma_Y$ the precoloring that colors the vertices in X according to γ_X and the vertices in Y according to Y, i.e. the precoloring $\gamma_{X\cup Y} \coloneqq \gamma_X \cup \gamma_Y$ defined as follows: for all $v \in X \cup Y$, if $v \in X$, then $\gamma_{X\cup Y}(v) = \gamma_X$, and if $v \in Y$ then $\gamma_{X\cup Y}(v) = \gamma_Y(v)$.

Next, we define a special type of precoloring with the property that any proper coloring that extends it is a b-coloring of the graph.

▶ **Definition 8** (b-Precoloring). *Let G be a graph, $k \in \mathbb{N}$, $X \subseteq V(G)$ and γ_X a precoloring. We call γ_X a* b-precoloring with k colors *if γ_X is a b-coloring of $G[X]$. A b-precoloring γ_X is called* minimal *if for any $Y \subset X$, $\gamma_X|_Y$ is not a b-precoloring.*

It is immediate that any b-coloring can be obtained by extending a minimal b-precoloring, a fact that we capture in the following observation.

▶ **Observation 9.** *Let G be a graph, $k \in \mathbb{N}$, and γ a b-coloring of G with k colors. Then, there is a set $X \subseteq V(G)$ such that $\gamma|_X$ is a minimal b-precoloring.*

The next observation captures the structure of minimal b-precolorings with k colors. Roughly speaking, each such precoloring only colors a set of k b-vertices and for each b-vertex a set of $k-1$ of its neighbors that make that vertex the b-vertex of its color. We will use this property in the enumeration algorithm in this section to guarantee that we indeed enumerate all minimal b-precolorings with a given number of colors.

▶ **Observation 10.** *Let γ_X be a minimal b-precoloring with k colors. Then, $X = B \cup Z$, where*

(i) *$B = \{x_1, \ldots, x_k\}$ and for $i \in [k]$, $\gamma_X(x_i) = i$, and*
(ii) *$Z = \bigcup_{i\in[k]} Z_i$, where $Z_i \in \binom{N(x_i)}{k-1}$ and $\gamma_X(Z_i) = [k] \setminus \{i\}$.*

We are now ready to give the enumeration algorithm for minimal b-precolorings.

▶ **Lemma 11** (♣). *Let G be a graph on n vertices and $k \in \mathbb{N}$. The number of minimal b-precolorings with k colors of G is at most*

$$\beta(k) \coloneqq n^k \cdot \Delta^{k(k-1)} \cdot (k-1)!^k, \tag{1}$$

where $\Delta \coloneqq \Delta(G)$ and they can be enumerated in time $\beta(k) \cdot k^{\mathcal{O}(1)}$.

4.2 Algorithm for $k = m(G)$

Our first application of Lemma 11 is to solve the b-Coloring problem in the case when $k = m(G)$ in time XP parameterized by k. It turns out that in this case, we are dealing with a Yes-instance as soon as we found a b-precoloring in the input graph that also colors all high-degree vertices (see Claim 12.1).

▶ **Theorem 12.** *Let G be a graph. There is an algorithm that decides whether G has a b-coloring with $k = m(G)$ colors in time $n^{k^2} \cdot 2^{\mathcal{O}(k^2 \log k)}$.*

Proof. Let $D \subseteq V(G)$ denote the set of vertices in G that have degree at least k. Note that by the definition of $m(G)$, we have that $|D| \leq k$.

▶ **Claim 12.1** (♣). *G has a b-coloring with k colors if and only if G has a b-precoloring γ_X such that $D \subseteq X$ and there exists $S \subseteq D$ such that $\gamma_X|_{(X \setminus S)}$ is a minimal b-precoloring.*

The algorithm enumerates all minimal b-precolorings with k colors and for each such precoloring, it enumerates all colorings of the vertices D. If combining one such pair of precolorings gives a b-precoloring, it returns a greedy extension of it; otherwise it reports that there is no b-coloring with k colors, see Algorithm 1.

Algorithm 1 Algorithm for b-Coloring with $k = m(G)$.

```
Input   : A graph $G$
Output : A $b$-coloring with $m(G)$ colors if it exists, No otherwise.
foreach minimal $b$-precoloring $\gamma_X : X \to [k]$ do
    foreach precoloring $\gamma_{D \setminus X} : (D \setminus X) \to [k]$ do
        if $\gamma_{X \cup D} := \gamma_X \cup \gamma_{D \setminus X}$ is proper then return a greedy extension of $\gamma_{X \cup D}$;
return No;
```

The correctness of the algorithm follows from the fact that it enumerates all precolorings that can satisfy Claim 12.1. We discuss its runtime. By Lemma 11, we can enumerate all minimal b-precolorings with k colors in time $\beta(k) \cdot k^{\mathcal{O}(1)}$. For each such minimal b-precoloring, we also enumerate all colorings of D. Since $|D| \le k$, this gives an additional factor of k^k to the runtime which (with $\Delta \le n$) then amounts to $\beta(k) \cdot k^k \cdot k^{\mathcal{O}(1)} \le n^{k^2} \cdot 2^{\mathcal{O}(k^2 \log k)}$. ◀

4.3 Algorithm for $k = \Delta(G)$

Next, we turn to the case when $k = \Delta(G)$. Here the strategy is to again enumerate all minimal b-precolorings, and then for each such precoloring we check whether it can be extended to the remainder of the graph. Formally, we use an algorithm for the following problem as a subroutine.

Precoloring Extension (PrExt)

Input: A graph G, an integer k, and a precoloring $\gamma_X : X \to [k]$ of a set $X \subseteq V(G)$

Question: Does G have a proper coloring with k colors extending γ_X?

Naturally, Precoloring Extension is a hard problem, since it includes Graph Coloring as the special case when $X = \emptyset$. However, when $\Delta(G) \le k - 1$, then the problem is trivially solvable: we simply check if the precoloring at the input is proper and if so, we compute an extension of it greedily. Since each vertex has degree at most $k - 1$, there is always at least one color available. The case when $k = \Delta(G)$ has also been shown to be solvable in polynomial time.

▶ **Theorem 13** (Thm. 3 in [5], see also [7]). *There is an algorithm that solves* Precoloring Extension *in polynomial time whenever $\Delta(G) \le k$.*

▶ **Theorem 14.** *There is an algorithm that decides whether a graph G has a b-coloring with $\Delta(G)$ colors in time $n^{k+\mathcal{O}(1)} \cdot 2^{\mathcal{O}(k^2 \log k)}$.*

Proof (sketch). For each minimal b-precoloring γ_X, we apply the algorithm for PrExt of Theorem 13. If it finds a proper coloring extending γ_X, we return it, and if there is no successful run of the algorithm for PrExt, we return No. The details are given in the full version [13]. ◀

4.4 Algorithm for $k = m(G) - 1$

Before we proceed to describe the algorithm for b-Coloring when $k = m(G) - 1$, we show that the algorithm of Theorem 13 can be used for a slightly more general case of Precoloring Extension, namely the case when all high-degree vertices in the input instance are precolored.

▶ **Lemma 15** (♣). *There is an algorithm that solves an instance (G, k, γ_X) of* Precoloring Extension *in polynomial time whenever $\max_{v \in V(G) \setminus X} \deg(v) \leq k$.*

▶ **Theorem 16.** *There is an algorithm that decides whether a graph G has a b-coloring with $k = m(G) - 1$ colors in time $n^{k^2 + \mathcal{O}(1)} \cdot 2^{k^2 \log k}$.*

Proof (sketch). Let D denote the set of vertices of degree at least $k + 1$ in G. By the definition of $m(G)$, we have that $|D| \leq k + 1$. We first enumerate all minimal b-precolorings of G, and for each such precoloring, we enumerate all precolorings of D. Since given a b-precoloring γ_X with $D \subseteq X$, we have that every vertex in $V(G) \setminus X$ has degree at most k, we can apply the algorithm of Lemma 15 to verify whether there is a proper coloring of G that extends γ_X. If so, we output that extension. If no such precoloring can be found, then we conclude that we are dealing with a No-instance. We give the details of the algorithm and its correctness proof in the full version [13].

It remains to argue the runtime. We enumerate $\beta(k)$ (see (1)) minimal b-precolorings in time $\beta(k) \cdot k^{\mathcal{O}(1)}$ using Lemma 11. For each such precoloring, we enumerate all precolorings of $D \setminus X$. Since $|D| \leq k + 1$, there are at most k^{k+1} such colorings. Finally, we run the algorithm for PrExt due to Lemma 15 which takes time $n^{\mathcal{O}(1)}$. The total runtime becomes $\beta(k) \cdot k^{\mathcal{O}(1)} \cdot k^{k+1} \cdot n^{\mathcal{O}(1)} \leq n^{k^2 + \mathcal{O}(1)} \cdot 2^{k^2 \log k}$. ◀

5 Maximum Degree Parameterizations

In this section we consider parameterizations of b-Coloring that involve the maximum degree $\Delta(G)$ of the input graph G. In Section 5.1 we show that we can solve b-Coloring when $k = m(G)$ in time FPT parameterized by $\Delta(G)$ and in Section 5.2 we show that b-Coloring is FPT parameterized by $\Delta(G) + \ell_k(G)$.

Both algorithms presented in this section make use of the following reduction rule, which has already been applied in [16, 17] to obtain the FPT algorithm for the problem of deciding whether a graph G has a b-coloring with $k = \Delta(G) + 1$ colors, parameterized by k.

▶ **Reduction Rule 17** ([16, 17]). *Let (G, k) be an instance of* b-Coloring. *If there is a vertex $v \in V(G)$ such that every vertex in $N[v]$ has degree at most $k - 2$, then reduce (G, k) to $(G - v, k)$.*

5.1 FPT Algorithm for $k = m(G)$ parameterized by $\Delta(G)$

Sampaio [17] and Panolan et al. [16] independently showed that parameterized by $\Delta(G)$, it can be decided in FPT time whether a graph G has a b-coloring with $\Delta(G) + 1$ colors. In this section we show that in the same parameterization, it can be decided in FPT time whether a graph has a b-coloring with $m(G)$ colors.

▶ **Theorem** (Thm. 2, restated). *There is an algorithm that given a graph G on n vertices decides whether G has a b-coloring with $k = m(G)$ colors in time $2^{\mathcal{O}(k^4 \cdot \Delta)} + n^{\mathcal{O}(1)} < 2^{\mathcal{O}(\Delta^5)} + n^{\mathcal{O}(1)}$, where $\Delta := \Delta(G)$.*

Proof. We apply Reduction Rule 17 exhaustively to G and consider the following 3-partition (D, T, R) of $V(G)$, where D contains the vertices of degree at least k, T the vertices of degree precisely $k-1$ and R the remaining vertices, i.e. $R := V(G) \setminus (D \cup T)$. Since we applied Reduction Rule 17 exhaustively, we make

▶ **Observation 2.1.** *Every vertex in R has at least one neighbor in $D \cup T$.*

We pick an inclusion-wise maximal set $B \subseteq D \cup T$ such that for each pair of distinct vertices $b_1, b_2 \in B$, we have that $dist(b_1, b_2) \geq 4$.

Case 1 ($|B \cap T| < k$).[1] We show that for any vertex in $u \in V(G) \setminus B$, there is a vertex $v \in B$ such that $dist(u, v) \leq 4$. Suppose $u \in D \cup T$. Since we did not include u in B, it immediately follows that there is some $v \in B$ such that $dist(u, v) < 4$. Now suppose $u \in R$. By Observation 2.1, u has a neighbor w in $D \cup T$ and by the previous argument, there is a vertex $v \in B$ such that $dist(w, v) < 4$. We conclude that $dist(u, v) \leq 4$. Using this observation, we now show that in this case, the number of vertices in G is polynomial in k and Δ.

▶ **Claim 2.2.** *If $|B \cap T| < k$, then $|V(G)| \leq \mathcal{O}(k^4 \cdot \Delta)$.*

Proof. Note that $(B \cup D, S_1, \ldots, S_4)$ constitutes a partition of $V(G)$, where S_i is the set of vertices of $V(G) \setminus (B \cup D)$ that are at distance exactly i from B. Since $|B \cap T| < k$ and $|D| \leq k$, we have that $|B \cup D| < 2k$, and therefore $|S_1| < 2k \cdot \Delta$. By the definition of $m(G)$, all the vertices in $S_1 \cup \ldots \cup S_4$ have degree at most $k-1$. This implies that $|S_i| < (k-1)^{i-1} \cdot 2k \cdot \Delta$. We conclude that the number of vertices in G is at most $2k + 2k \cdot \Delta \cdot \sum_{i=1}^{4} (k-1)^{i-1} = \mathcal{O}(k^4 \cdot \Delta)$. ◁

By Claim 2.2, we can solve the instance in Case 1 in time $2^{\mathcal{O}(k^4 \cdot \Delta)}$ using the algorithm of Panolan et al. [16].

Case 2 ($|B \cap T| \geq k$). Let $B' \subseteq B \cap T$ with $|B'| = k$ and denote this set by $B' = \{x_1, x_2, \ldots, x_k\}$. We show that we can construct a b-coloring $\gamma \colon V(G) \to [k]$ of G such that for $i \in [k]$, x_i is the b-vertex of color i. For $i \in [k]$, we let $\gamma(x_i) := i$. Next, we color the vertices in D. Recall that $|D| \leq k$, so we can color the vertices in D injectively with colors from $[k]$, ensuring that this will not create a conflict on any edge in $G[D]$. Furthermore, consider $i, j \in [k]$ with $i \neq j$. Since $dist(x_i, x_j) \geq 4$, we have that $N(x_i) \cap N(x_j) = \emptyset$. In particular, there is no vertex in D that has two or more neighbors in B'. To summarize, we can conclude that we can let γ color the vertices of D in such a way that:

(i) γ is injective on D, and

(ii) γ is a proper coloring of $G[B' \cup D]$.

These two items imply that for each x_i ($i \in [k]$), its neighbors $N(x_i) \cap D$ receive distinct colors which are also different from i. Let $\ell := |N(x_i) \cap D|$. It follows that we can let γ color the remaning $(k-1) - \ell$ neighbors of x_i in an arbitrary bijective manner with the $(k-1) - \ell$ colors that do not yet appear in the neighborhood of x_i.

After this process, x_i is a b-vertex for color i. We proceed in this way for all $i \in [k]$. Since for $i, j \in [k]$ with $i \neq j$ we have that $dist(x_i, x_j) \geq 4$, it follows that there are no edges between $N[x_i]$ and $N[x_j]$ in G. Hence, we did not introduce any coloring conflict in the previous step. Now, all vertices in G that have not yet received a color by γ have degree at most $k-1$, so we can extend γ to a proper coloring of G in a greedy fashion.

We summarize the whole procedure in Algorithm 2. We now analyze its runtime. Clearly,

[1] This case is almost identical to [16, Case II in the proof of Theorem 2].

Algorithm 2 An algorithm that either constructs a b-coloring of a graph G with $m(G)$ colors, or reports that there is none, and runs in FPT time parameterized by $\Delta(G)$.

```
Input  : A graph $G$ with $k = m(G)$ // More generally, graph $G$ with
         $\ell_k(G) \leq k$
Output : A $b$-coloring with $k$ colors of $G$ if it exists, and No otherwise.
Apply Reduction Rule 17 exhaustively;
Let $(D, T, R)$ be a partition of $V(G)$ such that for all $x \in D$, $\deg_G(x) \geq k$, for all
  $x \in T$, $\deg_G(x) = k - 1$, and $R = V(G) \setminus (D \cup T)$;
Let $B \subseteq D \cup T$ be a maximal set such that for distinct $b_1, b_2 \in B$, $dist(b_1, b_2) \geq 4$;
if $|B \cap T| < k$ then // Case 1
    Solve the instance in time $2^{\mathcal{O}(k^4 \cdot \Delta)}$ using the $b$-Coloring algorithm [16];
    if the algorithm of [16] returned a $b$-coloring $\gamma$ then return $\gamma$;
    else return No;
else // Case 2, i.e. $|B \cap T| \geq k$
    Pick a size-$k$ subset of $B \cap T$, say $B' := \{x_1, \ldots, x_k\}$;
    Initialize a $k$-coloring $\gamma \colon V(G) \to [k]$;
    For $i \in [k]$, let $\gamma(x_i) := i$;
    Let $\gamma$ color the vertices of $D$ injectively such that $\gamma$ remains proper on $G[B' \cup D]$;
    For $i \in [k]$, let $\gamma$ color $N(x_i) \cap D$ such that $x_i$ is the $b$-vertex of color $i$;
    Extend the coloring $\gamma$ greedily to the remainder of $G$;
    return $\gamma$;
```

exhaustively applying Reduction Rule 17 can be done in time $n^{\mathcal{O}(1)}$. As mentioned above, Case 1 can be solved in time $2^{\mathcal{O}(k^4 \cdot \Delta)}$. In Case 2, the coloring of $G[B' \cup D]$ can be found in time $\mathcal{O}(k^2)$, and extending the coloring to the remainder of G can be done in time $n^{\mathcal{O}(1)}$. The claimed bound follows. ◀

▶ Remark 18 (♣). Algorithm 2 solves the problem of deciding whether G admits a b-coloring with k colors in time $2^{\mathcal{O}(k^4 \cdot \Delta)} + n^{\mathcal{O}(1)}$ whenever $\ell_k(G) \leq k$.

Furthermore, in the proof of Theorem 2, we provide a polynomial kernel for the problem: In Case 1, we have a kernelized instance on $\mathcal{O}(k^4 \cdot \Delta)$ vertices (see Claim 2.2) and in Case 2, we always have a Yes-instance.

▶ **Corollary 19.** *The problem of deciding whether a graph G has a b-coloring with $k = m(G)$ colors admits a kernel on $\mathcal{O}(k^4 \cdot \Delta) = \mathcal{O}(\Delta^5)$ vertices.*

5.2 FPT Algorithm Parameterized by $\Delta(G) + \ell_k(G)$

The next parameterization of b-Coloring involving the maximum degree that we consider is by $\Delta(G) + \ell_k(G)$. We show that in this case, the problem is FPT. By Observation 6 we know that b-Coloring is NP-complete on graphs with $\ell_k(G) = 0$, and by Theorem 1, it is NP-complete even when $k = 3$ and $\Delta(G) = 4$. Hence, there is no FPT- nor XP-algorithm for a parameterization using only one of the two above mentioned parameters unless P = NP. Note that the algorithm we provide in this section can be used to solve the case of $k = m(G)$ for which we gave a separate algorithm in Section 5.1, see Algorithm 2. However, Algorithm 2 is much simpler than the algorithm presented in this section, and simply applying the following algorithm for the case $k = m(G)$ results in a runtime of $2^{\mathcal{O}(k^{k+3} \cdot \Delta)} + n^{\mathcal{O}(1)}$ which is far worse than the runtime of $2^{\mathcal{O}(k^4 \cdot \Delta)} + n^{\mathcal{O}(1)}$ of Theorem 2.

▶ **Theorem** (Thm. 3, restated). *There is an algorithm that given a graph G on n vertices decides whether G has a b-coloring with k colors in time $2^{\mathcal{O}(\ell \cdot \Delta \cdot \min\{\ell, \Delta\}^{\ell+2})} + n^{\mathcal{O}(1)}$, where $\Delta := \Delta(G)$ and $\ell := \ell_k(G)$.*

Proof. The overall strategy of the algorithm is similar to Algorithm 2. We can make the following assumptions. First, if $\ell \le k$, then we can apply Algorithm 2 directly to solve the instance at hand, see Remark 18. Hence we can assume that $k < \ell$. Furthermore, $k \le \Delta + 1$, otherwise we are dealing with a trivial No-instance; we have that $k \le \min\{\ell - 1, \Delta + 1\}$. Furthermore, we can assume that $k > 2$, otherwise the problem is trivially solvable in time polynomial in n.

We consider a partition (D, T, R) of $V(G)$, where the vertices in D have degree at least k, the vertices in T have degree $k - 1$ and the vertices in R have degree less than $k - 1$. We assume that Reduction Rule 17 has been applied exhaustively, so Observation 2.1 holds, i.e. every vertex in R has at least one neighbor in $D \cup T$.

Now, we pick an inclusion-wise maximal set $B \subseteq D \cup T$ such that for each pair of distinct vertices $b_1, b_2 \in B$, $dist_G(b_1, b_2) \ge \ell + 3$.

Case 1 ($|B \cap T| < k$). By the same argument given in Case 1 of the proof of Theorem 2, we have that any vertex in $T \cup R$ is at distance at most $\ell + 3$ from a vertex in B. We now give a bound on the number of vertices in G in terms of ℓ and Δ.

▶ **Claim 3.1** (♣). *If $|B \cap T| < k$, then $|V(G)| = \mathcal{O}(\ell \cdot \Delta \cdot \min\{\ell, \Delta\}^{\ell+2})$.*

By the previous claim, we can solve the instance in time $2^{\mathcal{O}(\ell \cdot \Delta \cdot \min\{\ell, \Delta\}^{\ell+2})}$ in this case, using the exact exponential time algorithm for b-Coloring due to Panolan et al. [16].

Case 2 ($|B \cap T| \ge k$). Let $B' \subseteq B \cap T$ be of size k and denote it by $B' := \{x_1, \ldots, x_k\}$. The strategy in this case is as follows: We compute a proper coloring of $G[D]$, and then modify it so that can be extended to a b-coloring of G. In this process we will be able to guarantee for each $i \in [k]$, that either x_i can be the b-vertex for color i, or we will have found another vertex in D that can serve as the b-vertex of color i. The difficulty here arises from the following situation: Suppose that in the coloring we computed for $G[D]$, a vertex x_i has two neighbors in D that received the same color. Then, x_i cannot be the b-vertex of color i in any extension of that coloring, since $\deg(x_i) = k - 1$, and $k - 1$ colors need to appear the neighborhood of x_i for it to be a b-vertex. However, recoloring a vertex in $N(x_i) \cap D$ might create a conflict in the coloring of $G[D]$. These potential conflicts can only appear in the connected component of $G[D \cup B']$ that contains x_i. We now show that each component of $G[D \cup B']$ can contain at most one such vertex, by our choice of the set B.

▶ **Claim 3.2** (♣). *Let C be a connected component of $G[D \cup B']$. Then, C contains at most one vertex from B'.*

Throughout the following, for $i \in [k]$, we denote by C_i the connected component of $G[D \cup B']$ that contains x_i, and by ℓ_i the number of vertices of C_i, i.e. $\ell_i := |V(C_i)|$. By Claim 3.2, $C_i \ne C_j$, for all $i, j \in [k], i \ne j$. We now show that each neighbor of x_i has no neighbor in $D \cap N[B']$ outside of $V(C_i) \cup N[x_i]$.

▶ **Claim 3.3** (♣). *Let $i \in [k]$, and $y \in N(x_i) \setminus D$. Then, $N_G[y] \cap (D \cup N[B']) \subseteq V(C_i) \cup N[x_i]$.*

Let $\mathcal{C}_\emptyset$ be the set of connected components of $G[D \cup B']$ that do not contain any vertex from B'. We observe that any proper coloring of $G[D \cup B']$ can be obtained from independently coloring the vertices in $C_1, \ldots, C_k$, and $\mathcal{C}_\emptyset$. If for some $i \in [k]$, C_i is a trivial[2] component,

[2] We call a connected component of a graph *trivial* if it contains only one vertex.

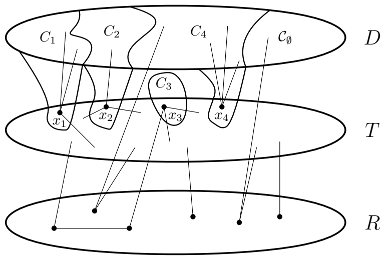


Figure 1 Illustration of the structure of a graph G in the proof of Theorem 3 where $k = 4$. Here, $B' = \{x_1, \dots, x_4\}$ and $C_1, \dots, C_4$ are the components of $G[D \cup B']$ containing $x_1, \dots, x_4$, respectively. Note that all vertices in T are of degree 3, all vertices in R of degree at most 2 and all vertices in R have a neighbor in $D \cup T$.

then $N(x_i) \cap D = \emptyset$. Hence, we can assign x_i any color without creating any conflict with the remaining vertices in $G[D \cup B']$. On top of that, Claim 3.3 ensures that assigning a color to a neighbor of any x_i (that is not contained in D) cannot create a coloring conflict with any vertex in $D \cup N[B']$ that is not contained in $V(C_i) \cup N[x_i]$. We illustrate the structure of G in Figure 1.

▶ **Claim 3.4 (♣).** *Let $i \in [k]$ and let $\gamma \colon V(C_i) \to [k]$ be a proper coloring of C_i. Then, one can find in time $\mathcal{O}(k^2 \cdot \ell_i^2)$ a set $Y_i \subseteq N_G(x_i) \setminus D$ and a proper coloring $\delta \colon V(C_i) \cup Y_i \to [k]$ of $G[V(C_i) \cup Y_i]$ that has a b-vertex for color i.*

We now wrap up the treatment of this case. We compute a proper k-coloring γ of $G[D \cup B']$ using standard methods [3]. We derive from γ another k-coloring δ of some induced subgraph of $G[D \cup N_G[B']]$ containing $D \cup B'$. For each $i \in [k]$, we do the following. With input $\gamma|_{V(C_i)}$ we compute a proper k-coloring δ_i of $G[V(C_i) \cup Y_i]$ using Claim 3.4, where Y_i is the set returned by its algorithm, and we let $\delta|_{V(C_i) \cup Y_i} := \delta_i$. Finally, we let $\delta|_{V(\mathcal{C}_\emptyset)} := \gamma|_{V(\mathcal{C}_\emptyset)}$. As for $i \neq j$, C_i and C_j are distinct connected components of $G[D \cup B']$ and by Claim 3.3, this construction is well-defined and there is no color conflict between any pair of vertices z_i, z_j where $z_i \in V(C_i) \cup Y_i$ and $z_j \in V(C_j) \cup Y_j$ for $i \neq j$. Since for each $i \in [k]$ we applied Claim 3.4, δ is a b-precoloring of G. All vertices in G that have not received a color so far (recall that δ colors all vertices in D) have degree at most $k - 1$, so we can extend the coloring δ greedily to the remainder of G and eventually obtain a b-coloring of G. The runtime of $2^{\mathcal{O}(\ell \cdot \Delta \cdot \min\{\ell, \Delta\}^{\ell+2})} + n^{\mathcal{O}(1)}$ is argued for in the full version [13]. ◀

Similar to above, we obtained a kernel for the problem. While this result does not provide a polynomial kernel for the parameterization $\Delta + \ell$, it does give a polynomial kernel if we consider the problem for *fixed* values of ℓ and parameter Δ.

▶ **Corollary 20.** *The problem of deciding whether a graph G admits a b-coloring with k colors admits a kernel on $\mathcal{O}(\ell \cdot \Delta \cdot \min\{\ell, \Delta\}^{\ell+2})$ vertices, where $\Delta := \Delta(G)$ and $\ell := \ell_k(G)$.*

6 Conclusion

We have presented a complexity dichotomy for b-Coloring with respect to two upper bounds on the b-chromatic number, in the following sense: We have shown that given a graph G and for fixed $k \in \{\Delta(G) + 1 - p, m(G) - p\}$, it can be decided in polynomial time whether G has a b-coloring with k colors whenever $p \in \{0, 1\}$ and the problem remains NP-complete whenever $p \geq 2$, already for $k = 3$.

The most immediate question left open in this work is the parameterized complexity of the b-Coloring problem when $k \in \{m(G), \Delta(G), m(G) - 1\}$. In all of these cases, we have provided XP-algorithms, and it would be interesting to see whether these problems are FPT or W[1]-hard. We showed that b-Coloring is FPT parameterized by $\Delta(G) + \ell_k(G)$, where $\ell_k(G)$ denotes the number of vertices of degree at least k in G, and this is optimal in the sense that there is no FPT nor XP algorithm for the problem parameterized by only one of the two invariants. It would be interesting to see if one could devise an FPT-algorithm for the parameterization that replaces the maximum degree by the number of colors.

References

1 Pierre Aboulker, Nick Brettell, Frédéric Havet, Dániel Marx, and Nicolas Trotignon. Coloring Graphs with Constraints on Connectivity. *J. Graph Theory*, 85(4):814–838, 2017.

2 Dominique Barth, Johanne Cohen, and Taoufik Faik. On the b-continuity property of graphs. *Discrete Appl. Math.*, 155:1761–1768, 2007.

3 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set Partitioning via Inclusion-Exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

4 Victor Campos, Carlos Vinicius G. C. Lima, and Ana Silva. Graphs of girth at least 7 have high b-chromatic number. *Eur. J. Combin.*, 48:154–164, 2015.

5 Miroslav Chelbík and Janka Chlebíkova. Hard Coloring Problems in Low Degree Planar Bipartite Graphs. *Discrete Appl. Math.*, 154:1960–1965, 2006.

6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 1st edition, 2015.

7 Konrad K. Dabrowski, François Dross, Matthew Johnson, and Daniël Paulusma. Filling the complexity gaps for colouring planar and bounded degree graphs. In *Proc. IWOCA '15*, volume 9538 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2015.

8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

9 Esther Galby, Paloma T. Lima, Daniël Paulusma, and Bernard Ries. On the Parameterized Complexity of k-Edge Colouring, 2019. arXiv:1901.01861.

10 Frédéric Havet, Cláudia Linhares Sales, and Leonardo Sampaio. b-Coloring of Tight Graphs. *Discrete Appl. Math.*, 160:2709–2715, 2012.

11 Frédéric Havet and Leonardo Sampaio. On the Grundy and b-chromatic numbers of a graph. *Algorithmica*, 65(4):885–899, 2013.

12 Robert W. Irving and David F. Manlove. The b-Chromatic Number of a Graph. *Discrete Appl. Math.*, 91(1-3):127–141, 1999.

13 Lars Jaffke and Paloma T. Lima. A Complexity Dichotomy for Critical Values of the b-Chromatic Number of Graphs. *CoRR*, 2018. arXiv:1811.03966.

14 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

15 Jan Kratochvíl, Zsolt Tuza, and Margit Voigt. On the b-Chromatic Number of Graphs. In *Proc. WG '02*, volume 2573 of *LNCS*, pages 310–320, 2002.

16 Fahad Panolan, Geevarghese Philip, and Saket Saurabh. On the parameterized complexity of b-Chromatic Number. *J. Comput. Syst. Sci.*, 84:120–131, 2017.

17 Leonardo Sampaio. *Algorithmic Aspects of Graph Coloring Heuristics*. PhD thesis, Université Nice Sophia Antipolis, France, 2012.

Parameterized Complexity of Conflict-Free Matchings and Paths

Akanksha Agrawal
Ben-Gurion University of the Negev, Beer-Sheva, Israel
agrawal@post.bgu.ac.il

Pallavi Jain
Institute of Mathematical Sciences, HBNI, Chennai, India
pallavij@imsc.res.in

Lawqueen Kanesh
Institute of Mathematical Sciences, HBNI, Chennai, India
lawqueen@imsc.res.in

Saket Saurabh
University of Bergen, Bergen, Norway
Institute of Mathematical Sciences, HBNI, Chennai, India
UMI ReLax
saket@imsc.res.in

Abstract

An input to a conflict-free variant of a classical problem Γ, called CONFLICT-FREE Γ, consists of an instance I of Γ coupled with a graph H, called the *conflict graph*. A solution to CONFLICT-FREE Γ in (I, H) is a solution to I in Γ, which is also an independent set in H. In this paper, we study conflict-free variants of MAXIMUM MATCHING and SHORTEST PATH, which we call CONFLICT-FREE MATCHING (CF-MATCHING) and CONFLICT-FREE SHORTEST PATH (CF-SP), respectively. We show that both CF-MATCHING and CF-SP are W[1]-hard, when parameterized by the solution size. Moreover, W[1]-hardness for CF-MATCHING holds even when the input graph where we want to find a matching is itself a matching, and W[1]-hardness for CF-SP holds for conflict graph being a unit-interval graph. Next, we study these problems with restriction on the conflict graphs. We give FPT algorithms for CF-MATCHING when the conflict graph is chordal. Also, we give FPT algorithms for both CF-MATCHING and CF-SP, when the conflict graph is d-degenerate. Finally, we design FPT algorithms for variants of CF-MATCHING and CF-SP, where the conflicting conditions are given by a (representable) matroid.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Conflict-free, Matching, Shortest Path, FPT algorithm, W[1]-hard, Matroid

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.35

Related Version `https://akanksha-agrawal.weebly.com/uploads/1/2/2/2/122276497/cf-matching-shortest_path.pdf`

Funding *Akanksha Agrawal*: PBC Program of Fellowships for Outstanding Post-doctoral Researchers from China and India, reference no. 5101479000.
Pallavi Jain: SERB-NPDF fellowship, reference no. PDF/2016/003508, DST, India.
Saket Saurabh: Horizon 2020 Framework program, the European Research Council (ERC) Consolidator Grant LOPPRE reference no. 819416.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 35; pp. 35:1–35:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the recent years, conflict-free variant of classical combinatorial optimization problems have gained attention from the viewpoint of algorithmic complexity. A typical input to a conflict-free variant of a classical problem Γ, which we call Conflict-Free Γ, consists of an instance I of Γ coupled with a graph H, called the *conflict graph*. A solution to Conflict-Free Γ in (I, H) is a solution to I in Γ, which is also an independent set in H. Notice that conflict-free version of the problem introduces the constraint of "impossible pairs" in the solution that we seek for. Such a constraint of "impossible pairs" in a solution arises, for example, in the context of program testing and validation [16, 23]. Gabow et al. [16] studied the conflict-free version of paths in a graph, which they showed to be NP-complete.

Conflict-free variants of several classical problems such as, Bin Packing [10, 18, 20], Knapsack [35, 32], Minimum Spanning Tree [5, 6], Maximum Matching [6], Maximum Flow [33, 34], Shortest Path [6] and Set Cover [11] have been studied in the literature from the viewpoint of algorithmic complexity, approximation algorithms, and heuristics. It is interesting to note that most of these problems are NP-hard even when their classical counterparts are polynomial time solvable. Recently, Jain et al. [19] and Agrawal et al. [2, 1] initiated the study of conflict-free problems in the realm of parameterized complexity. In particular, they studied Conflict-Free $\mathcal{F}$-Deletion problems for various families $\mathcal{F}$, of graphs such as the family of forests, independent sets, bipartite graphs, interval graphs, etc.

Maximum Matching and Shortest Path are among the classical graph problems which are of very high theoretical and practical interest. The Maximum Matching problem takes as input a graph G, and the objective is to compute a maximum sized subset $Y \subseteq E(G)$ such that no two edges in Y have a common vertex. Maximum Matching is known to be solvable in polynomial time [12, 28]. The Shortest Path problem takes as input a graph G and vertices s and t, and the objective is to compute a path between s and t in G with the minimum number of vertices. The Shortest Path problem, together with its variants such as all-pair shortest path, single-source shortest path, weighted shortest path, etc. are known to be solvable in polynomial time [7, 3].

Darmann et al. [6] (among other problems) studied the conflict-free variants of Maximum Matching and Shortest Path. They showed that the conflict-free variant of Maximum Matching is NP-hard even when the conflict graph is a disjoint union of edges (matching). Moreover, for the conflict-free variant of Shortest Path, they showed that the problem is APX-hard, even when the conflict graph belongs to the family of 2-ladders.

In this paper, we study the conflict-free versions of matching and shortest path from the viewpoint of parameterized complexity. A parameterized problem Π is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet. An instance of a parameterized problem is a pair (I, k), where I is a classical problem instance and k is an integer, which is called the *parameter*. One of the central notions in parameterized complexity is *fixed-parameter tractability*, where given an instance (I, k) of a parameterized problem Π, the goal is to design an algorithm that runs in time $f(k)n^{\mathcal{O}(1)}$, where, $n = |I|$ and $f(\cdot)$ is some computable function, whose value depends only on k. An algorithm with running time as described above, is called an FPT algorithm. A parameterized problem that admits an FPT algorithm is said to be in FPT. Not every parameterized problem admits an FPT algorithm, under reasonable complexity-theoretic assumptions. Similar to the notion of NP-hardness and NP-hard reductions in classical Complexity Theory, there are notions of W[t]-hardness, where $t \in \mathbb{N}$ and parameterized reductions in parameterized complexity. A parameterized problem which is W[t]-hard, for some $t \in \mathbb{N}$ is believed not to admit an FPT algorithm. For more details on parameterized complexity we refer to the books of Downey and Fellows [9], Flum and Grohe [13], Niedermeier [30], and Cygan et al. [4].

Our Results. We study conflict-free (parameterized) variants of Maximum Matching and Shortest Path, which we call Conflict Free Maximum Matching (CF-MM, for short) and Conflict Free Shortest Path (CF-SP, for short), respectively. These problems are formally defined below.

Conflict Free Maximum Matching (CF-MM) **Parameter:** k
Input: A graph $G = (V, E)$, a conflict graph $H = (E, E')$, and an integer k.
Question: Is there a matching M of size at least k in G, such that M is an independent set in H?

Conflict Free Shortest Path (CF-SP) **Parameter:** k
Input: A graph $G = (V, E)$, a conflict graph $H = (E, E')$, two special vertices s and t, and an integer k.
Question: Is there an st-path P of length at most k in G, such that $E(P)$ is an independent set in H?

We show that both CF-MM and CF-SP are W[1]-hard, when parameterized by the solution size. The W[1]-hardness for CF-MM is obtained by giving an appropriate reduction from Independent Set, which is known to be W[1]-hard, when parameterized by the solution size [4, 8]. In fact, our W[1]-hardness result for CF-MM holds even when the graph where we want to compute a matching is itself a matching. We show the W[1]-hardness of CF-SP by giving an appropriate reduction from a multicolored variant of the problem Unit 2-Track Independent Set (which we prove to be W[1]-hard). We note that Unit 2-Track Independent Set is known to be W[1]-hard, which is used to establish W[1]-hardness of its multicolored variant. We note that our W[1]-hardness result of CF-SP holds even when the conflict graph is a unit interval graph.

Having shown the W[1]-hardness results, we then restrict our attention to having conflict graphs belonging to some families of graphs, where the Independent Set problem is either polynomial time solvable or solvable in FPT time. Two of the very well-known graph families that we consider are the family of chordal graphs and the family of d-degenerate graphs. For the CF-MM problem, we give an FPT algorithm, when the conflict graph belongs to the family of chordal graphs. Our algorithm is based on a dynamic programming over a "structured" tree decomposition of the conflict graph (which is chordal) together with "efficient" computation of representative families at each step of our dynamic programming routine. Notice that we cannot obtain an FPT algorithm for the CF-SP problem when the conflict graph is a chordal graph. This holds because unit-interval graphs are chordal, and the problem CF-SP is W[1]-hard, even when the conflict graph is a unit-interval graph.

For conflict graphs being d-degenerate, we obtain FPT algorithms for both CF-MM and CF-SP. These algorithms are based on the computation of an independence covering family, a notion which was recently introduced by Lokshtanov et al. [25]. We note that even for nowhere dense graphs, such an independence covering family can be computed efficiently [25]. Since our algorithms are based on computation of independence covering families, hence, our results hold even when the conflict graph is a nowhere dense graph.

Finally, we study a variant of CF-MM and CF-SP, where instead of conflicting conditions being imposed by independent sets in a conflict graph, they are imposed by independence constraints in a (representable) matroid. We give FPT algorithms for the above variant of both CF-MM and CF-SP.

Due to space limitations, many proofs have been omitted from the extended abstract.

2 Preliminaries

Sets and graph notations. For $n \in \mathbb{N}$, by $[n]$ and $[0,n]$, we denote the sets $\{1,2,\cdots,n\}$ and $\{0,1,2,\cdots,n\}$, respectively. For a set U and $p \in \mathbb{N}$, a p-family (over U) is a family of subsets of U of size p. We let ω denote the exponent in the running time of algorithm for matrix multiplication, the current best known bound for it is $\omega < 2.373$ [36]. Consider a graph G. For $X \subseteq V(G)$, $G[X]$ denotes the subgraph of G with vertex set X and edge set $\{uv \in E(G) \mid u,v \in X\}$. For $Y \subseteq E(G)$, $G[Y]$ denotes the subgraph of G with vertex set $\cup_{uv\in Y}\{u,v\}$ and edge set Y.

We define a structured tree decomposition that will be used in our algorithm.

▶ **Definition 1** ([4, 22]). Let (T,X) be a tree decomposition of a graph H with r as the root node. That is, T is a tree rooted at r and $X = \{X_t \mid t \in V(T)\}$. Then, (T,X) is a *nice tree decomposition* if for each each leaf ℓ in T and the root r, we have that $X_\ell = X_r = \emptyset$, and each non-leaf node $t \in V(T)$ is of one of the following types: **Introduce node:** t has exactly one child, say t', and $X_t = X_{t'} \cup \{v\}$, where $v \notin X_{t'}$. We say that v is *introduced* at t; **Forget node:** t has exactly one child, say t', and $X_t = X_{t'} \setminus \{v\}$, where $v \in X_{t'}$. We say that v is *forgotten* at t; **Join node:** t has exactly two children, say t_1 and t_2, and $X_t = X_{t_1} = X_{t_2}$.

A tree decomposition (T,X) of a graph H, where for each $t \in V(T)$, the graph $H[X_t]$ is a clique, is called a *clique-tree*. The result below follows from existence of clique-tree for chordal graphs [17] and an algorithm for computation of a nice tree decomposition [4, 22].

▶ **Proposition 2.** *Given an n vertex chordal graph H, in polynomial time we can construct a nice tree decomposition which is also a clique-tree (nice clique-tree), (T,X) of H with $\mathcal{O}(n^2)$ nodes.*

Matroids and representative sets. In the following we state some definitions related to matroids used in the paper. We refer the reader to [31] for more details. We also state the definition of representative families and state some results related to them.

▶ **Definition 3** ([4, 31]). A matroid $\mathcal{M} = (U,\mathcal{I})$ is a *partition* matroid if the ground set U is partitioned into sets $U_1, U_2, \cdots, U_k$, and for each $i \in [k]$, there is an integer a_i associated with U_i. A set $S \subseteq U$ is an independent in $\mathcal{M}$ if and only if for each $i \in [k]$, $|S \cap U_i| \le a_i$.

▶ **Proposition 4** ([15, 31, 26]). *A representation of a partition matroid over $\mathbb{Q}$ (the field of rationals) can be computed in polynomial time.*

▶ **Definition 5.** Let $\mathcal{M}_1 = (U_1,\mathcal{I}_1), \mathcal{M}_2 = (U_2,\mathcal{I}_2) \cdots, \mathcal{M}_t = (U_t,\mathcal{I}_t)$ be t matroids with $U_i \cap U_j = \emptyset$, for all $1 \le i \ne j \le t$. The *direct sum* $\mathcal{M}_1 \oplus \cdots \oplus \mathcal{M}_t$, of $\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_t$ is the matroid with ground set $U = \cup_{i\in[t]} U_i$ and $X \subseteq U$ is independent in $\mathcal{M}$ if and only if for each $i \in [t]$, $X \cap U_i \in \mathcal{I}_i$.

▶ **Proposition 6** ([27, 31]). *Given matrices $A_1, A_2, \cdots, A_t$ (over $\mathbb{F}$) representing matroids $\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_t$, respectively, we can compute a representation of their direct sum, $\mathcal{M}_1 \oplus \cdots \oplus \mathcal{M}_t$, in polynomial time.*

Next, we state the definition of representative families.

▶ **Definition 7.** Let $\mathcal{M} = (U,\mathcal{I})$ be a matroid, and $\mathcal{A}$ be a p-family of U. We say that $\mathcal{A}' \subseteq \mathcal{A}$ is a q-representative for $\mathcal{A}$ if for every set $Y \subseteq U$ of size q, if there is a set $X \in \mathcal{A}$, such that $X \cap Y = \emptyset$ and $X \cup Y \in \mathcal{I}$, then there is a set $X' \in \mathcal{A}'$ such that $X' \cap Y = \emptyset$ and $X' \cup Y \in \mathcal{I}$. If $\mathcal{A}' \subseteq \mathcal{A}$ is a q-representative for $\mathcal{A}$ then we denote it by $\mathcal{A}' \subseteq^q_{rep} \mathcal{A}$.

▶ **Theorem 8** ([4, 14]). *Given a matrix M (over field $\mathbb{F}$) representing a matroid $\mathcal{M} = (U, \mathcal{I})$ of rank k, a p-family $\mathcal{A}$ of independent sets in $\mathcal{M}$, and an integer q such that $p+q=k$, there is an algorithm which computes a q-representative family $\mathcal{A}' \subseteq^q_{rep} \mathcal{A}$ of size at most $\binom{p+q}{p}$ using at most $\mathcal{O}\big(|\mathcal{A}|\big(\binom{p+q}{p}p^\omega + \binom{p+q}{p}^{\omega-1}\big)\big)$ operations over $\mathbb{F}$.*

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two families of sets over U and $\mathcal{M} = (U, \mathcal{I})$ be a matroid. We define their convolution as: $\mathcal{A}_1 \star \mathcal{A}_2 = \{A_1 \cup A_2 \mid A_1 \in \mathcal{A}_1, A_2 \in \mathcal{A}_2, A_1 \cap A_2 = \emptyset \text{ and } A_1 \cup A_2 \in \mathcal{I}\}$.

Universal sets and their computation. An (n, k)-*universal set* is a family $\mathcal{F}$ of subsets of $[n]$ such that for any set $S \subseteq [n]$ of size k, the family $\{A \cap S \mid A \in \mathcal{F}\}$ contains all 2^k subsets of S.

▶ **Proposition 9** ([4, 29]). *For any $n, k \geq 1$, we can compute an (n, k)-universal set of size $2^k k^{\mathcal{O}(\log k)} \log n$ in time $2^k k^{\mathcal{O}(\log k)} n \log n$.*

3 W[1]-hardness Results

In this section, we show that Conflict Free Maximum Matching and Conflict Free Shortest Path are W[1]-hard, when parameterized by the solution size.

W[1]-hardness of CF-MM. We show that CF-MM is W[1]-hard, when parameterized by the solution size, even when the graph where we want to find a matching, is itself a matching (disjoint union of edges). To prove our result, we give an appropriate reduction from Independent Set to CF-MM. It is known that Independent Set is W[1]-hard, when parameterized by the size of an independent set [4, 8]. Given an instance $(G^\star, k)$ of Independent Set, we construct an equivalent instance (G, H, k) of CF-MM as follows. We first describe the construction of G. For each $v \in V(G^\star)$, we add an edge vv' to G. Notice that G is a matching. This completes the description of G. Next, we move to the construction of H. We have $V(H) = \{e_v = vv' \mid v \in V(G^\star)\}$. Moreover, for $e_u, e_v \in V(H)$, we add the edge $e_u e_v$ to $E(H)$ if and only if $uv \in E(G^\star)$. We note that H is exactly the same as $G^\star$, with vertices being renamed. This completes the description of the reduction. We obtain the following theorem from the equivalence of instances $(G^\star, k)$ of Independent Set and (G, H, k) of CF-MM.

▶ **Theorem 10.** CF-MM *is W[1]-hard, when parameterized by the solution size.*

W[1]-hardness of CF-SP. We show that CF-SP is W[1]-hard, when parameterized by the solution size, even when the conflict graph is a proper interval graph. We refer to this restricted variant of the problem as Unit Interval CF-SP. To prove our result, we give an appropriate reduction from a multicolored variant of the problem Unit 2-Track Independent Set, which we call Unit 2-Track Multicolored IS. In the following, we define the problem Unit 2-Track Multicolored IS.

Unit 2-Track Multicolored IS (Unit 2-Track MIS) **Parameter:** k
Input: Two unit-interval graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, and a partition $V_1, V_2, \cdots, V_k$ of V.
Question: Is there a set $S \subseteq V$, such that S is an independent set in both G_1 and G_2, and for each $i \in [k]$, we have $|S \cap V_i| = 1$?

It is known that Unit 2-Track IS is W[1]-hard, when parameterized by the solution size [21]. We can show that the problem Unit 2-Track MIS is W[1]-hard, when parameterized by the number of sets in the partition by giving an appropriate (Turing) reduction from Unit 2-Track IS. We give a reduction from Unit 2-Track MIS to Unit Interval CF-SP, and hence obtaining the desired result.

We now give a parameterized reduction from Unit 2-Track MIS to Unit Interval CF-SP. Let $(G_1, G_2, V_1, \cdots, V_k)$ be an instance of Unit 2-Track MIS. We construct an instance (G', H, s, t, k') of Unit Interval CF-SP as follows. For each $v \in V(G_1)$, we add a path on 3 vertices namely, (v_1, v_2, v_3) in G'. For notational convenience, for $v \in V(G_1)$, by $e_{12}(v)$ and $e_{23}(v)$ we denote the edges v_1v_2 and v_2v_3, respectively. Consider $i \in [k-1]$. For $u \in V_i$ and $v \in V_{i+1}$, we add the edge $z_{uv} = u_3v_1$ to $E(G')$. Moreover, by Z_i, we denote the set $\{z_{uv} \mid u \in V_i, v \in V_{i+1}\}$. We add two new vertices s and t to $V(G')$, and add all the edges in $Z_0 = \{sv_1 \mid v \in V_1\}$ and $Z_k = \{v_3t \mid v \in V_k\}$ to $E(G')$. Next, we move to the construction of H. Note that H must be a unit-interval graph on the vertex set $E(G') = (\cup_{i\in[0,k]} Z_i) \cup (\cup_{v \in V(G_1)}\{e_{12}(v), e_{23}(v)\})$. In H, each vertex in $\cup_{i\in[0,k]} Z_i$ is an isolated vertex. Let $E_{12} = \{e_{12}(v) \mid v \in V(G_1)\}$ and $E_{23} = \{e_{23}(v) \mid v \in V(G_1)\}$. For $e_{12}(u), e_{12}(v) \in E_{12}$, we add the edge $e_{12}(u)e_{12}(v)$ to $E(H)$ if and only if $uv \in E(G_1)$. Similarly, for $e_{23}(u), e_{23}(v) \in E_{23}$, we add the edge $e_{23}(u)e_{23}(v)$ to $E(H)$ if and only if $uv \in E(G_2)$. Observe that $H[E_{12}]$ is isomorphic to G_1, with bijection $\phi_1 : V(G_1) \to E_{12}$ with $\phi_1(v) = e_{12}(v)$. Similarly, $H[E_{23}]$ is isomorphic to G_2 with bijection $\phi_2 : V(G_2) \to E_{23}$ with $\phi_2(v) = e_{23}(v)$. By construction, H is a disjoint union of unit-interval graphs, and hence is a unit-interval graph. Finally, we set $k' = 3k+1$. This completes the description of the reduction. We obtain the following theorem from the equivalence of instances $(G_1, G_2, V_1, \cdots, V_k)$ of Unit 2-Track MIS and (G', H, s, t, k') of Unit Interval CF-SP.

▶ **Theorem 11.** Unit Interval CF-SP *is* W[1]-hard*, when parameterized by the solution size.*

4 FPT Algorithm for CF-MM with Chordal Conflict

In this section, we show that CF-MM is FPT, when the conflict graph is a chordal graph. We call this restricted version of CF-MM as Chordal Conflict Matching. Towards designing an algorithm for Chordal Conflict Matching, we first give an FPT algorithm for a restricted version of Chordal Conflict Matching, where the graph where we want to compute a matching is a bipartite graph. We call this variant of Chordal Conflict Matching as Chordal Conflict Bipartite Matching (CCBM). We then employ the algorithm for CCBM to design an FPT algorithm for Chordal Conflict Matching.

4.1 FPT algorithm for CCBM

We design an FPT algorithm for the problem CCBM, where the conflict graph is chordal and the graph where we want to compute a matching is a bipartite graph. The problem CCBM is formally defined below.

Chordal Conflict Bipartite Matching (CCBM) **Parameter:** k
Input: A bipartite graph $G = (V, E)$ with vertex bipartition L, R, a conflict graph $H = (E, E')$, and an integer k.
Question: Is there a matching $M \subseteq E$ of size k in G, such that M is an independent set in H?

The FPT algorithm for CCBM is based on a dynamic programming routine over tree decomposition of the conflict graph H and representative sets on the graph G. Let (G, L, R, H, k) be an instance of CF-MM, where G is a bipartite graph on n vertices, with vertex bipartition L, R, and H is a chordal graph with $V(H) = E(G)$.

In the following, we construct three matroids $\mathcal{M}_L = (E, \mathcal{I}_L), \mathcal{M}_R = (E^c, \mathcal{I}_R)$, and $\mathcal{M} = (E \cup E^c, \mathcal{I})$. Matroids $\mathcal{M}_L$ and $\mathcal{M}_R$ are partition matroids and the matroid $\mathcal{M}$ is the direct sum of $\mathcal{M}_L$ and $\mathcal{M}_R$. The ground set of $\mathcal{M}_L$ is $E = E(G)$. The set E^c contains a copy of edges in E, i.e., $E^c = \{e^c \mid e \in E\}$. We create two (disjoint) sets E and E^c, because $\mathcal{M}$ is the direct sum of $\mathcal{M}_L$ and $\mathcal{M}_R$, and we want their ground sets to be disjoint. Next, we describe the partition $\mathcal{E}$ of E into $|L|$ sets and $|L|$ integers, one for each set in the partition, for the partition matroid $\mathcal{M}_L$. For $u \in L$, let $E_u = \{uv \mid uv \in E\}$. Notice that for $u, v \in L$, where $u \neq v$, we have $E_u \cap E_v = \emptyset$. Moreover, $\cup_{u \in E} E_u = E$. We let $\mathcal{E} = \{E_u \mid u \in L\}$, and for each $u \in L$, we set $a_u = 1$. Similarly, we define the partition $\mathcal{E}^c$ of E^c with respect to set R. That is, we let $\mathcal{E}^c = \{E^c_u = \{(uv)^c \mid uv \in E(G)\} \mid u \in R\}$. Furthermore, for $u \in R$, we let $a_{u^c} = 1$. We define the following notation, which will be used later. For $Z \subseteq E$, we let $Z^c = \{e^c \mid e \in Z\} \subseteq E^c$.

To capture the independence property on the conflict graph, we rely on the fact that a chordal graph admits a nice clique-tree (Proposition 2). This allows us to do dynamic programming over a nice clique-tree. At each step of our dynamic programming routine, using representative sets, we ensure that we store a family of sets which are enough to recover (some) independent set in $\mathcal{M}$, if a solution exists.

We now move to the formal description of the algorithm. The algorithm starts by computing a nice clique-tree (T, X) of H in polynomial time, using Proposition 2. Let $r \in V(T)$ be the root of the (rooted) tree T. For $X_t \in X$, we let $\mathcal{X}_t = \{\emptyset\} \cup \{\{v\} \mid v \in X_t\}$.For a node $t \in V(T)$, by $\mathsf{desc}(t)$ we denote the set descendant of t in T (including t). For $t \in V(T)$, H_t is the graph $H[V_t]$, where $V_t = \cup_{d \in \mathsf{desc}(t)} X_d$.

In the following, we state some notations, which will be used in the algorithm. For each $t \in V(T)$, $Y \in \mathcal{X}_t$, and an integer $p \in [0, k]$ we define a family $\mathcal{P}^p_{t,Y}$ as: $\mathcal{P}^p_{t,Y} = \{Z \cup Z^c \mid Z \subseteq V(H_t)(\subseteq E), |Z| = p, Z \cap X_t = Y, Z \cup Z_c \in \mathcal{I}$ and $H_t[Z]$ is edgeless$\}$. For a family $\mathcal{F}$ of subsets of $E \cup E^c$, $\mathcal{F}$ is called a *paired-family* if for each $F \in \mathcal{F}$, there is $Z \subseteq E$, such that $F = Z \cup Z^c$. Next, we state the entries in our dynamic programming routine.

▶ **Definition 12.** *For each $t \in V(T)$, $Y \in \mathcal{X}_t$ and $p \in [0, k]$, we have an entry $c[t, Y, p]$, which stores a paired-family $\mathcal{F}(t, Y, p)$ of subsets of $E \cup E^c$ of size $2p$, such that for each $F = Z \cup Z^c \in \mathcal{F}$, the following conditions are satisfied: $|Z| = p$; $Z \cap X_t = Y$; Z is a matching in G, i.e., Z and Z^c are independent sets in $\mathcal{M}_L$ and $\mathcal{M}_R$, respectively; Z is an independent set in H_t. Moreover, $\mathcal{F} \neq \emptyset$ if and only if $\mathcal{P}^p_{t,Y} \neq \emptyset$.*

Consider $t \in V(T)$, $Y \in \mathcal{X}_t$ and $p \in [0, k]$. Observe that $\mathcal{P}^p_{t,Y}$ is a valid candidate for $c[t, Y, p]$, which also implies that (G, H, k) is a yes instance of CCBM if and only if $c[r, \emptyset, k] \neq \emptyset$. However, we cannot set $c[t, Y, p] = \mathcal{P}^p_{t,Y}$ as the size of $\mathcal{P}^p_{t,Y}$ could be exponential in n, and the goal here is to obtain an FPT algorithm. Hence, we will store a much smaller subfamily (of size at most $\binom{2k}{2p}$) of $\mathcal{P}^p_{t,Y}$ in $c[t, Y, p]$, which will be computed using representative sets. Moreover, as we have a structured form of a tree decomposition (nice clique-tree) of H, we compute the entries of the table based on the entries of its children, which will be given by recursive formulae. For leaf nodes, which form base cases for recursive formulae, we compute all entries directly.

Next, we give (recursive) formulae for the computation of the table entries. Consider $t \in V(T)$, $Y \in \mathcal{X}_t$ and $p \in [0, k]$. We compute the entry $c[t, Y, k]$ based on the following cases.

Leaf node. t is a leaf node. In this case, we have $X_t = \emptyset$, and hence $\mathcal{X}_t = \{\emptyset\}$. If $p = 0$, then $\mathcal{P}^p_{t,\emptyset} = \{\emptyset\}$, and $\mathcal{P}^p_{t,\emptyset} = \emptyset$, otherwise. Since, $\mathcal{P}^p_{t,\emptyset}$ is a valid candidate for $c[t, Y, p]$, we set $c[t, Y, p] = \mathcal{P}^p_{t,\emptyset}$. Note that $c[t, Y, p]$ has size at most $1 \leq \binom{2k}{2p}$, and we can compute $c[t, Y, p]$ in polynomial time.

Introduce node. Suppose t is an introduce node with child t' such that $X_t = X_{t'} \cup \{e\}$, where $e \notin X_{t'}$. If $Y \neq \emptyset$ and $p < 1$, then we set $c[t, Y, p] = \emptyset$. Otherwise, we compute the entry as described below. Before computing the entry $c[t, Y, p]$, we first compute a set $\widetilde{\mathcal{P}}^p_{t,Y}$ as follows. If $Y \neq \{e\}$, then we set $\widetilde{\mathcal{P}}^p_{t,Y} = c[t', Y, p]$, and otherwise, $\widetilde{\mathcal{P}}^p_{t,Y} = c[t', \emptyset, p-1] \star \{\{e, e^c\}\}$.

Next, we compute $\widehat{\mathcal{P}}^p_{t,Y} \subseteq^{2k-2p}_{\mathsf{rep}} \widetilde{\mathcal{P}}^p_{t,Y}$ of size $\binom{2k}{2p}$, using Theorem 8. Finally, we set $c[t, Y, p] = \widehat{\mathcal{P}}^p_{t,Y}$.

Forget node. Suppose t is a forget node with child t' such that $X_t = X_{t'} \setminus \{e\}$, where $e \in X_{t'}$. Before computing the entry $c[t, Y, p]$, we first compute a set $\widetilde{\mathcal{P}}^p_{t,Y}$ as follows. If $Y \neq \emptyset$, we set $\widetilde{\mathcal{P}}^p_{t,Y} = c[t', Y, p]$, and otherwise, $\widetilde{\mathcal{P}}^p_{t,Y} = c[t', \emptyset, p]$.

Next, we compute $\widehat{\mathcal{P}}^p_{t,Y} \subseteq^{2k-2p}_{\mathsf{rep}} \widetilde{\mathcal{P}}^p_{t,Y}$ of size $\binom{2k}{2p}$, using Theorem 8. Finally, we set $c[t, Y, p] = \widehat{\mathcal{P}}^p_{t,Y}$.

Join node. Suppose t is a join node with children t_1 and t_2, such that $X_t = X_{t_1} = X_{t_2}$. If $Y \neq \emptyset$ and $p < 1$, then we set $c[t, Y, p] = \emptyset$. Otherwise, we compute the entry as described below. Before computing the entry $c[t, Y, p]$, we first compute a set $\widetilde{\mathcal{P}}^p_{t,Y}$ as follows. If $Y = \emptyset$, we set $\tilde{\mathcal{P}}^p_{t,Y} = \cup_{i \in [0,p]}(c[t_1, \emptyset, i] \star c[t_2, \emptyset, p-i])$, and otherwise, $\tilde{\mathcal{P}}^p_{t,Y} = \cup_{i \in [p]}(c[t_1, Y, i] \star c[t_2, \emptyset, p-i])$.

Next, we compute $\widehat{\mathcal{P}}^p_{t,Y} \subseteq^{2k-2p}_{\mathsf{rep}} \widetilde{\mathcal{P}}^p_{t,Y}$ of size $\binom{2k}{2p}$, using Theorem 8. Finally, we set $c[t, Y, p] = \widehat{\mathcal{P}}^p_{t,Y}$.

This completes the description of the (recursive) formulae for computing all entries of the table. The correctness of the algorithm follows from the correctness of the (recursive) formulae, and the fact that (G, H, k) is a yes instance of CCBM if and only if $c[r, \emptyset, k] \neq \emptyset$. From the above, we obtain the following theorem.

▶ **Theorem 13.** CCBM *admits an FPT algorithm running in time* $\mathcal{O}(2^{\mathcal{O}(\omega k)} n^{\mathcal{O}(1)})$.

4.2 FPT algorithm for Chordal Conflict Matching

We design an FPT algorithm for Chordal Conflict Matching, using the algorithm for CCBM (Theorem 13). Let (G, H, k) be an instance of CF-MM, where H is a chordal graph and G is a graph on n vertices. We assume that G is a graph on vertex set $[n]$, which can easily be achieved by renaming vertices.

The algorithm starts by computing an $(n, 2k)$-universal set $\mathcal{F}$, using Proposition 9. For each set $A \in \mathcal{F}$, the algorithm constructs an instance $I_A = (G_A, L_A, R_A, H_A, k)$ of CCBM as follows. We have $V(G_A) = V(G)$, $L_A = A$, $R = V(G) \setminus A$, $E(G_A) = \{uv \in E(G) \mid u \in L_A, v \in R_A\}$, and $H_A = H[E(G_A)]$. Note that H_A is a chordal graph because chordal graphs are closed under induced subgraphs and disjoint unions. The algorithm decides the instance I_A using Theorem 13, for each $A \in \mathcal{F}$. The algorithm outputs yes if and only if there is $A \in \mathcal{F}$, such that I_A is a yes instance of CCBM.

▶ **Theorem 14.** *The algorithm presented for* CF-MM *is correct, Moreover, it runs in time* $2^{\mathcal{O}(\omega k)}k^{\mathcal{O}(\log k)}n^{\mathcal{O}(1)}$*, where* $\omega < 2.373$ *is the exponent of matrix multiplication and* n *is the number of vertices in the input graph.*

Proof. Let (G, H, k) be an instance of CF-MM, where H is a chordal graph and G is a graph on vertex set $[n]$. Clearly, if the algorithm outputs yes, then indeed (G, H, k) is a yes instance of CF-MM. Next, we argue that if (G, H, k) is a yes instance of CF-MM then the algorithm returns yes. Suppose there is a solution $M \subseteq E(G)$ to CF-MM in (G, H, k). Let $S = \{i, j \mid ij \in M\}$, and $L = \{i \mid \text{there is } j \in [n] \text{ such that } ij \in M \text{ and } i < j\}$. Observe that $|S| = 2k$. Since $\mathcal{F}$ is an $(n, 2k)$-universal set, there is $A \in \mathcal{F}$ such that $A \cap S = L$. Note that S is a solution to CCBM in I_A. This together with Theorem 13 implies that the algorithm will return yes as output.

Next, we prove the claimed running time of the algorithm. The algorithm computes $(n, 2k)$-universal set of size $\mathcal{O}(2^{2k}k^{\mathcal{O}(\log k)} \log n)$, in time $\mathcal{O}(2^{2k}k^{\mathcal{O}(\log k)}\ n \log n)$, using Proposition 9. Then for each $A \in \mathcal{F}$, the algorithm creates an instance I_A of CCBM in polynomial time. Furthermore, it resolves the I_A of CCBM in time $\mathcal{O}(2^{\mathcal{O}(\omega k)}n^{\mathcal{O}(1)})$ using Theorem 13. Hence, the running time of the algorithm is bounded by $2^{\mathcal{O}(\omega k)}k^{\mathcal{O}(\log k)}n^{\mathcal{O}(1)}$. ◀

5 FPT algorithms for CF-MM and CF-SP with matroid constraints

In this section, we study the problems CF-MM and CF-SP, where the conflicting condition is being an independent set in a (representable) matroid. Due to technical reasons we only consider the case when the matroid is representable over $\mathbb{Q}$ (the field of rationals).

5.1 FPT algorithm for Matroid CF-MM

We study a variant of the problem CF-MM, where the conflicting condition is being an independent set in a matroid representable over $\mathbb{Q}$. We call this variant of CF-MM as Rational Matroid CF-MM (Rat Mat CF-MM, for short), which is formally defined below.

Rational Matroid CF-MM (Rat Mat CF-MM)	**Parameter:** k
Input: A graph G, a matrix $A_{\mathcal{M}}$ (representing a matroid $\mathcal{M}$ over $\mathbb{Q}$) with columns indexed by $E(G)$, and an integer k.	
Question: Is there a matching $M \subseteq E(G)$ of size at most k, such that the set of columns in M are linearly independent (over $\mathbb{Q}$)?	

We design an FPT algorithm for Rat Mat CF-MM. Towards designing an algorithm for Rat Mat CF-MM, we first give an FPT algorithm for a restricted version of Rat Mat CF-MM, where the graph in which we want to compute a matching is a bipartite graph. We call this variant of Rat Mat CF-MM as Rat Mat CF-Bipartite Matching (Rat Mat CF-BM). We then employ the algorithm for Rat Mat CF-BM to design an FPT algorithm for Rat Mat CF-MM.

5.1.1 FPT algorithm for Rat Mat CF-BM

We design an FPT algorithm for the problem Rat Mat CF-BM, where the conflicting condition is being an independent set in a matroid (representable over $\mathbb{Q}$) and the graph where we want to compute a matching is a bipartite graph.

Our algorithm takes an instance of Rat Mat CF-BM and generates an instance of 3-Matroid Intersection, and then employs the known algorithm for 3-Matroid

Intersection to resolve the instance. In the following, we formally define the problem 3-Matroid Intersection.

> 3-Matroid Intersection **Parameter:** k
> **Input:** Matrices $A_{\mathcal{M}_1}, A_{\mathcal{M}_2}$, and $A_{\mathcal{M}_3}$ over $\mathbb{F}$ (representing matroids $\mathcal{M}_1, \mathcal{M}_2$, and $\mathcal{M}_3$, respectively, on the same ground set E) with columns indexed by E, and an integer k.
> **Question:** Is there a set $M \subseteq E$ of size k, such that M is independent in each $\mathcal{M}_i$, for $i \in [3]$?

Before moving further, we briefly explain why we needed an additional constraint that the input matrix is representable over $\mathbb{Q}$. Firstly, we will be using partition matroids which are representable only on fields of large enough size, and we want all the matroids, i.e. the one which is part of the input and the ones that we create, to be representable over the same field. Secondly, the algorithmic result (with the desired running time) we use for 3-Matroid Intersection works only for certain types of fields.

Next, we state an algorithmic result regarding 3-Matroid Intersection [24], which is be used by the algorithm. We note that we only state a restricted form of the algorithmic result for 3-Matroid Intersection in [24], which is enough for our purpose.

▶ **Proposition 15** (Proposition 4.8 [24] (partial)). 3-Matroid Intersection *can be solved in $\mathcal{O}(2^{3\omega k}\|A_M\|^{\mathcal{O}(1)})$ time, when the matroids are represented over $\mathbb{Q}$.*

We are now ready to prove the desired result.

▶ **Theorem 16.** Rat Mat CF-BM *can be solved in $\mathcal{O}(2^{3\omega k}\|A_M\|^{\mathcal{O}(1)})$ time.*

Proof. Let $(G = (V, E), L, R, A_{\mathcal{M}}, k)$ be an instance of Rat Mat CF-BM, where the matrix $A_{\mathcal{M}}$ represents a matroid $\mathcal{M} = (E, \mathcal{I})$ over $\mathbb{Q}$ and L, R is a partition of V into independent sets. Let $\mathcal{M}_L = (E, \mathcal{I}_L), \mathcal{M}_R = (E, \mathcal{I}_R)$ be the partition matroids as defined in Section 4. Next we compute matrix representations $A_{\mathcal{M}_L}$ and $A_{\mathcal{M}_R}$ of matroids $\mathcal{M}_L, \mathcal{M}_R$, respectively, using Proposition 4. Now, we solve 3-Matroid Intersection on the instance $(\mathcal{M}, A_{\mathcal{M}_L}, A_{\mathcal{M}_R}, k)$ (over $\mathbb{Q}$) using Proposition 15, and return the same answer, as returned by the algorithm in it. The correctness follows directly from the following. $S \subseteq E$ is a matching in G if and only if S is an independent set in $\mathcal{M}_L$ and $\mathcal{M}_R$, that is $S \in \mathcal{I}_L \cap \mathcal{I}_R$. The claimed running time follows from Proposition 4 and Proposition 15. ◀

5.1.2 FPT algorithm for Rat Mat CF-MM

We design an FPT algorithm for Rat Mat CF-MM, using the algorithm for Rat Mat CF-BM (Theorem 13). Let $(G, A_{\mathcal{M}}, k)$ be an instance of Rat Mat CF-MM, where the matrix $A_{\mathcal{M}}$ represents a matroid $\mathcal{M} = (E, \mathcal{I})$ over $\mathbb{Q}$. We assume that G is a graph with the vertex set $[n]$, which can easily be achieved by renaming vertices.

The algorithm starts by computing an $(n, 2k)$-universal set $\mathcal{F}$, using Proposition 9. For each set $X \in \mathcal{F}$, the algorithm constructs an instance $I_X = (G_X, L_X, R_X, A_{\mathcal{M}}, k)$ of Rat Mat CF-BM as follows. We have $V(G_X) = V(G)$, $L_X = X$, $R = V(G) \setminus X$, $E(G_X) = \{uv \in E(G) \mid u \in L_X, v \in R_X\}$. The algorithm decides the instance I_X using Theorem 16, for each $X \in \mathcal{F}$. The algorithm outputs yes if and only if there is $X \in \mathcal{F}$, such that I_X is a yes instance of Rat Mat CF-BM.

▶ **Theorem 17.** *The algorithm presented for* Rat Mat CF-MM *is correct. Moreover, it runs in time $\mathcal{O}(2^{(3\omega+2)k}k^{\mathcal{O}(\log k)}\|A_M\|^{\mathcal{O}(1)}n^{\mathcal{O}(1)})$.*

Proof. Let $(G, A_{\mathcal{M}}, k)$ be an instance of Rat Mat CF-MM, where matrix $A_{\mathcal{M}}$ represent a matroid $\mathcal{M} = (E, \mathcal{I})$ over field $\mathbb{F}$. Clearly, if the algorithm outputs yes, then indeed $(G, A_{\mathcal{M}}, k)$ is a yes instance of Rat Mat CF-MM. Next, we argue that if $(G, A_{\mathcal{M}}, k)$ is a yes instance of Rat Mat CF-MM then the algorithm returns yes. Suppose there is a solution $M \subseteq E(G)$ to Rat Mat CF-MM in $(G, A_{\mathcal{M}}, k)$. Let $S = \{i, j \mid ij \in M\}$, and $L = \{i \mid \text{there is } j \in [n] \text{ such that } ij \in M \text{ and } i < j\}$. Observe that $|S| = 2k$. Since $\mathcal{F}$ is an $(n, 2k)$-universal set, there is $X \in \mathcal{F}$ such that $X \cap S = L$. Note that S is a solution to Rat Mat CF-BM in I_X. This together with Theorem 16 implies that the algorithm will return yes as the output.

Next, we prove the claimed running time of the algorithm. The algorithm computes $(n, 2k)$-universal set of size $\mathcal{O}(2^{2k}k^{\mathcal{O}(\log k)} \log n)$, in time $\mathcal{O}(2^{2k}k^{\mathcal{O}(\log k)}\ n \log n)$, using Proposition 9. Then for each $X \in \mathcal{F}$, the algorithm creates an instance I_X of Rat Mat CF-BM in polynomial time. Furthermore, it resolves the I_X of Rat Mat CF-BM in time $\mathcal{O}(2^{3\omega k}\|A_M\|^{\mathcal{O}(1)})$ using Theorem 16. Hence, the running time of the algorithm is bounded by $\mathcal{O}(2^{(3\omega+2)k}k^{\mathcal{O}(\log k)}\|A_M\|^{\mathcal{O}(1)}n^{\mathcal{O}(1)})$. ◀

5.2 FPT algorithm for Matroid CF-SP

In this section, we design an FPT algorithm for Matroid CF-SP. The algorithm is based on dynamic programming over representative families. Let $(G, s, t, A_{\mathcal{M}}, k)$ be an instance of Matroid CF-SP. Before moving to the description of the algorithm, we need to define some notations. For distinct vertices $u, v \in V(G)$ and an integer p, we define: $\mathcal{P}^p_{uv} = \{X \subseteq E(G) \mid |X| = p, \text{ there is a } u - v \text{ path in } G[X] \text{ containing all edges in } X, \text{ and } X \in \mathcal{I}\}$.

By the definition of convolution of sets, it is easy to see that $\mathcal{P}^p_{uv} = \bigcup_{wv \in E(G)} \mathcal{P}^{p-1}_{uw} \star \{\{wv\}\}$. Now we are ready to describe our algorithm for Matroid CF-SP. We aim to store, for each $v \in V(G) \setminus \{s\}$, $p \leq k$, and $q \leq k - p$, a q-representative set $\widehat{\mathcal{P}}^{pq}_{sv}$, of $\mathcal{P}^p_{sv}$, of size $\binom{p+q}{q}$. Notice that for each $v \in V(G) \setminus \{s\}$, we can compute $\mathcal{P}^1_{sv}$ in polynomial time, since $\mathcal{P}^1_{sv} = \{sv\}$ if $sv \in E(G)$, and is empty otherwise. Moreover, since $|\mathcal{P}^1_{sv}| \leq 1$, therefore, we can set $\widehat{\mathcal{P}}^{1q}_{sv} = \mathcal{P}^1_{sv}$, for each $q \leq k - 1$. Next, we iteratively compute, for each $p \in \{2, 3, \cdots, k\}$, in increasing order, for each $q \leq k - p$, a q-representative $\widehat{\mathcal{P}}^{pq}_{sv}$, of $\mathcal{P}^p_{sv}$. From the above discussions, we obtain the following theorem.

▶ **Theorem 18.** *The algorithm Alg-Mat-CF-SP is correct, and runs in time* $\mathcal{O}(2^{\mathcal{O}(\omega k)}n^{\mathcal{O}(1)})$.

6 FPT Algorithm for d-degenerate Conflict Graphs

In this section, we show that CF-MM and CF-SP both are in FPT, when the conflict graph H is a d-degenerate graphs. These algorithms are based on the notion of independence covering family, which was introduced in [25], defined below.

▶ **Definition 19** ([25]). For a graph $H^\star$ and an integer k, a *k-independence covering family*, $\mathscr{I}(H^\star, k)$, is a family of independent sets in $H^\star$ such that for any independent set I' in $H^\star$ of size at most k, there is a set $I \in \mathscr{I}(H^\star, k)$ such that $I' \subseteq I$.

Our algorithms rely on the construction of k-independence covering family, for a family of graphs. We first design an algorithm for an annotated version of the CF-MM and CF-SP problems, which we call Annotated CF-MM and Annotated CF-SP, respectively. In the Annotated CF-MM (Annotated CF-SP) problem, the input to CF-MM (CF-SP) is annotated with a k-independence covering family $\mathcal{F}$ of H.

Algorithm 1 Alg-CF-MM (Alg-CF-SP).

```
Input: A graph $G$,((distinct) vertices $s, t \in V(G)$), a conflict graph $H$, an integer $k$,
       and a $k$-independence covering family $\mathcal{F}$ of $H$.
Output: If there a set $M \subseteq E$ of size $k$ in $G$ such that $M$ is a matching in $G$ (there
        is an $s - t$ path in $G[M]$) and $M$ is an independent set in $H$, then yes, and
        no otherwise.
for each $I \in \mathcal{F}$ do
    Let $G_I$ be the graph with $V(G_I) = V(G)$ and $E(G_I) = I$ ;
    if $G_I$ has a matching (path) of size $k$ then
        return yes;
end
return no ;
```

6.1 Algorithms for Annotated CF-MM and Annotated CF-SP

In this section, we study the problems Annotated CF-MM and Annotated CF-SP. The algorithm that we design for them run in time polynomial in the size of the input. We give the algorithm Alg-CF-MM (Alg-CF-SP) (Algorithm 1) for Annotated CF-MM (Annotated CF-SP).

In the following lemma we prove the correctness of Alg-CF-MM (Alg-CF-SP).

▶ **Lemma 20.** *The algorithm Alg-CF-MM (Alg-CF-SP) is correct. Moreover, the algorithm runs in time polynomial in the size of the input.*

We use Alg-CF-MM (Alg-CF-SP) together with Independence Covering Lemma of [25] to obtain algorithms for CF-MM (CF-SP) when the conflict graph is d-degenerate or nowhere dense graph. Towards this we state some lemmata from [25] that we use in our algorithms.

▶ **Proposition 21.** [25, Lemma 1.1] *There is a randomized algorithm running in polynomial time, that given a d-degenerate graph $H^\star$ and an integer k as input, outputs an independent set I, such that for every independent set I' of size at most k in graph $H^\star$, the probability that $I' \subseteq I$ is at least $(\binom{k(d+1)}{k} \cdot k(d+1))^{-1}$.*

▶ **Proposition 22.** [25, Lemmas 3.2 and 3.3] *There are two deterministic algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, which given a d-degenerate graph $H^\star$ and an integer k, output independence covering families $\mathscr{I}_1(H^\star, k)$ and $\mathscr{I}_2(H^\star, k)$, respectively, such that the following conditions are satisfied: i) $\mathcal{A}_1$ runs in time $\mathcal{O}(|\mathscr{I}_1(H^\star, k)| \cdot (n+m))$, where $|\mathscr{I}_1(H^\star, k)| = \binom{k(d+1)}{k} \cdot 2^{o(k(d+1))} \cdot \log n$ and ii) $\mathcal{A}_2$ runs in time $\mathcal{O}(|\mathscr{I}_2(H^\star, k)| \cdot (n+m))$, where $|\mathscr{I}_2(H^\star, k)| = \binom{k^2(d+1)^2}{k} \cdot (k(d+1))^{\mathcal{O}(1)} \cdot \log n$.*

Next, using Proposition 21 and 22, together with Alg-CF-MM (Alg-CF-SP), we obtain randomized and deterministic algorithms, respectively for CF-MM (CF-SP), when the conflict graph is a d-degenerate graph.

▶ **Theorem 23.** *There is a randomized algorithm, which given an instance (G, H, k) of* CF-MM(CF-SP)*, where H is a d-degenerate graph, in time $\binom{k(d+1)}{k} \cdot k(d+1) \cdot n^{\mathcal{O}(1)}$, either reports a failure or correctly outputs that the input is a* yes *instance of* CF-MM(CF-SP)*. Moreover, if the input is a* yes *instance of* CF-MM(CF-SP)*, then the algorithm outputs correct answer with a constant probability.*

Proof. Let $(G,(s,t),H,k)$ be an instance CF-MM (CF-SP), where H is a d-degenerate graph. We repeat the following procedure $(\binom{k(1+d)}{k} \cdot k(d+1))$ many times: i) the algorithm computes an independent set I in (H,k) using Proposition 21, and ii) the algorithm calls Alg-CF-MM (Alg-CF-SP) with input $(G,(s,t)H,k,\{I\})$.

The algorithm outputs yes, if in one of the calls to Alg-CF-MM (Alg-CF-SP), it receives a yes. Otherwise, the algorithm outputs no. The running time analysis of the above procedure follows from Proposition 21 and Lemma 20. Also, given a yes instance, the guarantee on success probability follows from Proposition 21, the number of repetitions, and Lemma 20. Moreover, from Lemma 20 the yes output returned by the algorithm is indeed the correct output to CF-MM(CF-SP)for the given instance. This concludes the proof. ◀

▶ **Theorem 24.** CF-MM (CF-SP) *admits a deterministic algorithm running in time* $\min\left\{\binom{k(d+1)}{k} \cdot 2^{o(k(d+1))} \cdot \log n, \binom{k^2(d+1)^2}{k} \cdot (k(d+1))^{\mathcal{O}(1)} \cdot \log n\right\} \cdot n^{\mathcal{O}(1)}$, *when the conflict graph is a d-degenerate graph.*

Proof. Let $(G,(s,t),H,k)$ be an instance CF-MM (CF-SP), where H is a d-degenerate graph. The algorithm starts by computing a k-independence covering family $\mathscr{I}(H,k)$ of H, using Proposition 22. Next, we call Alg-CF-MM (Alg-CF-SP) with the input $(G,(s,t),H,k,\mathscr{I}(H,k))$. The correctness and running time analysis of the above procedure follows from Proposition 22 and Lemma 20. This completes the proof. ◀

7 Conclusion

We studied conflict-free (parameterized) variants of Maximum Matching (CF-MM) and Shortest Path (CF-SP). We showed that both CF-MM and CF-SP are W[1]-hard, when parameterized by the solution size. In fact, our W[1]-hardness result for CF-MM holds even when the graph where we want to compute a matching is itself a matching and W[1]-hardness result of CF-SP holds even when the conflict graph is a unit interval graph. Then, we restricted our attention to having conflict graphs belonging to some families of graphs, where the Independent Set problem is either polynomial time solvable or solvable in FPT time. In particular, we considered the family of chordal graphs and the family of d-degenerate graphs. For the CF-MM problem, we gave an FPT algorithm, when the conflict graph belongs to the family of chordal graphs. We observed that, we cannot obtain an FPT algorithm for the CF-SP problem when the conflict graph is a chordal graph. This holds because unit-interval graphs are chordal, and the problem CF-SP is W[1]-hard, even when the conflict graph is a unit-interval graph. For conflict graphs being d-degenerate, we obtained FPT algorithms for both CF-MM and CF-SP. Our results hold even when the conflict graph is a nowhere dense graph. Finally, we studied a variant of CF-MM and CF-SP, where instead of conflicting conditions being imposed by independent sets in a conflict graph, they are imposed by independence constraints in a (representable) matroid. We gave FPT algorithms for the above variant of both CF-MM and CF-SP.

An interesting question is to obtain (parameterized) dichotomy results for CF-MM and CF-SP, based on the families of graphs where the input graphs belong to. Another direction could be studying kernelization complexity for different families of graphs, and also to see what all FPT problems remain FPT with the conflicting constraints.

References

1 Akanksha Agrawal, Pallavi Jain, Lawqueen Kanesh, Daniel Lokshtanov, and Saket Saurabh. Conflict Free Feedback Vertex Set: A Parameterized Dichotomy. *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS*, pages 53:1–53:15, 2018.

2 Akanksha Agrawal, Pallavi Jain, Lawqueen Kanesh, Pranabendu Misra, and Saket Saurabh. Exploring the Kernelization Borders for Hitting Cycles. *13th International Symposium on Parameterized and Exact Computation, IPEC*, pages 14:1–14:14, 2018.

3 Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

4 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.

5 Andreas Darmann, Ulrich Pferschy, and Joachim Schauer. Determining a minimum spanning tree with disjunctive constraints. *International Conference on Algorithmic DecisionTheory (ADT)*, pages 414–423, 2009.

6 Andreas Darmann, Ulrich Pferschy, Joachim Schauer, and Gerhard J. Woeginger. Paths, trees and matchings under disjunctive constraints. *Discrete Applied Mathematics*, 159(16):1726–1735, 2011.

7 Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

8 Rodney G. Downey and Michael R. Fellows. Fixed Parameter Tractability and Completeness. *Complexity Theory: Current Research*, pages 191–225, 1992.

9 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

10 Leah Epstein, Lene M. Favrholdt, and Asaf Levin. Online variable-sized bin packing with conflicts. *Discrete Optimization*, 8(2):333–343, 2011.

11 Guy Even, Magnús M. Halldórsson, Lotem Kaplan, and Dana Ron. Scheduling with conflicts: online and offline algorithms. *Journal of Scheduling*, 12(2):199–224, 2009.

12 Shimon Even and Oded Kariv. An $O(n^{2.5})$ algorithm for maximum matching in general graphs. *Foundations of Computer Science (FOCS)*, pages 100–112, 1975.

13 J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series).* Springer-Verlag, Secaucus, NJ, USA, 2006.

14 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient Computation of Representative Families with Applications in Parameterized and Exact Algorithms. *Journal of the ACM*, 63(4):29:1–29:60, 2016.

15 Fedor V Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. *Symposium on Discrete Algorithms (SODA)*, pages 142–151, 2014.

16 Harold N. Gabow, Shachindra N Maheshwari, and Leon J. Osterweil. On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, 2(3):227–231, 1976.

17 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.

18 Michel Gendreau, Gilbert Laporte, and Frédéric Semet. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & OR*, 31(3):347–358, 2004.

19 Pallavi Jain, Lawqueen Kanesh, and Pranabendu Misra. Conflict Free Version of Covering Problems on Graphs: Classical and Parameterized. *Computer Science - Theory and Applications - 13th International Computer Science Symposium in Russia (CSR)*, pages 194–206, 2018.

20 Klaus Jansen. An approximation scheme for bin packing with conflicts. *Journal of combinatorial optimization*, 3(4):363–377, 1999.

21 Minghui Jiang. On the parameterized complexity of some optimization problems related to multiple-interval graphs. *Theoretical Computer Science*, 411(49):4253–4262, 2010.

22 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science.* Springer, 1994.

23 KW Krause, MA Goodwin, and RW Smith. *Optimal software test planning through automated network analysis.* TRW Systems Group, 1973.

24 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 922–934, 2015.

25 Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Covering small independent sets and separators with applications to parameterized algorithms. *Symposium on Discrete Algorithms (SODA)*, pages 2785–2800, 2018.

26 Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, 2009.

27 Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, 2009.

28 Silvio Micali and Vijay V Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. *Foundations of Computer Science (FOCS)*, pages 17–27, 1980.

29 Moni Naor, Leonard J Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. *Foundations of Computer Science (FOCS)*, pages 182–191, 1995.

30 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

31 James G Oxley. *Matroid theory*, volume 3. Oxford University Press, 2006.

32 Ulrich Pferschy and Joachim Schauer. The Knapsack Problem with Conflict Graphs. *Journal of Graph Algorithms and Applications*, 13(2):233–249, 2009.

33 Ulrich Pferschy and Joachim Schauer. The maximum flow problem with conflict and forcing conditions. *Network Optimization*, pages 289–294, 2011.

34 Ulrich Pferschy and Joachim Schauer. The maximum flow problem with disjunctive constraints. *Journal of Combinatorial Optimization*, 26(1):109–119, 2013.

35 Ulrich Pferschy and Joachim Schauer. Approximation of knapsack problems with conflict and forcing graphs. *Journal of Combinatorial Optimization*, 33(4):1300–1323, 2017.

36 Virginia Vassilevska Williams. Multiplying Matrices Faster Than Coppersmith-winograd. *Symposium on Theory of Computing (STOC)*, pages 887–898, 2012.

On the Strength of Uniqueness Quantification in Primitive Positive Formulas

Victor Lagerkvist
Department of Computer and Information Science, Linköping University, Linköping, Sweden
victor.lagerkvist@liu.se

Gustav Nordh
Independent researcher, Hällekis, Sweden
gustav.nordh@gmail.com

Abstract

Uniqueness quantification ($\exists!$) is a quantifier in first-order logic where one requires that exactly one element exists satisfying a given property. In this paper we investigate the strength of uniqueness quantification when it is used in place of existential quantification in conjunctive formulas over a given set of relations Γ, so-called *primitive positive definitions* (pp-definitions). We fully classify the Boolean sets of relations where uniqueness quantification has the same strength as existential quantification in pp-definitions and give several results valid for arbitrary finite domains. We also consider applications of $\exists!$-quantified pp-definitions in computer science, which can be used to study the computational complexity of problems where the number of solutions is important. Using our classification we give a new and simplified proof of the trichotomy theorem for the unique satisfiability problem, and prove a general result for the unique constraint satisfaction problem. Studying these problems in a more rigorous framework also turns out to be advantageous in the context of lower bounds, and we relate the complexity of these problems to the *exponential-time hypothesis*.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases Primitive positive definitions, clone theory, constraint satisfaction problems

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.36

Related Version `https://arxiv.org/abs/1906.07031`

Acknowledgements We thank Andrei Bulatov for helpful discussions concerning the topic of the paper, and the anonymous reviewers for their constructive feedback.

1 Introduction

A *primitive positive definition* (pp-definition) over a relational structure $\mathcal{A} = (A; R_1, \ldots, R_k)$ is a first-order formula $\exists y_1, \ldots, y_m \colon \varphi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ with free variables $x_1, \ldots, x_n$ where $\varphi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is a conjunctive formula. Primitive positive definitions have been extremely influential in the last decades due to their one-to-one correspondence with term algebras in universal algebra, making them a cornerstone in the *algebraic approach* for studying computational complexity [1, 10]. In short, pp-definitions can be used to obtain classical "gadget reductions" between problems by replacing constraints by their pp-definitions, which in the process might introduce fresh variables viewed as being existentially quantified. This approach has successfully been used to study the complexity of e.g. the *constraint satisfaction problem* (CSP) which recently led to a dichotomy between tractable and NP-complete CSPs [6, 28]. However, these reductions are typically not sufficient for optimisation problems and other variants of satisfiability, where one needs reductions preserving the number of models, so-called *parsimonious* reductions. Despite the tremendous advances in the algebraic approach there is currently a lack of methods for studying problems requiring parsimonious reductions, and in this paper we take the first step in developing such a framework. The requirement of parsimonious reductions can be realised by restricting existential quantification

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 36; pp. 36:1–36:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to *unique quantification* ($\exists!$), where we explicitly require that the variable in question can be expressed as a unique combination of other variables. That is, $\mathcal{A} \models \exists! x_i \colon \varphi(x_1, \ldots, x_i, \ldots, x_n)$ if and only if there exists a function f such that $f(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) = a_i$, for all $a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n \in A$ where $\mathcal{A} \models \varphi(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)$. This notion of unique quantification is not the only one possible and we discuss an alternative viewpoint in Section 5. As a first step in understanding the applicability of uniqueness quantification in complexity classifications we are interested in studying the expressive power of unique existential quantification when used in place of existential quantification in pp-definitions, which we call *upp-definitions*. Any variables introduced by the resulting gadget reductions are then uniquely determined and do not affect the number of models.

Our main question is then: for which relational structures $\mathcal{A}$ is it the case that for every pp-formula $\varphi(x_1, \ldots, x_n)$ there exists a upp-formula $\vartheta(x_1, \ldots, x_n)$ such that $\mathcal{A} \models \varphi(a_1, \ldots, a_n) \Leftrightarrow \mathcal{A} \models \vartheta(a_1, \ldots, a_n)$ for all $a_1, \ldots, a_n \in A$? If this holds over $\mathcal{A}$ then uniqueness quantification has the same expressive power as existential quantification. The practical motivation for studying this is that if upp-definitions are as powerful as pp-definitions in $\mathcal{A}$, then, intuitively, any gadget reduction between two problems can be replaced with a parsimonious reduction. Given the generality of this question a complete answer for arbitrary relational structures is well out of reach, and we begin by introducing simplifying concepts. First, pp-definitions can be viewed as a closure operator over relations, and the resulting closed sets of relations are known as *relational clones*, or *co-clones* [21]. For each universe A the set of co-clones over A then forms a lattice when ordered by set inclusion, and given a set of relations Γ we write $\langle \Gamma \rangle$ for the smallest co-clone over A containing Γ. Similarly, closure under upp-definitions can also be viewed as a closure operator, and we write $\langle \Gamma \rangle_{\exists!}$ for the smallest set of relations over A containing Γ and which is closed under upp-definitions. Using these notions the question of the expressive strength of upp-definitions can be stated as: for which sets of relations Γ is it the case that $\langle \Gamma \rangle = \langle \Gamma \rangle_{\exists!}$? The main advantage behind this viewpoint is that a co-clone $\langle \Gamma \rangle$ can be described as the set of relations *invariant* under a set of operations F, $\mathrm{Inv}(F)$, such that the operations in F describe all permissible combinations of tuples in relations from Γ. An operation $f \in F$ is also said to be a *polymorphism* of Γ and if we let $\mathrm{Pol}(\Gamma)$ be the set of polymorphisms of Γ then $\mathrm{Pol}(\Gamma)$ is called a *clone*. This relationship allows us to characterise the cases that need to be considered by using known properties of $\mathrm{Pol}(\Gamma)$, which is sometimes simpler than working only on the relational side.

Our Results

Our main research question is to identify Γ such that $\langle \Delta \rangle_{\exists!} = \langle \Gamma \rangle$ for each Δ such that $\langle \Delta \rangle = \langle \Gamma \rangle$. If this holds we say that $\langle \Gamma \rangle$ is $\exists!$-*covered*. The main difficulty for proving this is that it might not be possible to directly transform a pp-definition into an equivalent upp-definition. To mitigate this we analyse relations in co-clones using *partial polymorphisms*, which allows us to analyse their expressibility in a very nuanced way. In Section 3.1 we show how partial polymorphisms can be leveraged to prove that a given co-clone is $\exists!$-covered. Most notably, we prove that $\langle \Gamma \rangle$ is $\exists!$-covered if $\mathrm{Pol}(\Gamma)$ consists only of projections of the form $\pi(x_1, \ldots, x_i, \ldots, x_n) = x_i$, or of projections and constant operations. As a consequence, Γ pp-defines all relations over A if and only if Γ upp-defines all relations over A. One way of interpreting this result is that if Γ is "sufficiently expressive" then pp-definitions can always be turned into upp-definitions. However, there also exists $\exists!$-covered co-clones where the reason is rather that Γ is "sufficiently weak". For example, if Γ is invariant under the affine operation $x - y + z \pmod{|A|}$, then existential quantification does not add any expressive power over unique existential quantification, since any existentially quantified variable occurring in a pp-definition can be expressed via a linear equation, and is therefore uniquely determined by other arguments. In Section 3.2 we then turn to the Boolean domain, and obtain a full classification of the ∃!-covered co-clones. Based on the results in Section 3.1 it is reasonable to expect that the covering property holds for sufficiently expressive languages and sufficiently weak languages, but that there may exist cases in between where unique quantification differs from existential quantification. This is indeed true, and we prove that the Boolean co-clones corresponding to non-positive Horn clauses, implicative and positive clauses, and their dual cases, are not ∃!-covered. Last, in Section 4 we demonstrate how the results from Section 3 can be used for obtaining complexity classifications of computational problems. One example of a problem requiring parsimonious reductions is the *unique satisfiability problem* over a Boolean set of relations Γ (U-SAT(Γ)) and its multi-valued generalization the *unique constraint satisfaction problem* (U-CSP(Γ)), where the goal is to determine if there exists a unique model of a given conjunctive Γ-formula. The complexity of U-SAT(Γ) was settled by Juban [15] for finite sets of relations Γ, essentially using a large case analysis. Using the results from Section 3.2 this complexity classification can instead be proved in a succinct manner, and we are also able to extend the classification to infinite Γ and large classes of non-Boolean Γ. This systematic approach is also advantageous for proving lower bounds, and we relate the complexity of U-SAT(Γ) to the highly influential *exponential-time hypothesis* (ETH) [12], by showing that none of the intractable cases of U-SAT(Γ) admit subexponential algorithms without violating the ETH.

Related Work

Primitive positive definitions with uniqueness quantification appeared in Creignou & Hermann [7] in the context of "quasi-equivalent" logical formulas, and in the textbook by Creignou et al. [8] under the name of *faithful implementations*. Similarly, upp-definitions were utilised by Kavvadias & Sideri [16] to study the complexity of the *inverse satisfiability problem*. A related topic is *frozen quantification*, which can be viewed as uniqueness quantification restricted to variables that are constant in any model [22].

2 Preliminaries

2.1 Operations and Relations

In the sequel, let $D \subseteq \mathbb{N}$ be a finite domain of values. A k-ary function $f \colon D^k \to D$ is sometimes referred to as an *operation* over D and we write $\mathrm{ar}(f)$ to denote the arity k. Similarly, a *partial operation* over D is a map $f \colon \mathrm{dom}(f) \to D$ where $\mathrm{dom}(f) \subseteq D^k$ is called the *domain* of f, and we let $\mathrm{ar}(f) = k$ be the arity of f. If f and g are k-ary partial operations such that $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and $f(t) = g(t)$ for each $t \in \mathrm{dom}(f)$ then f is said to be a *suboperation* of g. For $k \geq 1$ and $1 \leq i \leq k$ we let π^k_i be the ith *projection*, i.e., $\pi^k_i(x_1, \ldots, x_i, \ldots, x_k) = x_i$ for all $x_1, \ldots, x_i, \ldots, x_k \in D$. We write OP_D for the set of all operations over D and pOP_D for the set of all partial operations over D. As a notational shorthand we for $k \geq 1$ write $[k]$ for the set $\{1, \ldots, k\}$. For $d \in D$ we by $\mathbf{d}^n$ denote the constant n-ary tuple $(d, \ldots, d)$. Say that a k-ary $f \in \mathsf{OP}_D$ is *essentially unary* if there exists unary $g \in \mathsf{OP}_D$ and i such that $f(x_1, \ldots, x_i, \ldots, x_n) = g(x_i)$ for all $x_1, \ldots, x_i, \ldots, x_n \in D$.

Given an n-ary relation $R \subseteq D^n$ we write $\mathrm{ar}(R)$ to denote its arity n. If $t = (x_1, \ldots, x_n)$ is an n-ary tuple we write $t[i]$ to denote the ith element x_i, and $\mathrm{Proj}_{i_1, \ldots, i_{n'}}(t) = (t[i_1], \ldots, t[i_{n'}])$ to denote the *projection* on the coordinates $i_1, \ldots, i_{n'} \in \{1, \ldots, n\}$. Similarly, if R is an n-ary relation we let $\mathrm{Proj}_{i_1, \ldots, i_{n'}}(R) = \{\mathrm{Proj}_{i_1, \ldots, i_{n'}}(t) \mid t \in R\}$. The ith argument of a relation R is said to be *redundant* if there exists $j \neq i$ such that $t[i] = t[j]$ for each $t \in R$, and is said to be *fictitious* if for all $t \in R$ and $d \in D$ have $t' \in R$ where $t'[i] = d$ and $\mathrm{Proj}_{1,\ldots,i-1,i+1,\ldots,n}(t) = \mathrm{Proj}_{1,\ldots,i-1,i+1,\ldots,n}(t')$.

We write Eq_D for the equality relation $\{(x,x) \mid x \in D\}$ over D. We often represent relations by their defining first-order formulas, and if $\varphi(x_1,\ldots,x_n)$ is a first-order formula with n free variables we write $R(x_1,\ldots,x_n) \equiv \varphi(x_1,\ldots,x_n)$ to define the relation $R = \{(f(x_1),\ldots,f(x_n)) \mid f \text{ is a model of } \varphi(x_1,\ldots,x_n)\}$. We let REL^n_D be the set of all n-ary relations over D, $\mathsf{REL}^{\leq n}_D = \bigcup_{i=1}^{n} \mathsf{REL}^n_D$, and $\mathsf{REL}_D = \bigcup_{i=1}^{\infty} \mathsf{REL}^i_D$. A set $\Gamma \subseteq \mathsf{REL}_D$ will sometimes be called a *constraint language*.

2.2 Primitive Positive Definitions and Determined Variables

We say that an n-ary relation R has a *primitive positive definition* (pp-definition) over a set of relations Γ over a domain D if $R(x_1,\ldots,x_n) \equiv \exists y_1,\ldots,y_{n'} \colon R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m)$ where each $\mathbf{x}_i$ is a tuple of variables over $x_1,\ldots,x_n,y_1,\ldots,y_{n'}$ of length $\mathrm{ar}(R_i)$ and each $R_i \in \Gamma \cup \{\mathrm{Eq}_D\}$. Hence, R can be defined as a (potentially) existentially quantified conjunctive formula over $\Gamma \cup \{\mathrm{Eq}_D\}$. We will occasionally be interested in pp-definitions not making use of existential quantification, and call pp-definitions of this restricted type *quantifier-free primitive positive definitions* (qfpp-definitions).

▶ **Definition 1.** *Let R be an n-ary relation over a domain D. We say that $1 \leq i \leq n$ is* uniquely determined*, or just* determined*, if there exists $i_1,\ldots,i_k \in [n]$ and a function $h : D^k \to D$ such that $h(t[i_1],\ldots,\ldots,t[i_k]) = t[i]$ for each $t \in R$.*

When defining relations in terms of logical formulas we will occasionally also say that the ith variable is uniquely determined, rather than the ith index.

▶ **Definition 2.** *An n-ary relation R has a* unique primitive positive definition *(upp-definition) over a set of relations Γ if there exists a pp-definition*

$$R(x_1,\ldots,x_n) \equiv \exists y_1,\ldots,y_{n'} \colon R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m)$$

of R over Γ where each y_i is uniquely determined by $x_1,\ldots,x_n$.

We typically write $\exists! y_1,\ldots,y_{n'}$ for the existentially quantified variables in a upp-definition. Following Nordh & Zanuttini [22] we refer to unique existential quantification over constant arguments as *frozen existential quantification* ($i \in [\mathrm{ar}(R)]$ is constant if there exists $d \in D$ such that $t[i] = d$ for each $t \in R$). If R is upp-definable over Γ via a upp-definition only making use of frozen existential quantification then we say that R is *freezingly pp-definable* (fpp-definable) over Γ. Let us define the following closure operators over relations.

▶ **Definition 3.** *Let Γ be a set of relations. Then we define (1) $\langle \Gamma \rangle = \{R \mid R$ has a pp-definition over $\Gamma\}$, (2), $\langle \Gamma \rangle_{\exists!} = \{R \mid R$ has a upp-definition over $\Gamma\}$, (3), $\langle \Gamma \rangle_{\mathrm{fr}} = \{R \mid R$ has an fpp-definition over $\Gamma\}$, and (4), $\langle \Gamma \rangle_{\nexists} = \{R \mid R$ has a qfpp-definition over $\Gamma\}$.*

In all cases Γ is called a *base*. If $\Gamma = \{R\}$ is singleton then we write $\langle R \rangle$ instead of $\langle \Gamma \rangle$, and similarly for the other operators. Sets of relations of the form $\langle \Gamma \rangle$ are usually called *relational clones*, or *co-clones*, sets of the form $\langle \Gamma \rangle_{\nexists}$ *weak systems*, or *weak partial co-clones*, and sets of the form $\langle \Gamma \rangle_{\mathrm{fr}}$ are known as *frozen partial co-clones.* Note that $\langle \Gamma \rangle \supseteq \langle \Gamma \rangle_{\exists!} \supseteq \langle \Gamma \rangle_{\mathrm{fr}} \supseteq \langle \Gamma \rangle_{\nexists}$ for any $\Gamma \subseteq \mathsf{REL}_D$. Co-clones and weak systems can be described via algebraic invariants known as *polymorphisms* and *partial polymorphism.* More precisely, if $R \in \mathsf{REL}^n_D$ and $f \in \mathsf{OP}_D$ is a k-ary operation, then for $t_1,\ldots,t_k \in R$ we let

$f(t_1, \ldots, t_k) = (f(t_1[1], \ldots, t_k[1]), \ldots, f(t_1[n], \ldots, t_k[n]))$. We then say that a k-ary partial operation f *preserves* an n-ary relation R if $f(t_1, \ldots, t_k) \in R$ or there exists $i \in [n]$ such that $(t_1[i], \ldots, t_k[i]) \notin \mathrm{dom}(f)$, for each sequence of tuples $t_1, \ldots, t_k \in R$. If f preserves R then R is also said to be *invariant* under f. Note that if f is total then the condition is simply that $f(t_1, \ldots, t_k) \in R$ for each sequence $t_1, \ldots, t_k \in R$. We then let $\mathrm{pPol}(R) = \{f \in \mathsf{pOP}_D \mid f \text{ preserves } R\}$, $\mathrm{Pol}(R) = \mathrm{pPol}(R) \cap \mathsf{OP}_D$, $\mathrm{pPol}(\Gamma) = \bigcap_{R \in \Gamma} \mathrm{pPol}(R)$, and $\mathrm{Pol}(\Gamma) = \bigcap_{R \in \Gamma} \mathrm{Pol}(R)$. Similarly, if F is a set of (partial) operations we let $\mathrm{Inv}(F)$ be the set of relations invariant under F, and write $\mathrm{Inv}(f)$ if $F = \{f\}$ is singleton. It is then known that $\mathrm{Inv}(F)$ is a co-clone if $F \subseteq \mathsf{OP}_D$ and that $\mathrm{Inv}(F)$ is a weak system if $F \subseteq \mathsf{pOP}_D$. More generally, $\langle \Gamma \rangle = \mathrm{Inv}(\mathrm{Pol}(\Gamma))$ and $\langle \Gamma \rangle_{\nexists} = \mathrm{Inv}(\mathrm{pPol}(\Gamma))$, resulting in the following *Galois connections*.

▶ **Theorem 4** ([3, 4, 11, 26]). *Let Γ and Δ be two sets of relations. Then $\Gamma \subseteq \langle \Delta \rangle$ if and only if $\mathrm{Pol}(\Delta) \subseteq \mathrm{Pol}(\Gamma)$ and $\Gamma \subseteq \langle \Delta \rangle_{\nexists}$ if and only if $\mathrm{pPol}(\Delta) \subseteq \mathrm{pPol}(\Gamma)$.*

Last, we remark that sets of the form $\mathrm{Pol}(\Gamma)$ and $\mathrm{pPol}(\Gamma)$ are usually called *clones*, and *strong partial clones*, respectively, and form lattices when ordered by set inclusion. Boolean clones are particularly well understood and the induced lattice is known as *Post's lattice* [24]. If $F \subseteq \mathsf{OP}_D$ then we write $[F]$ for the intersection of all clones over D containing F. Hence, $[F]$ is the smallest clone over D containing F.

2.3 Weak and Plain Bases of Co-Clones

In this section we introduce two special types of bases of a co-clone, that are useful for understanding the expressibility of upp-definitions.

▶ **Definition 5** (Schnoor & Schnoor [27]). *Let $\langle \Gamma \rangle$ be a co-clone. A base Γ_w of $\langle \Gamma \rangle$ with the property that $\langle \Gamma_w \rangle_{\nexists} \subseteq \langle \Delta \rangle_{\nexists}$ for every base Δ of $\langle \Gamma \rangle$ is called a* weak base *of $\langle \Gamma \rangle$.*

Although not immediate from Definition 5, Schnoor & Schnoor [27] proved that a weak base exists whenever $\langle \Gamma \rangle$ admits a finite base, by the following relational construction.

▶ **Definition 6.** *For $s \geq 1$, let $U^s_D = \{t_1, \ldots, t_s\}$ where $t_1, \ldots, t_s$ is the sequence of $|D|^s$-ary tuples where $(t_1[1], \ldots, t_s[1]), \ldots, (t_1[|D|^s], \ldots, t_s[|D|^s])$ is a lexicographic enumeration of D^s.*

For $R \in \mathsf{REL}_D$ and $F \subseteq \mathsf{OP}_D$ we let $F(R) = \bigcap_{R' \in \mathrm{Inv}(F), R \subseteq R' \in \mathsf{REL}_D} R'$. We typically write U^s instead of U^s_D if the domain D is clear from the context, and say that a co-clone $\mathrm{Inv}(\mathsf{C})$ has *core-size* s if there exist relations R, R' such that $\mathrm{Pol}(R) = \mathsf{C}$, $R = \mathsf{C}(R')$, and $s = |R'|$. Weak bases can then be described via core-sizes as follows (a clone C is finitely related if there exists a finite base of $\mathrm{Inv}(\mathsf{C})$).

▶ **Theorem 7** (Schnoor & Schnoor [27]). *Let C be a finitely related clone where $\mathrm{Inv}(\mathsf{C})$ has core-size s. Then $\mathsf{C}(U^t)$ is a weak base of $\mathrm{Inv}(\mathsf{C})$ for every $t \geq s$.*

See Table 2 for a list of weak bases for the Boolean co-clones of interest in this paper [17, 18]. Here, and in the sequel, we use the co-clone terminology developed by Reith & Wagner [25] and Böhler et al. [5], where a Boolean co-clone $\mathrm{Inv}(\mathsf{C})$ is typically written as IC. Many relations in Table 2 are provided by their defining logical formulas; for example, $x_1 \rightarrow x_2$ is the binary relation $\{(0,0), (0,1), (1,1)\}$. See Table 1 for definitions of the remaining relations. As a convention we use c_0 to indicate a variable which is constant 0 in any model, and c_1 for a variable which is constant 1. On the functional side we use the bases by Böhler et al. [5] and let $\mathsf{I}_2 = [\pi^1_1], \mathsf{I}_0 = [0], \mathsf{I}_1 = [1], \mathsf{I} = [\{0,1\}]$, $\mathsf{N}_2 = [\overline{x}]$, $\mathsf{N} = [\{\overline{x}, 0, 1\}]$, $\mathsf{E}_2 = [\wedge]$,

Table 1 Relations.

Relation	Definition
F	$\{(0)\}$
T	$\{(1)\}$
Ne	$\{(0,1),(1,0)\}$
n-EVEN	$\{(x_1,\ldots,x_n)\in\{0,1\}^n \mid x_1+\ldots+x_n \text{ is even}\}$
n-EVEN$^{n\neq}$	$n\text{-EVEN}(x_1,\ldots,x_n)\wedge \mathrm{Ne}(x_1,x_{n+1})\wedge\ldots\wedge\mathrm{Ne}(x_n,x_{2n})$
n-ODD	$\{(x_1,\ldots,x_n)\in\{0,1\}^n \mid x_1+\ldots+x_n \text{ is odd}\}$
n-ODD$^{n\neq}$	$n\text{-ODD}(x_1,\ldots,x_n)\wedge \mathrm{Ne}(x_1,x_{n+1})\wedge\ldots\wedge\mathrm{Ne}(x_n,x_{2n})$
NA^n	$\{0,1\}^n\setminus\{(1,\ldots,1)\}$

Table 2 Weak and plain bases of selected Boolean co-clones.

C	Weak base of Inv(C)	Plain base of Inv(C)
S_1^n	$\{\mathrm{NA}^n(x_1,\ldots,x_n)\wedge F(c_0)\}$	$\{\mathrm{NA}^n\}$
S_1	$\{\mathrm{NA}^n(x_1,\ldots,x_n)\wedge F(c_0) \mid n\geq 2\}$	$\{\mathrm{NA}^n \mid n\geq 1\}$
S_{12}^n	$\{\mathrm{NA}^n(x_1,\ldots,x_n)\wedge F(c_0)\wedge T(c_1)\}$	$\{\mathrm{NA}^n, T(c_1)\}$
S_{12}	$\{\mathrm{NA}^n(x_1,\ldots,x_n)\wedge F(c_0)\wedge T(c_1) \mid n\geq 2\}$	$\{\mathrm{NA}^n \mid n\geq 1\}\cup\{T(c_1)\}$
S_{11}^n	$\{\mathrm{NA}^n(x_1,\ldots,x_n)\wedge(\neg x\rightarrow\neg x_1\cdots\neg x_n)\wedge F(c_0)\}$	$\{\mathrm{NA}^n,(x_1\rightarrow x_2)\}$
S_{11}	$\{R_{\mathsf{S}_{11}^n} \mid n\geq 2\}$	$\{\mathrm{NA}^n \mid n\geq 1\}\cup\{(x_1\rightarrow x_2)\}$
S_{10}^n	$\{R_{\mathsf{S}_{11}^n}(x_1,\ldots,x_n,c_0)\wedge T(c_1)\}$	$\{\mathrm{NA}^n,(x_1\rightarrow x_2),T(c_1)\}$
S_{10}	$\{R_{\mathsf{S}_{10}^n} \mid n\geq 2\}$	$\{\mathrm{NA}^n \mid n\geq 1\}\cup\{(x_1\rightarrow x_2),T(c_1)\}$
D	$\{(x_1\oplus x_2=1)\}$	$\{(x_1\oplus x_2=1)\}$
D_1	$\{(x_1\oplus x_2=1)\wedge F(c_0)\}\wedge T(c_1)$	$\{(x_1\oplus x_2=1)\}\cup\{F(c_0),T(c_1)\}$
D_2	$\{(x_1\vee x_2)\wedge\mathrm{Ne}(x_1,x_3)\wedge\mathrm{Ne}(x_2,x_4)\wedge F(c_0)\wedge T(c_1)\}$	$\{(x_1\vee x_2),(\neg x_1\vee x_2),(\neg x_1\vee\neg x_2)\}$
E	$\{(x_1\leftrightarrow x_2x_3)\wedge(x_2\vee x_3\rightarrow x_4)\}$	$\{(\neg x_1\vee\ldots\vee\neg x_k\vee x) \mid k\geq 1\}$
E_0	$\{(x_1\leftrightarrow x_2x_3)\wedge(x_2\vee x_3\rightarrow x_4)\wedge F(c_0)\}$	$\{\mathrm{NA}^n \mid n\in\mathbb{N}\}\cup\{(\neg x_1\vee\ldots\vee\neg x_k\vee x) \mid k\geq 1\}$
E_1	$\{(x_1\leftrightarrow x_2x_3)\wedge T(c_1)\}$	$\{(\neg x_1\vee\ldots\vee\neg x_k\vee x) \mid k\in\mathbb{N}\}$
E_2	$\{(x_1\leftrightarrow x_2x_3)\wedge F(c_0)\wedge T(c_1)\}$	$\{\mathrm{NA}^n \mid n\in\mathbb{N}\}\cup\{(\neg x_1\vee\ldots\vee\neg x_k\vee x) \mid k\in\mathbb{N}\}$

$\mathsf{E}_0=[\{\wedge,0\}]$, $\mathsf{E}_1=[\{\wedge,1\}]$, $\mathsf{E}=[\{\wedge,0,1\}]$, $\mathsf{L}_2=[x\oplus y\oplus z]$, and $\mathsf{S}_{11}=[\{x\wedge(y\vee z),0\}]$, where $\overline{x}=1-x$ and where $0,1$ are shorthands for the two constant Boolean operations. We conclude this section by defining the dual notion of a weak base.

▶ **Definition 8** (Creignou et al. [9]). *Let $\langle\Gamma\rangle$ be a co-clone. A base Γ_p of $\langle\Gamma\rangle$ with the property that $\langle\Delta\rangle_{\nexists}\subseteq\langle\Gamma_p\rangle_{\nexists}$ for every base Δ of $\langle\Gamma\rangle$ is called a* plain base *of $\langle\Gamma\rangle$.*

Clearly, every co-clone is a trivial plain base of itself, but the question remains for which co-clones more succinct plain bases can be found. For arbitrary finite domains little is known but in the Boolean domain succinct plain bases have been described [9] (see Table 2).

2.4 Duality

Many questions concerning Boolean co-clones can be simplified by only considering parts of Post's lattice. If $f\in\mathsf{OP}_{\{0,1\}}$ is k-ary then the *dual* of f, $\mathrm{dual}(f)$, is the operation $\mathrm{dual}(f)(x_1,\ldots,x_k)=\overline{f(\overline{x_1},\ldots,\overline{x_k})}$, and we let $\mathrm{dual}(F)=\{\mathrm{dual}(f)\mid f\in F\}$ for a set $F\subseteq\mathsf{OP}_{\{0,1\}}$. Each Boolean clone C can then be associated with a dual clone $\mathrm{dual}(\mathsf{C})$. Similarly, for $R\in\mathsf{REL}_{\{0,1\}}$ we let $\mathrm{dual}(R)=\{\overline{t}\mid t\in R\}$ and $\mathrm{dual}(\Gamma)=\{\mathrm{dual}(R)\mid R\in\Gamma\}$ for $\Gamma\subseteq\mathsf{REL}_{\{0,1\}}$. It is then known that $\mathrm{Inv}(\mathrm{dual}(\mathsf{C}))=\mathrm{dual}(\mathrm{Inv}(\mathsf{C}))$.

3 The Expressive Power of Unique Existential Quantification

The main goal of this paper is to understand when the expressive power of unique existential quantification coincides with existential quantification in primitive positive formulas. Let us first consider an example where a pp-definition can be rewritten into a upp-definition.

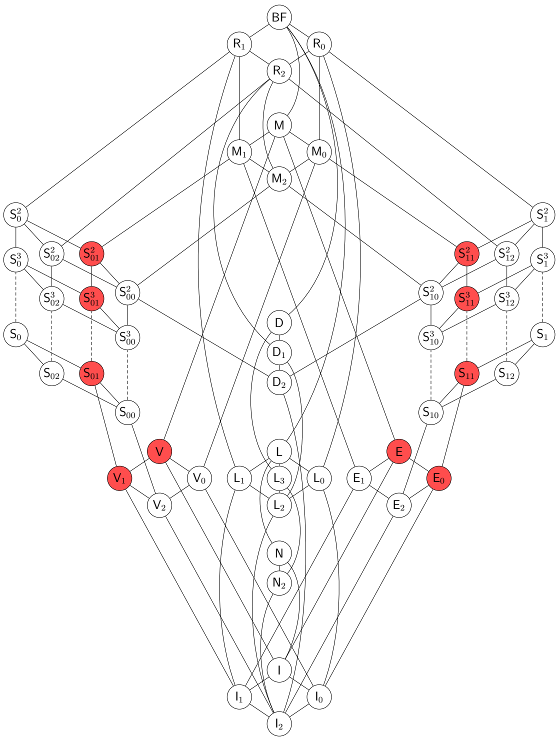


Figure 1 The lattice of Boolean clones. Inv(C) is coloured in red if and only if Inv(C) is not ∃!-covered.

▶ **Example 9.** Consider the canonical reduction from k-SAT to $(k-1)$-SAT via pp-definitions of the form $(x_1 \vee \ldots \vee x_k) \equiv \exists y \colon (x_1 \vee \ldots \vee x_{k-2} \vee y) \wedge (x_{k-1} \vee x_k \vee \neg y)$. In this pp-definition the auxiliary variable y is *not* uniquely determined since, for example, $y = 0$ and $y = 1$ are both consistent with $x_1 = 1, \ldots x_{k-2} = 1, x_{k-1} = 1, x_k = 1$. On the other hand, if we instead take the pp-definition $(x_1 \vee \ldots \vee x_k) \equiv \exists y \colon (x_1 \vee \ldots \vee x_{k-2} \vee y) \wedge (y \leftrightarrow (x_{k-1} \vee x_k))$, which can be expressed by $(k-1)$-SAT, it is easily verified that y is determined by x_{k-1} and x_k.

Using the algebraic terminology from Section 2 this property can be phrased as follows.

▶ **Definition 10.** *A co-clone $\langle \Gamma \rangle$ is* ∃!-covered *if $\langle \Gamma \rangle = \langle \Delta \rangle_{\exists!}$ for every base Δ of $\langle \Gamma \rangle$.*

Thus, we are interested in determining the ∃!-covered co-clones, and since every constraint language Γ belongs to a co-clone, namely $\langle \Gamma \rangle$, Definition 10 precisely captures the aforementioned question concerning the expressive strength of uniqueness quantification in primitive positive formulas. The remainder of this section will be dedicated to proving covering results of this form, with a particular focus on proving a full classification for the Boolean domain. See Figure 1 for a visualisation of this dichotomy. We begin in Section 3.1 by outlining some of the main ideas required to prove that a co-clone is ∃!-covered, and consider covering results applicable for arbitrary finite domains. In Section 3.2 we turn to the Boolean domain where we prove the classification in Figure 1. Throughout, the missing proofs can be found in the extended preprint [19], and the affected statements are marked with an asterisk $(*)$.

3.1 General Constructions

Given an arbitrary constraint language Γ it can be difficult to directly reason about the strength of upp-definitions over Γ. Fortunately, there are methods to mitigate this difficulty. Recall from Definition 5 that a weak base of a co-clone $\langle\Gamma\rangle$ is a base which is qfpp-definable by any other base of $\langle\Gamma\rangle$, and that a plain base is a base with the property that it can qfpp-define every relation in the co-clone. We then have the following useful lemma.

▶ **Lemma 11.** *Let $\langle\Gamma\rangle$ be a co-clone with a weak base Γ_w and a plain base Γ_p. If $\Gamma_p \subseteq \langle\Gamma_w\rangle_{\exists!}$ then $\langle\Gamma\rangle$ is $\exists!$-covered.*

Proof. Let Δ be a base of $\langle\Gamma\rangle$ and R an n-ary relation from $\langle\Gamma\rangle$, with a qfpp-definition $R(x_1,\ldots,x_n) \equiv \varphi(x_1,\ldots,x_n)$ over Γ_p. By assumption, Γ_w can upp-define every relation in Γ_p, and it follows that $R(x_1,\ldots,x_n) \equiv \exists! y_1,\ldots,y_m \colon \varphi'(x_1,\ldots,x_n,y_1,\ldots,y_m)$ for a Γ_w-formula $\varphi'(x_1,\ldots,x_n,y_1,\ldots,y_m)$ since each constraint in $\varphi(x_1,\ldots,x_n)$ can be replaced by its upp-definition over Γ_w. Last, since Δ can qfpp-define Γ_w, we obtain a upp-definition of R by replacing each constraint in $\varphi'(x_1,\ldots,x_n,y_1,\ldots,y_m)$ by its qfpp-definition over Δ. ◀

Although not difficult to prove, Lemma 11 offers the advantage that it is sufficient to prove that $\Gamma_p \subseteq \langle\Gamma_w\rangle_{\exists!}$ for two constraint languages Γ_w and Γ_p. Let us now illustrate some additional techniques for proving that $\langle\Gamma\rangle$ is $\exists!$-covered. Theorem 7 in Section 2.3 shows that the relation $\mathsf{C}(U^s)$ is a weak base of $\mathrm{Inv}(\mathsf{C})$ for s larger than or equal to the core-size of $\mathrm{Inv}(\mathsf{C})$. For s smaller than the core-size we have the following description of $\mathsf{C}(U^s)$.

▶ **Theorem 12 (*).** *Let C be a finitely related clone over a finite domain D. Then, for every $s \geq 1$, $\mathsf{C}(U^s) \in \langle\Gamma\rangle_{\nexists}$ for every base Γ of $\mathrm{Inv}(\mathsf{C})$.*

The applications of Theorem 12 in the context of upp-definitions might not be immediate. However, observe that each argument $i \in [|D|^s]$ of U^s is determined by at most s other arguments, and if C is sufficiently simple, this property can be proved to hold also for $\mathsf{C}(U^s)$. This intuition can then be formalised into the following general theorem.

▶ **Theorem 13.** *Let $\mathrm{Pol}(\Gamma)$ be a clone over a finite domain D such that each $f \in \mathrm{Pol}(\Gamma)$ is a constant operation or a projection. Then $\langle\Gamma\rangle$ is $\exists!$-covered.*

Proof. Let F be a set of operations such that $[F] = \mathrm{Pol}(\Gamma)$. We may without loss of generality assume that $F = \{f_1,\ldots,f_k\}$ for unary operations f_l such that $f_l(x) = d_l$ for some $d_l \in D$. Take an arbitrary n-ary relation $R \in \langle\Gamma\rangle$. Let $s = |R|$ and consider the relation $F(U^s)$ from Definition 6. Our aim is to prove that $F(U^s)$ can upp-define R, which is sufficient since $F(U^s) \in \langle\Gamma\rangle_{\nexists}$ via Theorem 12. Let $i_1,\ldots,i_n \in [|D|^s]$ denote the indices satisfying $\mathrm{Proj}_{i_1,\ldots,i_n}(F(U^s)) = R$. If $k = 0$, and $\mathrm{Pol}(\Gamma)$ consists only of projections, then $F(U^s) = U^s$, and each argument in $[|D|^s] \setminus \{i_1,\ldots,i_n\}$ is already determined by $i_1,\ldots,i_n$, and by the preceding remark $R \in \langle F(U^s)\rangle_{\exists!}$. Therefore, assume that $k \geq 1$. For each $f_l \in F$ then observe that $(d_l,\ldots,d_l) \in F(U^s)$ and that $(d_l,\ldots,d_l) \in \mathrm{Proj}_{i_1,\ldots,i_n}(U^s)$. Choose $j_1, j_2 \in [|D|^s]$ such that $t[j_1] \neq t[j_2]$ for $t \in U^s$ if and only if $\mathrm{Proj}_{i_1,\ldots,i_n}(t) = (d_l,\ldots,d_l)$, for a d_l such that $f_l(x) = d_l$. Thus, we choose a pair of indices differing in U^s if and only if the projection on $i_1,\ldots,i_n$ is constant. Such a choice is always possible since the arguments of U^s enumerate all s-ary tuples over D. Then construct the relation $R'(x_1,\ldots,x_{|D|^s}) \equiv F(U^s)(x_1,\ldots,x_{|D|^s}) \wedge \mathrm{Eq}(x_{j_1},x_{j_2})$. It follows that $\mathrm{Proj}_{i_1,\ldots,i_n}(R') = R$, and that every argument $l \in [|D|^s] \setminus \{i_1,\ldots,i_n\}$ is determined by $i_1,\ldots,i_n$. Hence, $R \in \langle F(U^s)\rangle_{\exists!}$. ◀

Theorem 13 implies that $\langle\Gamma\rangle$ is $\exists!$-covered if Γ is sufficiently powerful, and in particular implies that REL_D is $\exists!$-covered for every finite D. Hence, Γ pp-defines every relation if and only if Γ upp-defines every relation. However, as we will now illustrate, this is not the only possible case when a co-clone is $\exists!$-covered.

▶ **Lemma 14** (∗). *Let F be a set of operations over a finite domain D. If each argument $i \in [\mathrm{ar}(R)]$ is either fictitious or determined for every $R \in \mathrm{Inv}(F)$, then $\mathrm{Inv}(F)$ is $\exists!$-covered.*

▶ **Theorem 15.** *Let D be a finite domain such that $|D|$ is prime, and let $f(x, y, z) = x - y + z \pmod{|D|}$. Then, for any constraint language Γ over D such that $\langle\Gamma\rangle \subseteq \mathrm{Inv}(f)$, $\langle\Gamma\rangle$ is $\exists!$-covered.*

Proof. We will prove that the preconditions of Lemma 14 are satisfied for $\mathrm{Inv}(f)$, which is sufficient to prove the claim. Let R be invariant under f. Then it is known that R is the solution space of a system of linear equations modulo $|D|$ [14], from which it follows that each argument is either determined, since it can be written as a unique combination of other arguments, or is fictitious. ◀

3.2 Boolean Constraint Languages

In this section we use the techniques developed so far to prove that the classification in Figure 1 is correct. Note first that $\mathrm{Inv}(\mathsf{C})$ is $\exists!$-covered if and only if $\mathrm{Inv}(\mathrm{dual}(\mathsf{C}))$ is $\exists!$-covered, since a upp-definition $\exists! y_1, \ldots, y_{n'} \colon R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m)$ of n-ary $R \in \mathrm{Inv}(\mathsf{C})$ immediately yields a upp-definition $\exists! y_1, \ldots, y_{n'} \colon \mathrm{dual}(R_1)(\mathbf{x}_1) \wedge \ldots \wedge \mathrm{dual}(R_m)(\mathbf{x}_m)$ of $\mathrm{dual}(R) \in \mathrm{Inv}(\mathrm{dual}(\mathsf{C}))$. Thus, to simplify the presentation we omit the case when $\mathsf{C} \supseteq \mathsf{V}_2$ in Figure 1. Let us begin with the cases following directly from Section 3.1 or from existing results (recall that IC is a shorthand for $\mathrm{Inv}(\mathsf{C})$).

▶ **Lemma 16.** *Let IC be a Boolean co-clone. Then IC is $\exists!$-covered if $\mathsf{IC} \subseteq \mathsf{IM}_2$, $\mathsf{IC} \subseteq \mathsf{IL}_2$, $\mathsf{IC} \subseteq \mathsf{IS}_{12}$, $\mathsf{IC} = \mathsf{IS}_{10}$, $\mathsf{IC} = \mathsf{IS}^n_{10}$ for some $n \geq 2$, $\mathsf{IC} = \mathsf{IS}_1$, or $\mathsf{IC} = \mathsf{IS}^n_1$ for some $n \geq 2$.*

Proof. The case when $\mathsf{IC} \subseteq \mathsf{IL}_2$ follows from Theorem 15 since $\mathsf{L}_2 = [x \oplus y \oplus z]$. For each case when C belongs to the infinite chains in Post's lattice, or if $\mathsf{IC} \subseteq \mathsf{IM}_2$, it is known that $\mathsf{IC} = \langle\Gamma\rangle_{\mathrm{fr}}$ for any base Γ of IC [22], which is sufficient since $\langle\Gamma\rangle_{\mathrm{fr}} \subseteq \langle\Gamma\rangle_{\exists!}$. ◀

We now move on to the more interesting cases, and begin with the case when $\mathrm{Pol}(\Gamma)$ is essentially unary, i.e., consists of essentially unary operations. This covers $\mathsf{I}_2, \mathsf{I}_0, \mathsf{I}_1, \mathsf{I}, \mathsf{N}_2, \mathsf{N}$ from Figure 1.

▶ **Theorem 17** (∗). *Let Γ be a Boolean constraint language such that $\mathrm{Pol}(\Gamma)$ is essentially unary. Then $\langle\Gamma\rangle$ is $\exists!$-covered.*

Next, we consider ID_2, consisting of all relations pp-definable by binary clauses.

▶ **Lemma 18** (∗). *ID_2 is $\exists!$-covered.*

We now tackle the cases when $\mathrm{Inv}(\{\wedge, 0, 1\}) \subseteq \mathsf{IC} \subseteq \mathrm{Inv}(\{\wedge\})$ (E, E_0, E_1, and E_2 in Figure 1). First, we describe the determined arguments of relations in E_0.

▶ **Lemma 19.** *Let $R \in \mathsf{IE}_0$ be an n-ary relation. If $i \in [n]$ is determined in R then either (1) there exists $i_1, \ldots, i_k \in [n]$ distinct from i such that $t[i] = t[i_1] \wedge \ldots \wedge t[i_k]$ for every $t \in R$, or (2) $t[i] = 0$ for every $t \in R$.*

Proof. Assume that $i \in [n]$ is determined in R. Let $R_1 = \{t_1, \ldots, t_m\} = \{t \in R \mid t[i] = 1\}$ and $R_0 = \{s_1, \ldots, s_{m'}\} = \{s \in R \mid t[i] = 0\}$. Note first that $R_0 = \emptyset$ cannot happen since R is preserved by 0, and if $R_1 = \emptyset$ then we end up in case (2). Hence, in the remainder of the proof we assume that R_0 and R_1 are both non-empty.

Consider the tuple $t_1 \wedge \ldots \wedge t_m = t$ (applied componentwise), and observe that $t \in \{t_1, \ldots, t_m\}$ since R is preserved by $\wedge$, and that $t[i] = 1$ since $t_1[i] = \ldots = t_m[i] = 1$.

Furthermore, if $t[j] = 1$ for some $j \in [n]$ then it must also be the case that $t_1[j] = \ldots = t_m[j] = 1$. Let $i_1, \ldots, i_l \in [n] \setminus \{i\}$ denote the set of indices such that $t[i_j] = 1$. Then $t'[i] = t'[i_1] \wedge \ldots \wedge t'[i_l]$ for every $t' \in R_1$, and we also claim that $s[i] = s[i_1] \wedge \ldots \wedge s[i_l]$ for every $s \in R_0$, thus ending up in case (1). Note that $l > 0$, as otherwise every argument distinct from i is constantly 0 in t, which is not consistent with the fact that $\mathbf{0}^n \in R_0$, since it contradicts the assumption that i is determined. Assume that there exists $s \in R_0$ such that $s[i] = 0 \neq s[i_1] \wedge \ldots \wedge s[i_l]$. Then, clearly, $s[i_1] = \ldots = s[i_l] = 1$. But then $t \wedge s \in R$ implies that i is not determined, since $\text{Proj}_{1,\ldots,i-1,i+1,\ldots,n}(t \wedge s) = \text{Proj}_{1,\ldots,i-1,i+1,\ldots,n}(t)$ but $(t \wedge s)[i] \neq t[i]$. Hence, $s[i] = s[i_1] \wedge \ldots \wedge s[i_l]$ for every $s \in R$, which concludes the proof. ◀

Lemma 19 also shows that if $R \in \mathsf{IE}$ with a determined argument i then there exists $i_1, \ldots, i_k \in [\text{ar}(R)]$ such that $t[i] = t[i_1] \wedge \ldots \wedge t[i_k]$ for every $t \in R$, since the constant relation $\{(0)\} \notin \mathsf{IE}$. Before we use Lemma 19 to show the non-covering results for IE and IE_0, we will need the following lemma, relating the existence of a upp-definition to a qfpp-definition of a special form. The proof essentially follows directly from the statement of the lemma and is therefore omitted.

▶ **Lemma 20.** *Let Γ be a constraint language. Then an n-ary relation $R \in \langle \Gamma \rangle_{\exists!}$ has a upp-definition $R(x_1, \ldots, x_n) \equiv \exists! y_1, \ldots, y_m \colon \varphi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ if and only if there exists an $(n+m)$-ary relation $R' \in \langle \Gamma \rangle_{\nexists}$ such that $\text{Proj}_{1,\ldots,n}(R') = R$ where each $n < i \leq n+m$ is determined by $1, \ldots, n$.*

Say that a partial operation f is *$\wedge$-closed* if $\text{dom}(f)$ is preserved by $\wedge$ and that it is *0-closed* if $\mathbf{0}^{\text{ar}(f)} \in \text{dom}(f)$. We may now describe partial polymorphisms of $\langle \Gamma \rangle_{\exists!}$ using $\wedge$-closed and 0-closed partial polymorphisms of Γ.

▶ **Lemma 21 (∗).** *Let Γ be a constraint language such that $\langle \Gamma \rangle = \mathsf{IE}_0$. If $f \in \text{pPol}(\Gamma)$ is $\wedge$- and 0-closed then $f \in \text{pPol}(\langle \Gamma \rangle_{\exists!})$.*

We now have all the machinery in place to prove that IE_0 and IE are not $\exists!$-covered.

▶ **Theorem 22.** *Let R_w be the weak base of IE_0 from Table 2. Then $\langle R_w \rangle_{\exists!} \subset \mathsf{IE}_0$.*

Proof. We prove that the relation $R(x_1, x_2, x_3) \equiv x_1 \leftrightarrow x_2 x_3$ is not upp-definable over R_w, which is sufficient since $R \in \mathsf{IE}_0$, as evident in Table 2. Furthermore, using Lemma 20, we only have to prove that any $(3+n)$-ary R' where $\text{Proj}_{1,2,3}(R') = R$, and where each other argument is determined by the three first, is not included in $\langle R_w \rangle_{\nexists}$. Assume, without loss of generality, that R' does not contain any redundant arguments. Define the binary partial operation f such that $f(0,0) = 0$, $f(0,1) = f(1,0) = 1$. By construction, f is both 0-closed and $\wedge$-closed, and it is also readily verified that f preserves R_w, which via Lemma 21 then implies that $f \in \text{pPol}(\langle R_w \rangle_{\exists!})$. To finish the proof we also need to show that $f \notin \text{pPol}(R')$, which is sufficient since it implies that $R' \notin \langle R_w \rangle_{\exists!}$. Take two tuples $s, t \in R'$ such that $\text{Proj}_{1,2,3}(s) = (0,0,1)$, and $\text{Proj}_{1,2,3}(t) = (0,1,0)$. From Lemma 19, for each $3 < i \leq n+3$, either i is constant 0 in R' or there exists $i_1, \ldots, i_k \in \{1,2,3\}$, $k \leq 3$, such that $t[i] = t[i_1] \wedge \ldots t[i_k]$ for each $t \in R'$. But then $(s[i], t[i]) \in \text{dom}(f)$ for each $3 < i \leq n+3$, since either $(s[i], t[i]) = (0,0) \in \text{dom}(f)$ or $(s[i], t[i])$ is a conjunction over $(0,0,1)$ and $(0,1,0)$. However, this implies that $f(s,t) = u \notin R'$ since $\text{Proj}_{1,2,3}(u) = (0,1,1)$. Hence, f does not preserve R', and $R' \notin \langle R_w \rangle_{\nexists}$ via Theorem 4. ◀

The proof for IE uses the same construction and we omit the details. Surprisingly, as we will now see, IE_1 and IE_2 behave entirely differently and are in fact $\exists!$-covered.

▶ **Lemma 23 (∗).** IE_1 *and* IE_2 *are $\exists!$-covered.*

The natural generalisation of the Boolean operations $\wedge$ and $\vee$ are so-called *semilattice operations*; binary operations that are idempotent, associative, and commutative. It is then tempting to conjecture that Lemma 19 can be generalized to arbitrary semilattice operations, i.e., that every determined argument can be described as a semilattice combination of other arguments, whenever a relation is preserved by a given semilattice operation. This, however, is not true. For a simple counterexample define the semilattice operation $s\colon \{0,1,2\}^2 \to \{0,1,2\}$ as $s(x,x) = x$ and $s(x,y) = 0$ otherwise. If we then consider the relation $R = \{(0,0),(1,1),(2,0)\}$ it is easily verified that s preserves R, and that the second argument is uniquely determined by the first argument but cannot be described via the operation s.

The only co-clones remaining are IS_{11} and IS_{11}^n (for $n \geq 2$). As we will see, unique existential quantification is only as powerful as frozen quantification for these co-clones. We state the following lemma only for IS_{11} but the same construction is valid also for IS_{11}^n.

▶ **Lemma 24** (∗). *Let Γ be a constraint language such that $\langle\Gamma\rangle = \mathsf{IS}_{11}$. Then $\langle\Gamma\rangle_{\exists!} = \langle\Gamma\rangle_{\mathrm{fr}}$.*

It thus only remains to prove that IS_{11} and IS_{11}^n do not collapse into a single frozen co-clone. Here, we state the lemma only for IS_{11}^n, but the same argument works for IS_{11}.

▶ **Lemma 25** (∗). *Let Γ_p denote the plain base and Γ_w the weak base of IS_{11}^n ($n \geq 2$) from Table 2. Then $\langle\Gamma_w\rangle_{\mathrm{fr}} \subset \langle\Gamma_p\rangle_{\mathrm{fr}}$.*

Combining the results in this section we can now finally prove our dichotomy theorem.

▶ **Theorem 26.** *Let $\langle\Gamma\rangle$ be a Boolean co-clone. Then $\langle\Gamma\rangle$ is not $\exists!$-covered if and only if*

1. $\langle\Gamma\rangle \in \{\mathsf{IE},\mathsf{IE}_0,\mathsf{IV},\mathsf{IV}_1\}$, *or*
2. $\langle\Gamma\rangle \in \{\mathsf{IS}_{01}^n,\mathsf{IS}_{11}^n \mid n \geq 2\} \cup \{\mathsf{IS}_{01},\mathsf{IS}_{11}\}$ *(where, in addition, $\langle\Gamma\rangle_{\exists!} = \langle\Gamma\rangle_{\mathrm{fr}}$).*

Proof. Each negative case either follows immediately from Lemma 22, Lemma 24, Lemma 25, or is the dual of one of those cases. Each $\exists!$-covered co-clone is proved in Lemma 16, Theorem 17, Lemma 18, and Lemma 23. ◀

4 Applications in Complexity

In this section we apply Theorem 26 to study the complexity of computational problems not compatible with pp-definitions. Let us begin by defining the *constraint satisfaction problem* over a constraint language Γ (CSP(Γ)).

INSTANCE: A tuple (V,C) where V is a set of variables and C a set of constraints of the form $R_i(x_{i_1},\ldots,x_{i_{\mathrm{ar}(R)}})$ for $R_i \in \Gamma$.
QUESTION: Does (V,C) have at least one model? That is, a function $f\colon V \to D$ such that $f(x_{i_1},\ldots,x_{i_{\mathrm{ar}(R_i)}}) \in R_i$ for each $R_i(x_{i_1},\ldots,x_{i_{\mathrm{ar}(R_i)}}) \in C$?

For Boolean constraint languages Γ we write SAT(Γ) instead of CSP(Γ). If $\Delta \subseteq \langle\Gamma\rangle$ (or, equivalently, $\mathrm{Pol}(\Gamma) \subseteq \mathrm{Pol}(\Delta)$) then CSP($\Delta$) is polynomial-time reducible to CSP(Γ) [13]. However, there exist many natural variants of CSPs not compatible with pp-definitions, but compatible with more restricted closure operators such as upp-definitions. One such example is the *unique satisfiability problem* over a Boolean constraint language Γ (U-SAT(Γ)).

Instance: A SAT(Γ) instance I.
Question: Does I have a unique model?

The unrestricted U-SAT problem, i.e., the U-SAT problem where all possible constraints are allowed, can be seen as the intersection of satisfiability (in NP), and the satisfiability problem of checking if a given instance does not admit two distinct models (in co-NP). Hence, U-SAT is included in the second level of the Boolean hierarchy, BH_2, but is not believed to be complete for this class [23]. This unclear status motivated Blass and Gurevich [2] to introduce the complexity class *unique polynomial-time*, US, the set of decision problems solvable by a non-deterministic polynomial-time Turing machine where an instance is a yes-instance if and only if there exists a unique accepting path. Blass and Gurevich then quickly observed that U-SAT is US-complete and that $\mathrm{US} \subseteq \mathrm{BH}_2$.

We will present a simple, algebraic proof of Juban's trichotomy theorem for U-SAT(Γ) [15], showing that U-SAT(Γ) for finite Γ is either tractable, co-NP-complete, or US-complete. Using our machinery we will also be able to generalise this result to arbitrary infinite constraint languages. However, for infinite Γ we first need to specify a method of representation. We assume that the elements $R_1, R_2, \ldots$ of Γ are recursively enumerable by their arity, are represented as lists of tuples, and that there exists a computable function $f\colon \mathbb{N} \to \mathbb{N}$ such that for every $k \geq 1$ and every k-ary relation R, $R \in \langle \Gamma \rangle_{\exists !}$ if and only if $R \in \langle \Gamma \cap \mathsf{REL}^{\leq f(k)}_{\{0,1\}} \rangle_{\exists !}$. Thus, if a relation is upp-definable it is always possible to bound the arities of the required relations in the definition. The complexity of U-SAT(Γ) is then determined by $\langle \Gamma \rangle_{\exists !}$ in the following sense.

▶ **Theorem 27.** *Let Γ and Δ be Boolean constraint languages. If $\Delta \subseteq \langle \Gamma \rangle_{\exists !}$ is finite then* U-SAT(Δ) *is polynomial-time many-one reducible to* U-SAT(Γ).

Proof. By assumption every $R \in \Delta$ is upp-definable over Γ. First let $k = \max\{f(\mathrm{ar}(R)) \mid R \in \Delta\}$. We then begin by computing a upp-definition of R over $\Gamma \cap \mathsf{REL}^{\leq k}_{\{0,1\}}$, and store this upp-definition in a table. Since Δ is finite this can be done in constant time. Next, given an instance $I = (V, C)$ of U-SAT(Δ), we similar to the ordinary CSP case simply replace each constraint in C by its upp-definition over Γ, and identify any potential variables occurring in equality constraints. This procedure might introduce additional variables, but since they are all determined by V, the existence of a unique model is preserved. ◀

▶ **Theorem 28 (∗).** *Let Γ be a Boolean constraint language. Then* U-SAT(Γ) *is co-NP-complete if $\langle \Gamma \rangle \in \{\mathsf{II}_0, \mathsf{II}_1\}$, US-complete if $\langle \Gamma \rangle = \mathsf{II}_2$, and is tractable otherwise.*

A complexity classification akin to Theorem 28 is useful since it clearly separates tractable from intractable cases. However, in the last decade, a significant amount of research has been devoted to better understanding the "fine-grained" complexity of intractable problems, with a particular focus on ruling out algorithms running in $O(c^{|V|})$ time for every $c > 1$, so-called *subexponential time.* This line of research originates from Impagliazzo et al. [12] who conjectured that 3-SAT is not solvable in subexponential time; a conjecture known as the *exponential-time hypothesis* (ETH). Lower bounds for U-SAT(Γ) can then be proven using the ETH and the results from Section 3.

▶ **Theorem 29 (∗).** *Let Γ be a Boolean constraint language such that* U-SAT(Γ) *is US-complete or co-NP-complete. Then* U-SAT(Γ) *is not solvable in subexponential time, unless the ETH is false.*

Using our algebraic framework, hardness results can effortlessly be proven for the CSP generalisation of U-SAT, i.e., the problem U-CSP(Γ) of answering yes if and only if the given instance of CSP(Γ) admits a unique model.

▶ **Theorem 30** (∗). *Let Γ be a constraint language over a finite domain D. If $\langle \Gamma \rangle = \mathsf{REL}_D$ then* U-CSP(Γ) *is US-complete, and if* $\mathrm{Pol}(\Gamma) = [\{f\}]$ *for a constant operation f, then* U-CSP(Γ) *is co-NP-complete.*

5 Concluding Remarks and Future Research

We have studied unique existential quantification in pp-definitions, with a particular focus on finding constraint languages where existential quantification and unique existential quantification coincide. In general, this question appears highly challenging, but we have managed to find several broad classes of languages where this is true, and established a complete dichotomy theorem in the Boolean domain. We also demonstrated that upp-definitions can be applied to obtain complexity theorems for problems in a more systematic manner than what has earlier been possible. Many interesting open question hinge on the possibility of finding an algebraic characterisation of upp-closed sets of relations. For example, it would be interesting to determine the cardinality of the set $\{\langle \Gamma \rangle_{\exists !} \mid \Gamma \subseteq \mathsf{II}_2\}$, and hopefully describe all such upp-closed sets. By our classification theorem it suffices to investigate the Boolean co-clones that are not $\exists!$-covered, but even this question appears difficult to resolve using only relational tools. Similarly, a continued description of the $\exists!$-covered co-clones over finite domains would be greatly simplified by an algebraic characterisation. Thus, given a set of relations Γ, what is the correct notion of a "polymorphism" of a upp-definable relation over Γ? This question also has a strong practical motivation: essentially all complexity classifications for CSP related problems over non-Boolean domain require stronger algebraic tools than pp-definitions, and this is likely the case also for problems that can be studied with upp-definitions.

Another interesting topic is the following computational problem concerning upp-definability. Fix a constraint language Γ, and let R be a relation. Is it the case that R is upp-definable over Γ? The corresponding problem for pp-definitions is tractable for Boolean constraint languages Γ [9] while the corresponding problem for qfpp-definitions is co-NP-complete [16, 20]. Note that if $\langle \Gamma \rangle$ is $\exists!$-covered (which can be checked in polynomial time) then $R \in \langle \Gamma \rangle_{\exists !}$ can be answered by checking whether $R \in \langle \Gamma \rangle$. Thus, only the co-clones that are not $\exists!$-covered would need to be investigated in greater detail.

Last, it is worth remarking that our notion of uniqueness quantification in pp-definitions is not the only one possible. Assume that we in $\exists ! x_i \colon R(x_1, \ldots, x_i, \ldots, x_n)$ over a domain D do not require that x_i is determined by $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ but instead simply obtain the relation $\{(d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_n) \mid \exists ! d_i \in D \text{ such that } (d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_n) \in R)\}$. This notion of unique existential quantification is in general *not* comparable to existential quantification, since if we e.g. let $R = \{(0,0), (0,1), (1,0)\}$ then $T(x) \equiv \exists ! y \colon R(y, x)$ even though $T \notin \langle R \rangle$, i.e., is not even pp-definable by R (where $T = \{(1)\}$). Thus, it would be interesting to determine the resulting closed classes of relations and see in which respect they differ from the ordinary co-clone lattice.

References

1 L. Barto, A. Krokhin, and R. Willard. Polymorphisms, and How to Use Them. In Andrei Krokhin and Stanislav Zivny, 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.

2 A. Blass and Y. Gurevich. On the unique satisfiability problem. *Information and Control*, 55(1):80–88, 1982.

3 V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. I. *Cybernetics*, 5:243–252, 1969.

4 V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. II. *Cybernetics*, 5:531–539, 1969.

5 E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean Blocks, Part I: Post's Lattice with Applications to Complexity Theory. *ACM SIGACT-Newsletter*, 34(4):38–52, 2003.

6 A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *Proceedings of the 58th Annual Symposium on Foundations of Computer Science (FOCS-2017)*. IEEE Computer Society, 2017.

7 N. Creignou and M. Hermann. Complexity of Generalized Satisfiability Counting Problems. *Information and Computation*, 125(1):1–12, 1996.

8 N. Creignou, S. Khanna, and M. Sudan. *Complexity classifications of Boolean constraint satisfaction problems.* SIAM Monographs on Discrete Mathematics and Applications, 2001.

9 N. Creignou, P. Kolaitis, and B. Zanuttini. Structure identification of Boolean relations and plain bases for co-clones. *Journal of Computer and System Sciences*, 74(7):1103–1115, November 2008.

10 N. Creignou and H. Vollmer. Boolean Constraint Satisfaction Problems: When Does Post's Lattice Help? In N. Creignou, P. G. Kolaitis, and H. Vollmer, editors, *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 3–37. Springer Berlin Heidelberg, 2008.

11 D. Geiger. Closed Systems of Functions and Predicates. *Pacific Journal of Mathematics*, 27(1):95–100, 1968.

12 R. Impagliazzo and R. Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

13 P. Jeavons. On The Algebraic Structure Of Combinatorial Problems. *Theoretical Computer Science*, 200:185–204, 1998.

14 P. Jeavons, D. Cohen, and M. Gyssens. A unifying framework for tractable constraints. In *Proceedings of the First International Conference in Principles and Practice of Constraint Programming (CP-1995)*, pages 276–291, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

15 L. Juban. Dichotomy Theorem for the Generalized Unique Satisfiability Problem. In *Proceedings of the 12th International Symposium of Fundamentals of Computation Theory (FCT-1999)*, volume 1684 of *Lecture Notes in Computer Science*, pages 327–337. Springer, 1999.

16 D. Kavvadias and M. Sideri. The Inverse Satisfiability Problem. *SIAM Journal on Computing*, 28:152–163, 1998.

17 V. Lagerkvist. Weak bases of Boolean co-clones. *Information Processing Letters*, 114(9):462–468, 2014.

18 V. Lagerkvist. *Strong Partial Clones and the Complexity of Constraint Satisfaction Problems: Limitations and Applications.* PhD thesis, Linköping University, The Institute of Technology, 2016.

19 V. Lagerkvist and G. Nordh. On the Strength of Uniqueness Quantification in Primitive Positive Formulas. *ArXiv e-prints*, June 2019. `arXiv:1906.07031`.

20 V. Lagerkvist and B. Roy. A Dichotomy Theorem for the Inverse Satisfiability Problem. In *Proceedings of the 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS-2017)*, volume 93, pages 39:39–39:14, 2018.

21 D. Lau. *Function Algebras on Finite Sets: Basic Course on Many-Valued Logic and Clone Theory (Springer Monographs in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

22 G. Nordh and B. Zanuttini. Frozen Boolean Partial Co-clones. In *Proceedings of the 39th International Symposium on Multiple-Valued Logic (ISMVL-2009)*, pages 120–125, 2009. doi:10.1109/ISMVL.2009.10.

23 C.H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259, 1984.

24 E. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, 5:1–122, 1941.

25 S. Reith and K. W. Wagner. The Complexity of Problems Defined by Boolean Circuits. In *Proceedings International Conference Mathematical Foundation of Informatics (MFI-1999)*, pages 25–28, 1999.

26 B.A. Romov. The algebras of partial functions and their invariants. *Cybernetics*, 17(2):157–167, 1981.

27 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.

28 D. Zhuk. The Proof of CSP Dichotomy Conjecture. In *Proceedings of the 58th Annual Symposium on Foundations of Computer Science (FOCS-2017)*. IEEE Computer Society, 2017.

Resolution Lower Bounds for Refutation Statements

Michal Garlík

Dept. Ciències de la Computació, Universitat Politècnica de Catalunya,
C. Jordi Girona, 1-3, 08034 Barcelona, Spain
mgarlik@cs.upc.edu

Abstract

For any unsatisfiable CNF formula we give an exponential lower bound on the size of resolution refutations of a propositional statement that the formula has a resolution refutation. We describe three applications. (1) An open question in [2] asks whether a certain natural propositional encoding of the above statement is hard for Resolution. We answer by giving an exponential size lower bound. (2) We show exponential resolution size lower bounds for reflection principles, thereby improving a result in [1]. (3) We provide new examples of CNFs that exponentially separate Res(2) from Resolution (an exponential separation of these two proof systems was originally proved in [10]).

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases reflection principles, refutation statements, Resolution, proof complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.37

Related Version `https://arxiv.org/abs/1905.12372v1`

Funding Funded by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement ERC-2014-CoG 648276 (AUTAR).

1 Introduction

Proving lower bounds on the size of propositional proofs is the central task of proof complexity theory. After Cook and Reckhow [4] motivated this line of research as an approach towards establishing NP $\neq$ coNP, some initial success for weak proof systems followed, e.g., the first exponential size lower bound for Resolution was proved by Haken [6]. Nevertheless, many important open problems from the 1980s and 1990s remain unsolved, and it seems that proving nontrivial lower bounds on the size of propositional proofs is hard. If it is hard for people, it is natural to ask if it is also hard for the proof systems themselves. In trying to formalize this question so that it makes sense to a proof system, we must say what we mean by "proving is hard". It can be "there are no short proofs", a statement which appears in a propositional formalization of reflection principles. By "short" we mean polynomial in the size of the formula being proven or refuted. The negation of the *reflection principle* for a proof system P is a conjunction of the statement "y is a P-refutation of length s of formula x of length n" and the statement "z is a satisfying assignment of formula x". In a propositional formulation of the principle, P, s, n are fixed parameters and x, y, z are disjoint sets of variables. A possible way to formalize the above question is then to take the first conjunct of the negation of the reflection principle and plug in for the x-variables some formula F of length n. The resulting formula was discussed and utilized by Pudlák [9]; we denote it by $\mathrm{REF}^F_{P,s}$ and call it a *refutation statement* for P. We may now ask whether some proof system Q can shortly refute $\mathrm{REF}^F_{P,s}$, and if it can not, we can interpret this to mean that lower bounds for P-refutations of F are hard for Q.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 37; pp. 37:1–37:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Pudlák [9] found connections between the reflection principles and automatizability, and these were elaborated on in [1]. Following [3], a proof system P is *automatizable* if there is a deterministic algorithm that when given as input an unsatisfiable CNF formula F outputs a P-refutation of F in time polynomial in the size of the shortest P-refutation of F. Recently, Atserias and Müller [2] showed that Resolution is not automatizable unless P = NP. Refutation statements for Resolution play a prominent role in their proof. They show that strong enough resolution size lower bounds for $\mathrm{REF}^F_{\mathrm{Res},s}$ with an unsatisfiable F imply their result. However, they leave the lower bound problem for $\mathrm{REF}^F_{\mathrm{Res},s}$ as an open question, and in place of $\mathrm{REF}^F_{\mathrm{Res},s}$ they use in the proof a different formulation of the refutation statement, obtained by a *relativization* of $\mathrm{REF}^F_{\mathrm{Res},s}$, for which lower bounds are easier to get. In this paper we focus mainly on giving an answer to the question.

1.1 Results in This Paper

The result that requires the most work is the following lower bound.

▶ **Theorem 1.** *For each $\epsilon > 0$ there is $\delta > 0$ and an integer t_0 such that if n, r, s, t are integers satisfying $t \geq s \geq n + 1$, $r \geq n \geq 2$, $t \geq r^{3+\epsilon}$, $t \geq t_0$, and F is an unsatisfiable CNF consisting of r clauses $C_1, \dots, C_r$ in n variables $x_1, \dots, x_n$, then any resolution refutation of $\mathrm{REF}^F_{s,t}$ has length greater than 2^{t^δ}.*

We then show that this theorem implies an exponential resolution size lower bound for the encoding of the refutation statement for which the lower bound question in [2] is originally asked.

The formula $\mathrm{REF}^F_{s,t}$ in the theorem is a variant of the refutation statement insisting that the resolution refutation it describes has the form of a levelled graph. A similar simplifying assumption, making it more practical to design random restrictions, is used in [11] for a propositional version of the coloured polynomial local search principle. Our proof proceeds with defining a random restriction tailored to $\mathrm{REF}^F_{s,t}$ and to an adversary argument. The nature of the refutation statement and the fact that the relations between refutation lines are encoded in unary, rather than in binary, necessitate a more complicated adversary argument than in [8] or [11], and this in turn poses more requirements on the random restriction. We discuss these details after the proof, in Remarks 20 and 21.

We see two reasons for working with the unary encoding of $\mathrm{REF}^F_{s,t}$. First, $\mathrm{REF}^F_{s,t}$ is weaker than refutation statements encoded in binary or relativized refutation statements. Hence lower bounds for $\mathrm{REF}^F_{s,t}$ imply lower bounds for the other encodings. Second, researchers who dealt with propositional encodings of reflection principles or refutation statements opted for the unary encoding [1, 7, 9].

Besides these two reasons, we need to work with the unary encoding to give an answer to the above mentioned lower bound question from [2]. Our answer is stated in the following theorem, the proof of which (an easy reduction to Theorem 1) is in the full version of this paper [5]. Below, $\mathrm{REF}(F, \tilde{s})$ denotes the encoding of the resolution refutation statement in [2]. We can assume that the formula $\mathrm{REF}^F_{\mathrm{Res},s}$ is the same as $\mathrm{REF}(F, s)$.

▶ **Theorem 2.** *For each $\epsilon > 0$ there is $\delta > 0$ and an integer t_0 such that if $n, r, \tilde{s}$ are integers satisfying $r \geq n \geq 2$, $\left\lfloor \frac{\tilde{s}}{n+1} \right\rfloor \geq r^{3+\epsilon}$, $\left\lfloor \frac{\tilde{s}}{n+1} \right\rfloor \geq t_0$, and F is an unsatisfiable CNF consisting of r clauses $C_1, \dots, C_r$ in n variables $x_1, \dots, x_n$, then any resolution refutation of $\mathrm{REF}(F, \tilde{s})$ has length greater than $2^{\lfloor \frac{\tilde{s}}{n+1} \rfloor^\delta}$.*

Our next result is that the negation of the reflection principle for Resolution, expressed by the formula $\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{s,t}$, exponentially separates the system Res(2) from Resolution. It was shown by Atserias and Bonet [1] that a similar encoding of the negation of the reflection principle separates the two theories almost-exponentially (giving a $2^{\Omega(2^{\log^\epsilon n})}$ resolution lower bound and a polynomial Res(2) upper bound). The exponential separation of Res(2) from Resolution was originally proved in [10] using a variation of the graph ordering principle. Our lower bound is stated in Theorem 3 below.

▶ **Theorem 3.** *For every $c > 4$ there is $\delta > 0$ and an integer n_0 such that if n, r, s, t are integers satisfying $t \geq s \geq n + 1$, $r \geq n \geq n_0$, $n^c \geq t \geq r^4$, then any resolution refutation of* $\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{s,t}$ *has length greater than 2^{n^δ}.*

The proof of the theorem also yields new examples of CNFs exponentially separating Res(2) from Resolution.

▶ **Theorem 4.** *Let $\delta_1 > 0$ and let $\{A_n\}_{n\geq 1}$ be a family of unsatisfiable CNFs such that A_n is in n variables, has the number of clauses polynomial in n, and has no resolution refutations of length at most $2^{n^{\delta_1}}$. Then there is $\delta > 0$ and a polynomial p such that $A_n \wedge \mathrm{REF}^{A_n}_{n+1,p(n)}$ has no resolution refutations of length at most 2^{n^δ} and has polynomial size* Res(2) *refutations.*

A Res(2) upper bound for $\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{s,t}$, needed for completing the separation by this formula as well as by the formulas in Theorem 4, is stated in the following theorem.

▶ **Theorem 5.** *The negation of the reflection principle for Resolution expressed by the formula* $\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{s,t}$ *has Res*(2) *refutations of size $O(trn^2 + tr^2 + st^2n^3 + st^3n)$.*

A polynomial size Res(2) upper bound on a similar encoding of the negation of the reflection principle for Resolution was proved in [1]. We simplify the proof and adapt it to $\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{s,t}$ (see the full version [5]).

1.2 Outline of This Paper

The rest of the paper is organized as follows.

In Section 2 we give the necessary preliminaries.

In Section 3, Resolution of s levels of t clauses is introduced, and the clauses of the refutation statement $\mathrm{REF}^F_{s,t}$ for this refutation system are listed. We also state here a quadratic simulation of Resolution by this system.

In Section 4 we prove Theorem 1.

In Section 5 we define the formula $\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{s,t}$ and we prove Theorems 3 and 4.

2 Preliminaries

For an integer s, the set $\{1, \ldots, s\}$ is denoted by $[s]$. We write $\mathrm{dom}(f), \mathrm{im}(f)$ for the domain and image of a funciton f. If x is a propositional variable, the *positive literal of* x, denoted by x^1, is x, and the *negative literal of* x, denoted by x^0, is $\neg x$. A *clause* is a set of literals. A clause is written as a disjunction of its elements. A *term* is a set of literals, and is written as a conjunction of the literals. A *CNF* is a set of clauses, written as a conjunction of the clauses. A k*-CNF* is a CNF whose every clause has at most k literals. A *DNF* is a set of terms, written as a disjunction of the terms. A k*-DNF* is a DNF whose every term has at most k literals. We will identify 1-DNFs with clauses. A clause is *non-tautological* if it does not contain both the positive and negative literal of the same variable. A clause C is a *weakening* of a clause D if $D \subseteq C$. A clause D is the *resolvent of* clauses C_1 and C_2 *on* a variable x if $x \in C_1, \neg x \in C_2$ and $D = (C_1 \setminus \{x\}) \cup (C_2 \setminus \{\neg x\})$. If E is a weakening of the resolvent of C_1 and C_2 on x, we say that E is obtained by the *resolution rule* from C_1 and C_2, and we call C_1 and C_2 the *premises* of the rule.

Let F be a CNF and C a clause. A *resolution derivation of C from F* is a sequence of clauses $\Pi = (C_1, \dots, C_s)$ such that $C_s = C$ and for all $u \in [s]$, C_u is a weakening of a clause in F, or there are $v, w \in [u-1]$ such that C_u is obtained by the resolution rule from C_v and C_w. The *length* of the derivation Π is s. For $u \in [s]$, the *height of u in* Π is the maximum h such that there is a subsequence $(C_{u_1}, \dots, C_{u_h})$ of Π in which $u_h = u$ and for each $i \in [h-1]$, C_{u_i} is a premise of a resolution rule by which $C_{u_{i+1}}$ is obtained in Π. The *height* of Π is the maximum height of u in Π for $u \in [s]$. A *resolution refutation of F* is a resolution derivation of the empty set from F.

A *partial assignment* to the variables $x_1, \dots, x_n$ is a partial map from $\{x_1, \dots, x_n\}$ to $\{0,1\}$. Let σ be a partial assignment. The CNF $F \restriction \sigma$ is formed from F by removing every clause containing a literal satisfied by σ, and removing every literal falsified by σ from the remaining clauses. If $\Pi = (C_1, \dots, C_s)$ is a sequence of clauses, $\Pi \restriction \sigma$ is formed from Π by the same operations. Note that if Π is a resolution refutation of F, then $\Pi \restriction \sigma$ is a resolution refutation of $F \restriction \sigma$.

The Res(k) refutation system is a generalization of Resolution. Its lines are k-DNFs and it has the following inference rules (A, B are k-DNFs, $j \in [k]$, and $l, l_1, \dots, l_j$ are literals):

$$\frac{A \vee l_1 \qquad B \vee (l_2 \wedge \dots \wedge l_j)}{A \vee B \vee (l_1 \wedge \dots \wedge l_j)}\ \wedge\text{-introduction} \qquad\qquad \frac{}{x \vee \neg x}\ \text{Axiom}$$

$$\frac{A \vee (l_1 \wedge \dots \wedge l_j) \qquad B \vee \neg l_1 \vee \dots \vee \neg l_j}{A \vee B}\ \text{Cut} \qquad\qquad \frac{A}{A \vee B}\ \text{Weakening}$$

Let F be a CNF. A *Res(k) derivation from F* is a sequence of k-DNFs $(D_1, \dots, D_s)$ so that each D_i either belongs to F or follows from the preceding lines by an application of one of the inference rules. The *size* of a Res(k) derivation is the number of symbols in it.

3 Resolution Refutations of s Levels of t Clauses

We introduce a variant of Resolution in which the clauses forming a refutation are arranged in layers.

▶ **Definition 6.** Let F be a CNF of r clauses in n variables $x_1, \dots, x_n$. We say that F has a *resolution refutation of s levels of t clauses* if there is a sequence of clauses $C_{i,j}$ indexed by all pairs $(i,j) \in [s] \times [t]$, such that each clause $C_{1,j}$ on the first level is a weakening of a clause in F, each clause $C_{i,j}$ on level $i \in \{2, \dots, s\}$ is a weakening of the resolvent of two clauses from level $i-1$ on a variable, and the clause $C_{s,t}$ is empty.

The following proposition shows that this system quadratically simulates Resolution and preserves the refutation height. The proof (in the full version [5] of the paper) uses a simple self-replicating pattern both to transport a premise of the resolution rule to the required level and to fill in all clauses $C_{i,j}$ that do not directly participate in the simulation.

▶ **Proposition 7.** *If a $(n-1)$-CNF F in n variables has a resolution refutation of height h and length s, then F has a resolution refutation of h levels of $3s$ clauses.*

We proceed to our formalization of the refutation statement for this refutation system. Let n, r, s, t be integers. Let F be a CNF consisting of r clauses $C_1, \dots, C_r$ in n variables $x_1, \dots, x_n$. We define a propositional formula $\mathrm{REF}^F_{s,t}$ expressing that F has a resolution refutation of s levels of t clauses.

We first list the variables of $\mathrm{REF}^F_{s,t}$. *D-variables* $D(i,j,k,b)$, $i \in [s]$, $j \in [t], k \in [n], b \in \{0,1\}$, encode clauses $C_{i,j}$ as follows: $D(i,j,k,1)$ (resp. $D(i,j,k,0)$) means that the literal x_k (resp. $\neg x_k$) is in $C_{i,j}$. *L-variables* $L(i,j,j')$ (resp. *R-variables* $R(i,j,j')$),

$i \in \{2,\dots,s\}, j, j' \in [t]$, say that $C_{i-1,j'}$ is a premise of the resolution rule by which $C_{i,j}$ is obtained, and it is the premise containing the positive (resp. negative) literal of the resolved variable. *V-variables* $V(i,j,k)$, $i \in \{2,\dots,s\}, j \in [t], k \in [n]$, say that $C_{i,j}$ is obtained by resolving on x_k. *I-variables* $I(j,m)$, $j \in [t], m \in [r]$, say that $C_{1,j}$ is a weakening of C_m.

$\mathrm{REF}^F_{s,t}$ is the union of the following fifteen sets of clauses:

$$\neg I(j,m) \vee D(1,j,k,b) \qquad j \in [t], m \in [r], b \in \{0,1\}, x_k^b \in C_m, \tag{1}$$

clause $C_{1,j}$ contains the literals of C_m assigned to it by $I(j,m)$,

$$\neg D(i,j,k,1) \vee \neg D(i,j,k,0) \qquad i \in [s], j \in [t], k \in [n], \tag{2}$$

no clause $C_{i,j}$ contains x_k and $\neg x_k$ at the same time,

$$\neg L(i,j,j') \vee \neg V(i,j,k) \vee D(i-1,j',k,1) \qquad i \in \{2,\dots,s\}, j, j' \in [t], k \in [n], \tag{3}$$

$$\neg R(i,j,j') \vee \neg V(i,j,k) \vee D(i-1,j',k,0) \qquad i \in \{2,\dots,s\}, j, j' \in [t], k \in [n], \tag{4}$$

clause $C_{i-1,j'}$ used as the premise given by $L(i,j,j')$ (resp. $R(i,j,j')$) in resolving on x_k must contain x_k (resp. $\neg x_k$),

$$\neg L(i,j,j') \vee \neg V(i,j,k) \vee \neg D(i-1,j',k',b) \vee D(i,j,k',b)$$
$$i \in \{2,\dots,s\}, j, j' \in [t], k, k' \in [n], b \in \{0,1\}, (k',b) \neq (k,1), \tag{5}$$

$$\neg R(i,j,j') \vee \neg V(i,j,k) \vee \neg D(i-1,j',k',b) \vee D(i,j,k',b)$$
$$i \in \{2,\dots,s\}, j, j' \in [t], k, k' \in [n], b \in \{0,1\}, (k',b) \neq (k,0), \tag{6}$$

clause $C_{i,j}$ derived by resolving on x_k must contain each literal different from x_k (resp. $\neg x_k$) from the premise given by $L(i,j,j')$ (resp. $R(i,j,j')$),

$$\neg D(s,t,k,b) \qquad k \in [n], b \in \{0,1\}, \tag{7}$$

clause $C_{s,t}$ is empty,

$$V(i,j,1) \vee V(i,j,2) \vee \ldots \vee V(i,j,n) \qquad i \in \{2,\dots,s\}, j \in [t], \tag{8}$$

$$I(j,1) \vee I(j,2) \vee \ldots \vee I(j,r) \qquad j \in [t], \tag{9}$$

$$L(i,j,1) \vee L(i,j,2) \vee \ldots \vee L(i,j,t) \qquad i \in \{2,\dots,s\}, j \in [t], \tag{10}$$

$$R(i,j,1) \vee R(i,j,2) \vee \ldots \vee R(i,j,t) \qquad i \in \{2,\dots,s\}, j \in [t], \tag{11}$$

$$\neg V(i,j,k) \vee \neg V(i,j,k') \qquad i \in \{2,\dots,s\}, j \in [t], k, k' \in [n], k \neq k', \tag{12}$$

$$\neg I(j,m) \vee \neg I(j,m') \qquad j \in [t], m, m' \in [r], m \neq m', \tag{13}$$

$$\neg L(i,j,j') \vee \neg L(i,j,j'') \qquad i \in \{2,\dots,s\}, j, j', j'' \in [t], j' \neq j'', \tag{14}$$

$$\neg R(i,j,j') \vee \neg R(i,j,j'') \qquad i \in \{2,\dots,s\}, j, j', j'' \in [t], j' \neq j'', \tag{15}$$

the V, I, L, R-variables define functions with the required domains and ranges.

4 A Lower Bound on Lengths of Resolution Refutations of $\mathrm{REF}^F_{s,t}$

We restate Theorem 1 from the Introduction.

▶ **Theorem 8.** *For each $\epsilon > 0$ there is $\delta > 0$ and an integer t_0 such that if n, r, s, t are integers satisfying*

$$t \geq s \geq n+1, \qquad r \geq n \geq 2, \qquad t \geq r^{3+\epsilon}, \qquad t \geq t_0, \tag{16}$$

and F is an unsatisfiable CNF consisting of r clauses $C_1, \dots, C_r$ in n variables $x_1, \dots, x_n$, then any resolution refutation of $\mathrm{REF}^F_{s,t}$ has length greater than 2^{t^δ}.

The rest of this section is devoted to a proof of the theorem. We argue by contradiction. Fix $\epsilon > 0$ and assume that for each $\delta > 0$ and t_0 there are integers n, r, s, t satisfying (16), an unsatisfiable CNF F, and a resolution refutation Π of $\mathrm{REF}^F_{s,t}$, such that F consists of r clauses $C_1, \ldots, C_r$ in n variables $x_1, \ldots, x_n$, and Π has length at most 2^{t^δ}.

The forthcoming distribution on partial assignments to the variables of $\mathrm{REF}^F_{s,t}$ employs in its definition and analysis two important parameters, p and w. We choose them as function of t and ϵ as follows:

$$p = t^{-a} \text{ with } a = \min\left\{\frac{2+\epsilon/2}{3+\epsilon/2}, \frac{3}{4}\right\}, \qquad w = t^{4/5}.$$

We now fix values of t_0, δ for which we will get the desired contradiction. Take t_0 so large and $\delta > 0$ so small that the inequalities

$$\max\left\{e^{-\frac{pw}{3}} + 2s \cdot e^{-\frac{pt}{3}}, e^{-\frac{pt}{8r}}\right\} \cdot 2^{t^\delta} + 3s \cdot e^{-\frac{pt}{3}} + 3p + 67p^3st < 1, \tag{17}$$

$$10pt + 4w < \frac{t}{4}, \tag{18}$$

$$e^{e^{\ln(t) - \frac{pt}{3}}} < 2, \tag{19}$$

hold for any n, r, s, t satisfying (16).

▶ **Definition 9.** For $i \in [s], j, j' \in [t], k \in [n], b \in \{0,1\}, m \in [r]$, we say that (i,j) is the *home pair* of the variable $D(i,j,k,b)$ (resp. $R(i,j,j')$; $L(i,j,j')$; $V(i,j,k')$; $I(j,m)$ if $i = 1$).

We write $V(i,j,\cdot)$ to stand for the set $\{V(i,j,k) : k \in [n]\}$. Similarly, we write $I(j,\cdot), L(i,j,\cdot), R(i,j,\cdot)$ to stand for the corresponding sets of variables, and we denote by $D(i,j,\cdot,\cdot)$ the set of variables $\{D(i,j,k,b) : k \in [n], b \in \{0,1\}\}$.

Let σ be a partial assignment. We say that $V(i,j,\cdot)$ is *set to* k *by* σ if $\sigma(V(i,j,k)) = 1$ and $\sigma(V(i,j,k')) = 0$ for all $k' \in [n], k' \neq k$. Similarly for $I(j,\cdot), L(i,j,\cdot), R(i,j,\cdot)$. We say that $D(i,j,\cdot,\cdot)$ is *set to* a clause $C_{i,j}$ *by* σ if for all $k \in [n], b \in \{0,1\}$, $\sigma(D(i,j,k,b)) = 1$ if $x_k^b \in C_{i,j}$ and $\sigma(D(i,j,k,b)) = 0$ if $x_k^b \notin C_{i,j}$.

For $Y \in \{D(i,j,\cdot,\cdot), V(i,j,\cdot), I(j,\cdot), R(i,j,\cdot), L(i,j,\cdot),\}$, we say that Y is *set by* σ if Y is set to v by σ for some value v. We will often omit saying "by σ" if σ is clear from the context.

▶ **Definition 10.** A *random restriction* ρ is a partial assignment to the variables of $\mathrm{REF}^F_{s,t}$ given by the following experiment:

1. For each pair $(i,j) \in [s] \times [t]$, with independent probability p include (i,j) in a set A_D. Then for each $(i,j) \in A_D$ and for each $k \in [n]$, independently, with probability $1/2$ choose between including the literal x_k or $\neg x_k$ in a clause $C_{i,j}$. Set $D(i,j,\cdot,\cdot)$ to $C_{i,j}$.
2. For each $j \in [t]$, with independent probability p include the pair $(1,j)$ in a set A_I. Then for each $(1,j) \in A_I \setminus A_D$, independently, choose at random $m \in [r]$ and set $I(j,\cdot)$ to m.
3. For each pair $(i,j) \in \{2,\ldots,s\} \times [t]$, with independent probability p include (i,j) in a set A_V. Then for each $(i,j) \in A_V$, independently, choose at random $k \in [n]$ and set $V(i,j,\cdot)$ to k.
4. For each pair $(i,j) \in \{2,\ldots,s\} \times [t]$, with independent probability p include the pair (i,j) in a set A_{RL}. Then, for each $i \in \{2,\ldots,s\}$, define $A_i := A_{RL} \cap (\{i\} \times [t])$ and do the following. If $|A_i| > 2pt$, define $h_i := \emptyset$, $B_{i-1} := \emptyset$. Otherwise, choose at random an injection h_i from $\{L(i,j,\cdot) : (i,j) \in A_i\} \cup \{R(i,j,\cdot) : (i,j) \in A_i\}$ to $[t]$. Define $B_{i-1} := \{(i-1,j) : j \in \mathrm{im}(h_i)\}$. Set $L(i,j,\cdot)$ to $h_i(L(i,j,\cdot))$ and $R(i,j,\cdot)$ to $h_i(R(i,j,\cdot))$ for all $(i,j) \in A_i$.

▶ **Lemma 11.** *With probability at least $1 - 3s \cdot e^{-pt/3}$, all of the following are satisfied.*

(i) *For each $i \in [s]$, the cardinality of $A_D \cap (\{i\} \times [t])$ is at most $2pt$.*

(ii) *For each $i \in \{2, \dots, s\}$, the cardinality of A_i is at most $2pt$ and the cardinality of $A_V \cap (\{i\} \times [t])$ is at most $2pt$.*

(iii) *The cardinality of A_I is at most $2pt$.*

Proof. By the Chernoff bound and the union bound it follows that item i is false with probability at most $s \cdot e^{-pt/3}$. Similarly for the remaining items. ◀

▶ **Definition 12.** Denote by G_ρ the graph with vertices $A_D \cup A_V \cup A_I \cup A_{RL} \cup \bigcup_{i\in[s-1]} B_i$, and with edges only between vertices on neighboring levels, such that (i, j) is connected by an edge to $(i-1, j')$ if and only if $h_i(L(i, j, \cdot)) = j'$ (then $(i-1, j')$ is called the *left child* of (i, j)) or $h_i(R(i, j, \cdot)) = j'$ (then $(i-1, j')$ is the *right child* of (i, j)).

The following lemma will be used later to show that a random restriction likely does not falsify any clause of $\mathrm{REF}^F_{s,t}$.

▶ **Lemma 13.** *With probability at least $1 - 3p - 67p^3st$, the following are satisfied.*

(i) *$(s, t) \notin (A_D \cup A_{RL} \cup A_V)$.*

(ii) *There is no triple $((i_1, j_1), (i_2, j_2), (i_3, j_3))$ of elements of $[s] \times [t]$, such that all the following hold:*

- **(a)** *For each $u \in [3]$ there is $X \in \{D, V, I, RL\}$ with $(i_u, j_u) \in A_X$,*
- **(b)** *$|\{(i_u, j_u, X) : u \in [3], X \in \{D, V, I, RL\}, (i_u, j_u) \in A_X\}| \geq 3$,*
- **(c)** *the subgraph of G_ρ consisting of the vertices that are in the triple and their children and all edges that go from a vertex of the triple to its children, is connected.*

Proof. The probability that item i is true is $(1 - p)^3 \geq 1 - 3p$.

Regarding item ii, we distinguish several cases based on the relative positions of the elements in a triple $((i_1, j_1), (i_2, j_2), (i_3, j_3))$. Note that the order in which the elements of the triple are listed does not matter in what we are proving, but some of the elements may coincide. When considering the cases, recall that due to our choice of the function h_i in the definition of ρ, two vertices in G_ρ cannot share a child.

In case all the elements of the triple are the same, iib is satisfied only if the element is chosen to A_X for three distinct values of X. This cannot happen on level 1, and on the other levels it happens with probability p^3. There are st many triples considered in the present case, so by the union bound the probability that there is any such triple satisfying all conditions in ii is at most p^3st.

In case $(i_1, j_1) \neq (i_2, j_2) = (i_3, j_3)$, condition iic is satisfied only if $i_1 = i_2 + 1$ or $i_2 = i_1 + 1$. In each of these two subcases, there are at most st^2 such triples. In the former subcase, we must have $(i_1, j_1) \in A_{RL}$ and at the same time $h_{i_1}(R(i_1, j_1, \cdot)) = j_2$ or $h_{i_1}(L(i_1, j_1, \cdot)) = j_2$. This happens with probability at most $2p/t$. Also, (i_2, j_2) has to be in A_X and $A_{X'}$ for distinct X, X', which happens with probability at most $3p^2$. So, the probability that any triple considered in this subcase satisfies iia - iic is at most $st^2 \cdot 6p^3/t = 6p^3st$. In the latter subcase, (i_2, j_2) has to be in A_{RL}, connected to (i_1, j_1), and additionally it has to be in A_D or A_V, while (i_1, j_1) has to be in arbitrary possible A_X. This happens with probability at most $2p/t \cdot 2p \cdot 3p = 12p^3/t$, so the probability that any such triple satisfies iia - iic is at most $12p^3st$.

In case all the elements of the triple are distinct, we again consider two subcases: first, $i_1 = i_2 + 1 = i_3 + 2$, and second, $i_1 - 1 = i_2 = i_3$. Each subcase concerns at most st^3 triples. In the first subcase, (i_3, j_3) has to be a child of (i_2, j_2), which in turn has to be a child of (i_1, j_1), and (i_3, j_3) also has to be in arbitrary possible A_X. This happens with probability at most $12p^3/t^2$. Hence the probability that any such triple satisfies iia - iic is at most $12p^3st$. In the second subcase, (i_1,j_1) has to have children (i_2,j_2) and (i_3,j_3), and each child has to be in some A_X for any suitable X. This happens with probability at most $2p/(t(t-1)) \cdot (3p)^2 = 18p^3/(t(t-1)) \leq 36p^3/t^2$. Hence the probability that any such triple satisfies iia - iic is at most $36p^3st$. ◀

We now define some specific ways to measure a clause and we use them in the next lemma to describe how a clause simplifies under a restriction.

▶ **Definition 14.** Let E be a clause in $\Pi{\restriction}\rho$, and let $(i,j) \in [s] \times [t]$. If E contains a literal of a variable from $D(i,j,\cdot,\cdot)$ (resp. $R(i,j,\cdot)$; $L(i,j,\cdot)$; $V(i,j,\cdot)$; $I(j,\cdot)$ and $i = 1$), we say that the pair (i,j) is *D-mentioned* (resp. *R-mentioned*; *L-mentioned*; *V-mentioned*; *I-mentioned*) *in E*.

We say that (i,j) is *V-important* (resp. *L-important*; *R-important*; *I-important*) *in E* if E contains the negative literal of a variable in $V(i,j,\cdot)$ (resp. $L(i,j,\cdot); R(i,j,\cdot); I(j,\cdot)$ and $i=1$) or if E contains at least $n/2$ (resp. $t/2$; $t/2$; $r/2$) positive literals of variables in $V(i,j,\cdot)$ (resp. $L(i,j,\cdot)$; $R(i,j,\cdot)$; $I(j,\cdot)$ and $i = 1$). A pair is *D-important in E* if it is D-mentioned in E.

▶ **Lemma 15.** *With probability at least* $1 - \max\left\{e^{-\frac{pw}{3}} + 2s \cdot e^{-\frac{pt}{3}}, e^{-\frac{pt}{8r}}\right\} \cdot 2^{t^\delta}$, *for every clause E in $\Pi{\restriction}\rho$ all of the following are satisfied.*

(i) *At most w many pairs (i,j) are D-mentioned in E.*
(ii) *At most w many pairs $(1,j)$ are I-important in E.*
(iii) *At most w many pairs (i,j) are V-important in E.*
(iv) *At most w many pairs (i,j) are L-important in E.*
(v) *At most w many pairs (i,j) are R-important in E.*
(vi) *For each $m \in [r]$, $|\{j : I(j,m) \in E\}| \leq \frac{t}{4}$.*
(vii) *For each $i \in \{s-n+1,\dots,s-1\}$ and $k \in [n]$, $|\{j : V(i,j,k) \in E\}| \leq \frac{t}{4}$.*

Proof. It is sufficient to prove that if E' is a clause in Π that violates any of i - vii, then with probability at least $1 - \max\left\{e^{-\frac{pw}{3}} + 2s \cdot e^{-\frac{pt}{3}}, e^{-\frac{pt}{8r}}\right\}$, E' is satisfied by ρ. Since Π has length at most 2^{t^δ}, the lemma then follows by the union bound.

Regarding item i, assume that E' in Π D-mentions more than w pairs (i,j). This means that a literal of a variable in $D(i,j,\cdot,\cdot)$ is in E' for more than w many pairs (i,j). For each such (i,j), such a literal is satisfied by ρ with probability at least $p/2$. So the probability that none of these literals in E' is satisfied is at most $(1-p/2)^w < e^{-pw/2}$.

Regarding item ii, suppose that more than w pairs $(1,j)$ are I-important in E'. For each such $(1,j)$, the probability that $(1,j) \in A_I \setminus A_D$ is $p(1-p)$, and provided this happens, the probability that ρ satisfies a literal in E' of a variable in $I(j,\cdot)$ is at least $\min\{(r-1)/r, 1/2\} = 1/2$. Hence the probability that E' is not satisfied by ρ is at most $(1-p(1-p)/2)^w < (1-p/3)^w < e^{-pw/3}$ (the first inequality follows from (18)).

Regarding item iii, a calculation similar to that for ii gives that a clause E' in Π with more than w many V-important pairs (i,j) is not satisfied by ρ with probability at most $(1-p/2)^w < e^{-pw/2}$.

Regarding item iv, suppose that more than w many pairs (i,j) from $\{2,\dots,s\} \times [t]$ are L-important in E'. For each $i \in \{2,\dots,s\}$, assume without loss of generality that the set of pairs (i,j) that are L-important in E' is the set $\{(i,1),\dots,(i,w_i)\}$; denote it by W_i. Note that the distribution of ρ does not change if we choose A_i and h_i in t many steps as follows. Start with $A_{i,0} = h_{i,0} = \emptyset$. At step $j = 1,2,\dots,t$, first add (i,j) to $A_{i,j-1}$ with probability p to get $A_{i,j}$. Then, if $|A_{i,j}| \leq 2pt$ and $(i,j) \in A_{i,j}$, choose at random two distinct elements j', j'' from $[t] \setminus \mathrm{im}(h_{i,j-1})$, and define $h_{i,j} := h_{i,j-1} \cup \{(L(i,j,\cdot), j'), (R(i,j,\cdot), j'')\}$. If $|A_{i,j}| \leq 2pt$ and $(i,j) \notin A_{i,j}$, define $h_{i,j} := h_{i,j-1}$. If $|A_{i,j}| > 2pt$ define $h_{i,j} := \emptyset$. This finishes step j. Finally, define $A_i := A_{i,t}$ and $h_i := h_{i,t}$.

For $i \in \{2, \dots, s\}$, let H_i be the set of literals in E' of a variable in $L(i,j,\cdot)$ for some $(i,j) \in W_i$. Also, for $(i,j) \in W_i$, let $T_{i,j}$ be the set of those $j' \in [t]$ such that the partial assignment given by setting $L(i,j,\cdot)$ to j' satisfies some literal in H_i. We know that $|T_{i,j}| \geq t/2$ for each $(i,j) \in W_i$.

The event that no literal in H_i is satisfied by ρ is a subset of the union of events (a) $|A_{i,t}| > 2pt$, and (b) $|A_{i,w_i}| \leq 2pt$ and for each $(i,j) \in A_{i,w_i}$, $h_{i,j}(L(i,j,\cdot)) \notin T_{i,j}$. Event (a) happens with probability at most $e^{-pt/3}$ by the Chernoff bound. We bound the probability of event (b). For each $j \in [w_i]$, if $(i,j) \in A_{i,j}$ and $|A_{i,j}| \leq 2pt$, then the probability that $h_{i,j}(L(i,j,\cdot)) \in T_{i,j}$ is at least $(|T_{i,j} \setminus \mathrm{im}(h_{i,j-1})|)/t \geq (t/2 - 4pt)/t = (1-8p)/2 \geq 1/3$ (the last inequality follows from (18)). Therefore, denoting $\ell := \min\{2pt, w_i\}$, the probability of event (b) is at most

$$\sum_{k=0}^{\ell} \binom{w_i}{k} p^k (1-p)^{w_i-k} \left(\frac{2}{3}\right)^k \leq \sum_{k=0}^{w_i} \binom{w_i}{k} \left(\frac{2p}{3}\right)^k (1-p)^{w_i-k} = (1-p/3)^{w_i}.$$

Thus, the probability that no literal in H_i is satisfied by ρ is at most $e^{-pt/3} + e^{-pw_i/3}$, and, denoting $S := \{i \in \{2, \dots, s\} : w_i \neq 0\}$, the probability that no literal in $\bigcup_{i \in S} H_i$ is satisfied by ρ is at most

$$\begin{aligned}\prod_{i \in S} \left(e^{-\frac{pw_i}{3}} + e^{-\frac{pt}{3}}\right) &\leq e^{-\frac{pw}{3}} + \sum_{k=1}^{|S|} \binom{|S|}{k} e^{-\frac{ptk}{3}} \\ &\leq e^{-\frac{pw}{3}} + |S| \cdot e^{-\frac{pt}{3}} \sum_{k=1}^{|S|} \binom{|S|-1}{k-1} e^{-\frac{pt(k-1)}{3}} \\ &= e^{-\frac{pw}{3}} + |S| \cdot e^{-\frac{pt}{3}} \cdot \left(1 + e^{-\frac{pt}{3}}\right)^{|S|-1} \\ &\leq e^{-\frac{pw}{3}} + s \cdot e^{-\frac{pt}{3}} \cdot e^{e^{\ln(t) - \frac{pt}{3}}} \leq e^{-\frac{pw}{3}} + 2s \cdot e^{-\frac{pt}{3}},\end{aligned}$$

where the penultimate inequality follows from $|S| - 1 \leq s \leq t$, and the last inequality follows from (19).

Item v is handled in the same way as iv.

Regarding item vi, suppose that for some $m \in [r]$ there are more than $t/4$ of $I(j,m)$ in E'. Similarly to the case ii, each such $I(j,m)$ is satisfied by ρ with independent probability at least $p(1-p)/r > p/(2r)$, so E' is not satisfied with probability at most $(1-p/(2r))^{t/4} < e^{-pt/(8r)}$.

Item vii is treated similarly to vi, with the resulting probability of not satisfying E' being $(1-p/n)^{t/4} < e^{-pt/(4n)} < e^{-pt/(8r)}$, where the last inequality follows from (16). ◀

By (17) and by Lemmas 11, 13, and 15, there is a restriction ρ satisfying all the assertions of the lemmas. Fix any such ρ.

▶ **Definition 16.** A partial assignment σ to the variables of $\mathrm{REF}^F_{s,t}$ is called an *admissible assignment* if it extends ρ and satisfies all the following conditions.

(C1) For each $(i,j) \in [s] \times [t]$, $D(i,j,\cdot,\cdot)$ (resp. $V(i,j,\cdot)$, $I(j,\cdot)$, $L(i,j,\cdot)$, $R(i,j,\cdot)$) either is set to some clause (resp. some $k \in [n]$, some $m \in [r]$, some $j' \in [t]$, some $j' \in [t]$) by σ or contains no variable that is in $\mathrm{dom}(\sigma)$.

(C2) For each $(i,j) \in [s] \times [t]$, if $L(i,j,\cdot)$ or $R(i,j,\cdot)$ is set to some $j' \in [t]$, then both $D(i,j,\cdot,\cdot)$ and $D(i-1,j',\cdot,\cdot)$ are set.

(C3) For each $(i,j) \in [s] \times [t]$, if $D(i,j,\cdot,\cdot)$ is set, then $V(i,j,\cdot)$ is set (if $i \in \{2,\ldots,s\}$) or $I(j,\cdot)$ is set (if $i=1$).

(C4) For each $(i,j) \in [s] \times [t]$, if $D(i,j,\cdot,\cdot)$ is set to a clause $C_{i,j}$, then $C_{i,j}$ is non-tautological and has at least $\min\{s-i,n\}$ many literals. If $D(i,j,\cdot,\cdot)$ is set to a clause $C_{i,j}$ with less than n literals and $V(i,j,\cdot)$ is set to some $k \in [n]$, then none of the literals of x_k is in $C_{i,j}$.

(C5) If $D(s,t,\cdot,\cdot)$ is set, it is set to the empty clause.

(C6) For each $j \in [t]$, if $D(1,j,\cdot,\cdot)$ and $I(j,\cdot)$ are set, then σ satisfies all clauses in (1) with this j.

(C7) For each $i \in \{2,\ldots,s\}, j,j' \in [t]$, if $L(i,j,\cdot)$ (resp. $R(i,j,\cdot)$) is set to j' and both $V(i,j,\cdot)$, $D(i-1,j',\cdot,\cdot)$ are set, then σ satisfies all clauses in (3) (resp. (4)) with these i,j,j'.

(C8) For each $i \in \{2,\ldots,s\}, j,j' \in [t]$, if $L(i,j,\cdot)$ (resp. $R(i,j,\cdot)$) is set to j' and $V(i,j,\cdot)$, $D(i,j,\cdot,\cdot)$, $D(i-1,j',\cdot,\cdot)$ are set, then σ satisfies all clauses in (5) (resp. (6)) with these i,j,j'.

(C9) For each $i \in \{2,\ldots,s\}$, the binary relation $h_{\sigma,i} := \{(Z(i,j,\cdot),j') : j,j' \in [t], Z \in \{L,R\}, \text{and } Z(i,j,\cdot) \text{ is set to } j' \text{ by } \sigma\}$ is a partial injection from $\{Z(i,j,\cdot) : j \in [t], Z \in \{L,R\}\}$ to $[t]$.

For the proofs of the following three lemmas, see the full version [5].

▶ **Lemma 17.** *No clause in* $\mathrm{REF}^F_{s,t}\lceil\rho$ *is falsified by any admissible assignment.*

▶ **Lemma 18.** *There is an admissible assignment.*

▶ **Lemma 19.** *Suppose that a clause E in $\Pi\lceil\rho$ is obtained by the resolution rule from clauses E_0 and E_1. Suppose further that there is an admissible assignment σ which satisfies both conditions*

(i) *every literal in E of a variable in* $\mathrm{dom}(\sigma)$ *is falsified by σ,*

(ii) *for each $Z \in \{D,V,I,R,L\}$, each Z-variable with a home pair Z-important in E is in* $\mathrm{dom}(\sigma)$.

Then there is an admissible assignment τ and $b \in \{0,1\}$ such that i and ii hold with τ in place of σ and E_b in place of E.

These three lemmas easily imply a contradiction, which concludes the proof of Theorem 8.

▶ Remark 20. If we assume $s = n+1$ in Theorem 8 (instead of assuming only $s \geq n+1$) then we can allow t to be smaller: it is enough to assume that $t \geq r^{2+\epsilon}$. This can be useful if one wants to reduce the number of variables of $\mathrm{REF}^F_{s,t}$ while keeping the lower bound of the theorem valid. The latter can be shown by making only the following modification in the proof of Theorem 8: change the definition of p to $p = s^{-1/3}t^{-a'}$ with $a' = \min\left\{\frac{1+\epsilon}{3+\epsilon}, \frac{1}{2}\right\}$, and change the definition of w to $w = s^{1/3}t^{3/5}$.

We note that if in the definition of $\mathrm{REF}^F_{s,t}$ we encode the functions determined by V- and I-variables in binary instead of in unary, the assumption $t \geq r^{3+\epsilon}$ in Theorem 8 is not necessary (and the proof of the theorem simplifies somewhat), and, in addition, the L- and R-variables can be encoded in binary too (with some further simplifications of the proof). This reduces the number of variables of $\mathrm{REF}^F_{s,t}$ in two ways, by allowing a smaller t and by using a more efficient encoding.

▶ Remark 21. Most of the obstacles our proof has to overcome are caused by the nature of the object described by $\mathrm{REF}^F_{s,t}$ and by the fact that the functions determined by V, I, L, R-variables are encoded in unary, rather than in binary. This forces us to work with several notions of width of two kinds, and we cannot keep as an invariant of the maintained partial assignment that it falsifies all literals of a clause as we traverse the refutation (as is the case e.g. in [11]). Moreover, keeping falsified just the literals with important indices and adding some simple conditions about not directly falsifying an axiom (a method which works e.g. in [8] for the pigeonhole principle) is not enough either, because we need to be prepared to consistently answer the prover's questions about clauses situated at remote parts of the same not too small component (learnt through the L- and R-variables). This is further complicated by the need to respond by adding a fresh literal to a clause that has too few literals to make sure its width grows fast enough (such clauses originate in the component of the empty clause), and by the necessity to arrive to a weakening of a clause in F when asked how a clause on level 2 is derived; both are more difficult to meet under the unary encoding and pose specific requirements on random restrictions. Our strategy stores some useful information in the form of negating some other literals than just those with important indices in a clause, as can be seen in the hierarchy of setting of variables of different kinds in Definition 16.

5 Reflection Principle for Resolution

We express the negation of the reflection principle for Resolution by a CNF in the form of a conjunction $\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{s,t}$. The only shared variables by the formulas $\mathrm{SAT}^{n,r}$ and $\mathrm{REF}^{n,r}_{s,t}$ encode a CNF with r clauses in n variables. The meaning of $\mathrm{SAT}^{n,r}$ is that the encoded CNF is satisfiable, while the meaning of $\mathrm{REF}^{n,r}_{s,t}$ is that it has a resolution refutation of s levels of t clauses. A formal definition is given next.

Formula $\mathrm{SAT}^{n,r}$ has the following variables. Variables $C(m,k,b)$, $m \in [r], k \in [n], b \in \{0,1\}$, encode clauses C_m as follows: $C(m,k,1)$ (resp. $C(m,k,0)$) means that the literal x_k (resp. $\neg x_k$) is in C_m. Variables $T(k)$, $k \in [n]$, and variables $T(m,k,b)$, $m \in [r], k \in [n], b \in \{0,1\}$, encode that an assignment to variables $x_1, \ldots, x_n$ satisfies the CNF $\{C_1, \ldots, C_r\}$. The meaning of $T(k)$ is that the literal x_k is satisfied by the assignment. The meaning of $T(m,k,1)$ (resp. $T(m,k,0)$) is that clause C_m is satisfied through the literal x_k (resp. $\neg x_k$).

We list the clauses of $\mathrm{SAT}^{n,r}$:

$$T(m,1,1) \vee T(m,1,0) \vee \ldots \vee T(m,n,1) \vee T(m,n,0) \qquad m \in [r], \tag{20}$$

$$\neg T(m,k,1) \vee T(k) \qquad m \in [r], k \in [n], \tag{21}$$

$$\neg T(m,k,0) \vee \neg T(k) \qquad m \in [r], k \in [n], \tag{22}$$

$$\neg T(m,k,b) \vee C(m,k,b) \qquad m \in [r], k \in [n], b \in \{0,1\}, \tag{23}$$

The meaning of (20) is that clause C_m is satisfied through at least one literal. The meaning of (21) and (22) is that if C_m is satisfied through a literal, then the literal is satisfied. The meaning of (23) is that if C_m is satisfied through a literal, then it contains the literal.

Variables of $\mathrm{REF}^{n,r}_{s,t}$ are the variables $C(m,k,b)$ of $\mathrm{SAT}^{n,r}$ together with all the variables of $\mathrm{REF}^{F}_{s,t}$ for some (and every) F of r clauses in n variables. That is, $\mathrm{REF}^{n,r}_{s,t}$ has the following variables:

$$\begin{array}{ll} C(m,k,b) & m \in [r], k \in [n], b \in \{0,1\}, \\ D(i,j,k,b) & i \in [s], j \in [t], k \in [n], b \in \{0,1\}, \\ R(i,j,j') \text{ and } L(i,j,j') & i \in \{2,\ldots,s\}, j,j' \in [t], \\ V(i,j,k) & i \in \{2,\ldots,s\}, j \in [t], k \in [n], \\ I(j,m) & j \in [t], m \in [r]. \end{array}$$

The clauses of $\mathrm{REF}^{n,r}_{s,t}$ are (2) - (15) of $\mathrm{REF}^{F}_{s,t}$ together with the following clauses (to replace clauses (1)):

$$\neg I(j,m) \vee \neg C(m,k,b) \vee D(1,j,k,b) \qquad j \in [t], m \in [r], k \in [n], b \in \{0,1\}, \tag{24}$$

saying that if clause $C_{1,j}$ is a weakening of clause C_m, then the former contains each literal of the latter.

We now prove the lower bound for $\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{s,t}$ stated in the Introduction as Theorem 3 and restated below as Theorem 22.

▶ **Theorem 22.** *For every $c > 4$ there is $\delta > 0$ and an integer n_0 such that if n, r, s, t are integers satisfying*

$$t \geq s \geq n+1, \qquad r \geq n \geq n_0, \qquad n^c \geq t \geq r^4, \tag{25}$$

then any resolution refutation of $\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{s,t}$ has length greater than 2^{n^δ} (which is exponential in the size of the formula).

Proof. Fix $c > 4$. We first observe that if Π is a resolution refutation of $\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{s,t}$ and σ is a partial assignment such that its domain are all C-variables, then $\Pi \restriction \sigma$ is either a refutation of $\mathrm{REF}^{n,r}_{s,t} \restriction \sigma$, or a refutation of $\mathrm{SAT}^{n,r} \restriction \sigma$. This is because $\Pi \restriction \sigma$ is a resolution refutation and the two restricted formulas do not share any variables.

Let F be a CNF with r clauses in n variables, and let σ_F be a partial assignment such that its domain are all C-variables and σ_F evaluates them so that they describe the clauses of F. Notice that $\mathrm{REF}^{n,r}_{s,t} \restriction \sigma_F$ is $\mathrm{REF}^{F}_{s,t}$, since σ_F turns the clauses (24) into the clauses (1) (and removes the satisfied clauses). Therefore, in the case that $\Pi \restriction \sigma_F$ is a refutation of $\mathrm{REF}^{n,r}_{s,t} \restriction \sigma_F$ and F is unsatisfiable, the lower bound of Theorem 8 applies (setting $\epsilon = 1$ in that theorem, there is n_0 such that conditions (16) on n, r, s, t follow from (25)): the theorem yields some $\delta_1 > 0$ such that the length of $\Pi \restriction \sigma_F$ is at least $2^{n^{\delta_1}}$.

Let us now consider the case that $\Pi \restriction \sigma_F$ is a refutation of $\mathrm{SAT}^{n,r} \restriction \sigma_F$. Let SAT^F stand for $\mathrm{SAT}^{n,r} \restriction \sigma_F$. There is a substitution τ to the variables of SAT^F that turns the clauses of SAT^F into all the clauses of F together with some tautological clauses. It is defined as follows. If $\sigma_F(C(m,k,b)) = 0$, then $\tau(T(m,k,b)) = 0$. This satisfies (21) - (23) and deletes $T(m,k,b)$ from (20). If $\sigma_F(C(m,k,b)) = 1$, then (23) has been satisfied and we define $\tau(T(m,k,b)) = x_k^b$ and $\tau(T(k)) = x_k$. This choice turns (21) - (22) into a tautological clause and correctly substitutes the remaining literals of (20) to yield the m-th clause of F. Thus, if $\Pi \restriction \sigma_F$ is a refutation of $\mathrm{SAT}^{n,r} \restriction \sigma_F$, the substitution τ takes it into a not larger resolution refutation of F (since tautological clauses can be removed from any resolution refutation).

It remains to take any unsatisfiable formula F whose number of clauses is polynomially related to the number of variables and that requires resolution refutations of exponential length, e.g. the pigeonhole principle [6]. A trivial modification of F to serve also in the extreme case $r = n$ allowed by (25) will yield $\delta_2 > 0$ such that any resolution refutation of F has length greater than $2^{n^{\delta_2}}$, where n is the number of variables of F.

Setting δ to the minimum of δ_1 and δ_2 concludes the proof of the theorem. ◀

A similar proof gives Theorem 4. We restate the theorem below for convenience.

▶ **Theorem 23.** *Let $\gamma > 0$ and let $\{A_n\}_{n \geq 1}$ be a family of unsatisfiable CNFs such that A_n is in n variables, has the number of clauses polynomial in n, and has no resolution refutations of length at most 2^{n^γ}. Then there is $\delta > 0$ and a polynomial p such that $A_n \wedge \mathrm{REF}^{A_n}_{n+1,p(n)}$ has no resolution refutations of length at most 2^{n^δ} and has polynomial size* Res(2) *refutations.*

Proof. Let $p(n) \geq \max\{r^4, t_0\}$, where r is the maximum of the number of clauses of A_n and n, and t_0 is given by Theorem 8 for $\epsilon = 1$. This theorem and the assumptions on A_n give the required lower bound. To get the upper bound, start with the Res(2) refutation of $\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{n+1,p(n)}$ given by Theorem 5. Define substitutions σ_{A_n} and τ like in the proof of Theorem 22 with A_n in place of F, and observe again that $((\mathrm{SAT}^{n,r} \wedge \mathrm{REF}^{n,r}_{n+1,p(n)}) \restriction \sigma_{A_n}) \restriction \tau$ is $A_n \wedge \mathrm{REF}^{A_n}_{n+1,p(n)}$ together with some tautological clauses. ◀

References

1 Albert Atserias and María Luisa Bonet. On the automatizability of resolution and related propositional proof systems. *Information and Computation*, 189(2):182–201, 2004.

2 Albert Atserias and Moritz Müller. Automating Resolution is NP-Hard. *arXiv e-prints*, April 2019. arXiv:1904.02991v1.

3 María Luisa Bonet, Toniann Pitassi, and Ran Raz. On Interpolation and Automatization for Frege Systems. *SIAM J. Comput.*, 29(6):1939–1967, 2000. doi:10.1137/S0097539798353230.

4 Stephen A. Cook and Robert A. Reckhow. The Relative Efficiency of Propositional Proof Systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979.

5 Michal Garlík. Resolution Lower Bounds for Refutation Statements. *arXiv e-prints*, May 2019. arXiv:arXiv:1905.12372v1.

6 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2–3):297–308, 1985.

7 Jan Krajíček, Alan Skelley, and Neil Thapen. NP search problems in low fragments of bounded arithmetic. *The Journal of Symbolic Logic*, 72(2):649–672, 2007.

8 Pavel Pudlák. Proofs as Games. *American Mathematical Monthly*, 107(6):541–550, 2000. doi:10.2307/2589349.

9 Pavel Pudlák. On reducibility and symmetry of disjoint NP-pairs. *Theoretical Computer Science*, 295:323–339, 2003.

10 Nathan Segerlind, Samuel R. Buss, and Russel Impagliazzo. A switching lemma for small restrictions and lower bounds for k-DNF resolution. *SIAM Journal on Computing*, 33(5):1171–1200, 2004.

11 Neil Thapen. A Tradeoff Between Length and Width in Resolution. *Theory of Computing*, 12(5):1–14, 2016.

Tangles and Single Linkage Hierarchical Clustering

Eva Fluck
RWTH Aachen University, Germany
fluck@cs.rwth-aachen.de

Abstract

We establish a connection between tangles, a concept from structural graph theory that plays a central role in Robertson and Seymour's graph minor project, and hierarchical clustering. Tangles cannot only be defined for graphs, but in fact for arbitrary connectivity functions, which are functions defined on the subsets of some finite universe, which in typical clustering applications consists of points in some metric space.

Connectivity functions are usually required to be submodular. It is our first contribution to show that the central duality theorem connecting tangles with hierarchical decompositions (so-called branch decompositions) also holds if submodularity is replaced by a different property that we call maximum-submodular.

We then define a natural, though somewhat unusual connectivity function on finite data sets in an arbitrary metric space and prove that its tangles are in one-to-one correspondence with the clusters obtained by applying the well-known single linkage clustering algorithms to the same data set.

The idea of viewing tangles as clusters has first been proposed by Diestel and Whittle [5] as an approach to image segmentation. To the best of our knowledge, our result is the first that establishes a precise technical connection between tangles and clusters.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Unsupervised learning and clustering

Keywords and phrases Tangles, Branch Decomposition, Duality, Hierarchical Clustering, Single Linkage

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.38

1 Introduction

Connectivity in graphs and connectivity systems is a widely studied topic in theoretical computer science, e.g. [11, 8, 6, 7, 5]. On the other hand similarity, especially clustering, is an important and well studied topic in Data Science, e.g. [2, 4, 3, 10, 12]. We study the connection between both concepts by interpreting similarity as connectivity, thus two points are highly connected if their data is very similar and two sets are highly connected if they contain similar data points. Both communities will benefit from such a connection, as it opens up a basis for a wide range of new results. For example connectivity systems provide us with witnesses for the absence of highly connected regions, which is not yet established for clusters, as well as tree like representations of all those highly connected regions. Additionally there is large variety of efficient algorithms to compute or approximate different kind of clusters, which can possibly be used to find algorithms for computing highly connected regions in connectivity systems.

The concept of connectivity systems is based on the notion of connectivity in graphs. Such systems consist of a universe U of usually finitely many elements and some set function on subsets of U describing the connectivity between a set and its complement. These so called connectivity functions are symmetric and submodular. In this context two complementary questions are of interest [9]:

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 38; pp. 38:1–38:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1. What are the highly connected regions of the universe?
2. How can we decompose the universe along low order separations?

An answer to the first question can be given by tangles, which describe highly connected regions in a non-rigorous way. For every low order separation the tangle describes on which side of the separation the large part of the region can be found. Nevertheless, a separation may cut off small parts of the region. In this way, for any single point it is not clearly defined whether it is part of the region or not. On the other hand, the orientation has to be consistent, meaning that all orientations have to point towards the same region.

The second question is addressed by branch decompositions. They contain a ternary tree and a mapping of the elements to the leafs of the tree. Then, the edges of the tree represent separations of the universe. The width of such a decomposition is the largest value of any separation induced by the tree. Both concepts have been introduced on graphs by Robertson and Seymour [11]. An overview on both branch decompositions and tangles for integer-valued functions, as well as their connection can be found in a survey of Grohe [9], which can be translated to real-valued functions. Branch decompositions and tangles address contrary questions, but they are dual. There is a tangle of a certain value if and only if there is no branch decomposition of smaller value. This duality result has first been shown on graphs in [11]. There have been other set functions, that are not submodular, for which duality has been shown. For example Adler et al. [1] have shown duality up to a constant factor for so called hypertangle number and hyperbranch width in hypergraphs. Diestel and Oum [6] developed a general duality theorem in combinatorial structures.

Besides tangles, there is another approach to identify highly similar (connected) regions, called clustering (e.g. [2, 4, 3, 10, 12]). Clustering is the umbrella term for different techniques to define sets of data points that are very similar to each other and not so similar to the data points contained in other sets. Thus, a question arises: How do clustering and tangles relate? A first approach towards this question was done by Diestel and Whittle [5], where they analyzed tangles in digital images as a way to describe the meaningful parts of the image. We consider hierarchical clustering algorithms as introduced by Carlson and Mémoli [2]. The basis of such an algorithm is a function that describes the distance between two sets. Single linkage clustering for example considers the distance of two sets to be the smallest distance between any element from one set to any element from the other set. The hierarchical clustering algorithm then merges the sets with the smallest distance, assigning the resulting partition said value. We allow more than one merge at once, if the distances are equal. The resulting sequence of partitions is represented by a so called dendogram.

1.1 Results

To find a correspondence between tangles and clustering one of the main goals is to find connectivity functions, or functions with similar properties, that represent the different clustering methods. Our main result is, that we are able to specify a function that is correspondent to hierarchical clustering using single linkage. For an arbitrary metric $\mathrm{d}\colon U \times U \to \mathbb{R}$, the set function corresponding to single linkage is the *minimum distance function* $\delta_{\mathrm{d}}\colon 2^U \to \mathbb{R}$, defined by

$$\delta_{\mathrm{d}}(X) = \max_{x \in X, y \notin X} \exp(-\mathrm{d}(x,y)),$$

for all $X \in 2^U \setminus \{\emptyset, U\}$ and $\delta_{\mathrm{d}}(\emptyset) = \delta_{\mathrm{d}}(U) = 0$. An introduction of this function is given in Section 3.

We show that this function is not a classical connectivity function, since it is not submodular. Therefore we define a different property, that we call maximum-submodularity. Our first theorem proves duality between branch decompositions and tangles of functions with this property.

▶ **Theorem 1.** *Let U be a finite set and let ϕ be a maximum-submodular connectivity function. The maximum order of a tangle of ϕ equals the minimum order of a branch decomposition of ϕ. The existence of one is witness to the non-existence of the other.*

A formal version of this theorem and its proof are shown in Section 4. This duality is a key result in the theory of tangles of connectivity systems and suggests that the chosen property on the functions results in a similarly deep theory. It allows us to use maximum-submodular connectivity functions to establish a connection between tangles and clustering.

Our second main result says that the tangles of the minimum distance function are in one-to-one correspondence with the resulting dendogram of single linkage hierarchical clustering. The technical notions appearing in the statement of the theorem will be explained later in this paper.

▶ **Theorem 2.** *Let (U, d) be a metric space.*

1. *For every $r \in \mathbb{R}$ and every cluster $\mathcal{B}$ of the dendogram resulting from single linkage with $|\mathcal{B}| > 1$,*

   $$\mathcal{T} := \{X \subseteq U \mid \delta_{\mathrm{d}}(X) < \exp(-r),\ \mathcal{B} \subseteq X\}$$

   is a δ_{d}-tangle of U of order $\exp(-r)$.
2. *For every δ_{d}-tangle $\mathcal{T}$ of U of order k we can identify a cluster $\mathcal{B}$ of the dendogram resulting from single linkage with $|\mathcal{B}| > 1$ such that*

   $$\mathcal{T} = \{X \subseteq U \mid \delta_{\mathrm{d}}(X) < k,\ \mathcal{B} \subseteq X\}.$$

For every non-singular set contained in a partition of the dendogram we find a distinct δ_{d}-tangle and vice versa. This is to the best of our knowledge the first precise technical connection between tangles and clusters.

2 Preliminaries and Definitions

In our definitions we follow [9]. Our goal is to describe connectivity within some data set U. Therefore we define set functions κ, that aim to describe how strong the connection is between a set and its complement. We say κ is *normalized* if $\kappa(\emptyset) = 0$, κ is *symmetric* if $\kappa(X) = \kappa(\overline{X})$, for all $X \subseteq U$ and κ is *submodular* if $\kappa(X) + \kappa(Y) \geq \kappa(X \cap Y) + \kappa(X \cup Y)$, for all $X, Y \subseteq U$. A set function that is normalized, symmetric and submodular is called *submodular connectivity function.*

▶ **Example 3** (see [9]). Let $G = (V, E)$ be a graph with edge weights $w_E \colon E \to \mathbb{R}$. The weighted edge-connectivity function $\nu \colon 2^V \to \mathbb{R}$, defined as

$$\nu(X) := \sum_{u \in X, v \in V \setminus X, (u,v) \in E} w_E(u, v),$$

is a submodular connectivity function.

We introduce a different type of function, that also describes connectivity. To show that this type has similar properties, we first take a look at basic concepts from the theory of connectivity systems. Most of these concepts have only been studied for integer-valued functions, but for our needs all properties are translatable to real-valued functions. We start with a formal definition of tangles, which are a way to describe highly connected regions.

▶ **Definition 4.** *Let κ be a symmetric set function on the universe U. A κ-*tangle *of order* $\mathrm{ord}(\mathcal{T}) = k \geq 0$ *is a set $\mathcal{T} \subseteq 2^U$ such that:*
T.0 *$\kappa(X) < k$ for all $X \in \mathcal{T}$,*
T.1 *for all $X \subseteq U$ with $\kappa(X) < k$, either $X \in \mathcal{T}$ or $\overline{X} \in \mathcal{T}$ holds,*
T.2 *$X_1 \cap X_2 \cap X_3 \neq \emptyset$ for all $X_1, X_2, X_3 \in \mathcal{T}$ and*
T.3 *$\{x\} \notin \mathcal{T}$ for all $x \in U$.*

We define the *tangle number* $\mathrm{tn}(\kappa)$ of a symmetric set function κ to be the largest possible order for which we can still define a κ-tangle.

We use the following well-known lemma, which states that tangles are in a way closed under intersection and supersets.

▶ **Lemma 5** (see [9]). *Let $\mathcal{T}$ be a κ-tangle of order k. Then it holds that*
1. *for all $X \in \mathcal{T}$ and all $Y \supseteq X$, if $\kappa(Y) < k$ then $Y \in \mathcal{T}$ and*
2. *for all $X, Y \in \mathcal{T}$, if $\kappa(X \cap Y) < k$ then $X \cap Y \in \mathcal{T}$.*

A different way to describe connectivity in a universe is given by branch decompositions. Here we do not look for highly connected regions, but ask ourselves how we can separate the universe into its single elements, using only separations of small value.

▶ **Definition 6.** *Let U be a finite set.*

- *A* pre-decomposition *of U is a pair (T, γ) consisting of a ternary (undirected) tree T and a mapping $\gamma\colon \overrightarrow{E}(T) \to 2^U$, from the directed edges of T to subsets of U, such that*
  - *$\gamma(t, u) = \overline{\gamma(u, t)}$, for all $(t, u) \in \overrightarrow{E}(T)$, and*
  - *$\gamma(s, u_1) \cup \gamma(s, u_2) \cup \gamma(s, u_3) = U$, for all internal nodes $s \in V(T)$ with $N(s) = \{u_1, u_2, u_3\}$.*
- *For leaves $\ell \in L(T)$ with neighbor $N(\ell) = \{u\}$, we write $\gamma(\ell)$ instead of $\gamma(u, \ell)$. We call the $\gamma(\ell)$* atoms *and define $\mathrm{At}(T, \gamma) \coloneqq \{\gamma(\ell) \mid \ell \in L(T)\}$.*
- *A pre-decomposition is* complete *if $|\gamma(\ell)| = 1$, for all leaves $\ell \in L(T)$.*
- *A pre-decomposition is* exact *at an internal node $t \in V(T)$ with $N(t) = \{u_1, u_2, u_3\}$ if all $\gamma(t, u_i)$ are mutually disjoint.*
- *A* decomposition *is a pre-decomposition that is exact at all internal nodes.*
- *A* branch decomposition *is a complete decomposition.*
- *Let κ be a set function on U. The* width *of a pre-decomposition (T, γ) is*

$$\mathrm{wd}(T, \gamma) \coloneqq \max\{\kappa(\gamma(t, u)) \mid (t, u) \in \overrightarrow{E}(T)\}.$$

We define the *branch width* $\mathrm{bw}(\kappa)$ of a symmetric set function κ to be the smallest possible width $\mathrm{wd}(T, \gamma)$ of any branch decomposition (T, γ) on U. For submodular connectivity functions duality between branch decompositions and tangles has been proven. The first to find this duality in graphs were Robertson and Seymour [11]. Duality between branch decompositions and tangles states that a branch decomposition of a certain width is a witness for the non-existence of a tangle of any larger order and vice versa. It follows that for any submodular connectivity function κ it holds that

$$\mathrm{tn}(\kappa) = \mathrm{bw}(\kappa).$$

3 The Minimum Distance Function

▶ **Definition 7** (Minimum Distance). *Let* $\mathrm{d}\colon U \times U \to \mathbb{R}$ *be an arbitrary metric. For a finite data set* U*, the* minimum distance function $\delta_{\mathrm{d}}\colon 2^U \to \mathbb{R}$*, is defined as follows:*

$$\delta_{\mathrm{d}}(X) := \begin{cases} 0 & \text{if } X = \emptyset \text{ or } \overline{X} = \emptyset, \\ \max_{x \in X, x' \in \overline{X}} \exp(-\,\mathrm{d}(x, x')) & \text{otherwise.} \end{cases}$$

This definition yields that X has a high value if there is a point outside of X very close to a point in X. The transformation $\exp(-c \cdot \mathrm{f}(x, y))$, for some constant c and some function f, is often used in clustering applications to transform a dissimilarity function f like a metric into a similarity function. The minimum distance function is in general not submodular, as can be seen with a small example. For an arbitrary $x \in \mathbb{R}^+$ define a one-dimensional universe containing only the following four points $a_1 = x$, $a_2 = x + 1$, $a_3 = -x$ and $a_4 = -x - 1$. Let $X = \{a_1, a_2\}$, $Y = \{a_1, a_3\}$ and the metric $d(u, v) = |u - v|$ is the absolute of the difference. Then $\delta_{\mathrm{d}}(X) = \exp(-2x) < \delta_{\mathrm{d}}(Y) = \delta_{\mathrm{d}}(X \cap Y) = \delta_{\mathrm{d}}(X \cup Y) = \exp(-1)$, for all $x > \frac{1}{2}$

We define a new property, that is similar to submodularity, which allows us to develop similar theories as for submodular connectivity functions.

▶ **Definition 8.** *A set function* κ *on a finite set* U *is* maximum-submodular *if, for all* $X, Y \subseteq U$*,*

$$\max(\kappa(X), \kappa(Y)) \geq \max(\kappa(X \cap Y), \kappa(X \cup Y)).$$

This property is neither a generalization of submodularity nor a specialization. For instance ν as in Example 3 is submodular but not maximum-submodular and in the next lemma we see that the minimum distance function, which in general is not submodular, is maximum-submodular. We call a normalized, symmetric and maximum-submodular set function *maximum-submodular connectivity function.*

▶ **Lemma 9.** *The minimum distance function is a maximum-submodular connectivity function.*

Proof. The minimum distance function is normalized by definition and symmetric since metrics are symmetric. If X or Y are equal to $\emptyset$ or U, maximum-submodularity trivially holds as $\{X, Y\} = \{X \cup Y, X \cap Y\}$ in these cases. Otherwise, we choose $u \in X \cap Y$ and $v \in \overline{X \cap Y}$ such that $\mathrm{d}(u, v) = \delta_{\mathrm{d}}(X \cap Y)$. Analogously we choose $u' \in \overline{X \cup Y} = \overline{X} \cap \overline{Y}$ and $v' \in X \cup Y$. Then w.l.o.g. we distinguish four cases, depending on v and v'.

Case 1: $v \in \overline{X} \cap \overline{Y}$ and $v' \in X \cap Y$ hold: Then w.l.o.g. $v = u'$ and $u = v'$ hold. Therefore $\delta_{\mathrm{d}}(X \cap Y) = \delta_{\mathrm{d}}(X \cup Y)$ and thus $\delta_{\mathrm{d}}(X) \geq \delta_{\mathrm{d}}(X \cap Y)$ and $\delta_{\mathrm{d}}(Y) \geq \delta_{\mathrm{d}}(X \cup Y)$ hold.

Case 2: $v \in \overline{X} \cap \overline{Y}$ and $v' \in \overline{X} \cap Y$ hold: It follows that $\delta_{\mathrm{d}}(X) \geq \delta_{\mathrm{d}}(X \cap Y)$ and $\delta_{\mathrm{d}}(Y) \geq \delta_{\mathrm{d}}(X \cup Y)$ hold.

Case 3: $v, v' \in \overline{X} \cap Y$ holds: It follows that $\delta_{\mathrm{d}}(X) \geq \delta_{\mathrm{d}}(X \cap Y)$ and $\delta_{\mathrm{d}}(Y) \geq \delta_{\mathrm{d}}(X \cup Y)$ hold.

Case 4: $v \in X \cap \overline{Y}$ and $v' \in \overline{X} \cap Y$ hold: In this case it holds that $\delta_{\mathrm{d}}(Y) \geq \delta_{\mathrm{d}}(X \cap Y)$ and $\delta_{\mathrm{d}}(Y) \geq \delta_{\mathrm{d}}(X \cup Y)$. Therefore we have $\delta_{\mathrm{d}}(Y) \geq \max(\delta_{\mathrm{d}}(X \cap Y), \delta_{\mathrm{d}}(X \cup Y))$ and the inequality holds.

As all other cases are symmetric to the four cases shown above, the inequality holds for all $X, Y \subseteq U$. ◀

Next, we consider tangles of the Minimum Distance Function. Firstly, we give an example of such a tangle.

▶ **Example 10.** Let $U \subset \mathbb{R}^n$ be a finite set of points. Let $x_1, x_2 \in U$ be two points such that $\mathrm{d}(x_1, x_2) = \min\{\mathrm{d}(x, y) \mid x, y \in U, x \neq y\}$. Then, for every $k \leq \exp(-\mathrm{d}(x_1, x_2))$,

$$\mathcal{T} := \{X \subseteq U \mid \delta_{\mathrm{d}}(X) < k,\ x_1, x_2 \in X\}$$

is a δ_{d}-tangle of order k.

$\mathcal{T}$ satisfies (T.0), (T.2) and (T.3) by construction. To see that (T.1) is satisfied note that if $x_1 \in X$ and $x_2 \in \overline{X}$ holds then $\delta_{\mathrm{d}}(X) = \exp(-\mathrm{d}(x_1, x_2)) \geq k$ holds.

Having this example we realize that the tangles described are the only δ_{d}-tangles.

▶ **Lemma 11.** *Every δ_{d}-tangle of order k is of the form described as in Example 10. That is, we can identify two points $u, v \in U$ such that $\exp(-\mathrm{d}(u, v)) \geq k$ and for all $X \in \mathcal{T}$ we have $u, v \in X$.*

Proof. To prove this we define the relation ${\delta_{\mathrm{d}}}^k := \{(u, v) \mid \exp(-\mathrm{d}(u, v)) \geq k\}$ and consider the graph $G_k := (U, {\delta_{\mathrm{d}}}^k)$. For every set $X \subseteq U$ with $\delta_{\mathrm{d}}(X) < k$ and every connected component C of G_k, holds either $V(C) \subseteq X$ or $V(C) \subseteq \overline{X}$ by definition. Thus for every δ_{d}-tangle $\mathcal{T}$ of order $\leq k$ it holds that all $X \in \mathcal{T}$ are disjoint unions of connected components of G_k. Additionally, using Lemma 5 (2) every such $\mathcal{T}$ is closed under intersection, as for two sets $X, Y \in \mathcal{T}$ and all connected components C of G_k either $V(C) \subseteq X \cap Y$ or $V(C) \cap X \cap Y = \emptyset$ and thus $\delta_{\mathrm{d}}(X \cap Y) < k$. Suppose for contradiction there is a δ_{d}-tangle $\mathcal{T}$ of order k such that there is no connected component of G_k, of size at least two, that is contained in all $X \in \mathcal{T}$. Let $C_0, \ldots, C_n$ be an enumeration of all connected components of G_k with $|C_i| \geq 2$. Then we can identify a sequence $X_0, \ldots, X_n \in \mathcal{T}$ such that $V(C_i) \nsubseteq X_i$, thus $V(C_i) \cap X_i = \emptyset$. We set $Y_1 := X_0 \cap X_1$ and $Y_{i+1} := Y_i \cap X_{i+1}$. Since $\mathcal{T}$ is closed under intersection, we get $Y_1, \ldots, Y_n \in \mathcal{T}$. As $\mathcal{T}$ is a tangle, $|Y_n| > 1$ has to hold. For every subset $Y \subseteq Y_n$ we have $\kappa(Y) < k$ as $Y_n \cap \bigcup_{i=0}^{n} C_i = \emptyset$. Take an enumeration of all elements $y_1, \ldots, y_m \in Y_n$ and construct a series of sets $Z_1, \ldots, Z_\ell \in \mathcal{T}$ such that $|Z_\ell| = 1$. Clearly this contradicts the existence of $\mathcal{T}$. If $\{y_1\} \in \mathcal{T}$ set $Z_1 := \{y_1\}$, else set $Z_1 := Y_n \setminus \{y_1\} = Y_n \cap \overline{\{y_1\}} \in \mathcal{T}$. If $|Z_i| = 1$ set $\ell = i$ and stop the construction. Otherwise if $\{y_{i+1}\} \in \mathcal{T}$ set $Z_{i+1} := \{y_{i+1}\}$, else set $Z_{i+1} := Z_i \setminus \{y_{i+1}\} = Z_i \cap \overline{\{y_{i+1}\}} \in \mathcal{T}$. As $|Z_i| > |Z_{i+1}|$ this construction terminates and yields the desired contradiction. ◀

From this lemma an important corollary follows. In Section 5 we use this to identify for each tangle a cluster resulting from single linkage hierarchical clustering.

▶ **Corollary 12.** *Let $\mathcal{T}$ be a δ_{d}-tangle of order k over the universe U. There is a unique connected component C of the graph $G = (U, {\delta_{\mathrm{d}}}^k)$, with ${\delta_{\mathrm{d}}}^k := \{(u, v) \mid \exp(-\mathrm{d}(u, v)) \geq k\}$, such that $C \subseteq X$, for all $X \in \mathcal{T}$.*

Proof. We already showed that there exists some component C such that $C \subseteq X$, for all $X \in \mathcal{T}$. Assume there is some component $C' \neq C$ such that $C' \subseteq X$, for all $X \in \mathcal{T}$. Then we have $C' \in \mathcal{T}$ and thus $C \subseteq C'$ which contradicts $C' \neq C$. ◀

4 Duality for Submodular Bounded Functions

Now we prove duality for all maximum-submodular connectivity functions, thus also for the minimal distance function. We achieve a result similar to the theory for submodular connectivity functions, first shown in [11]. To formulate the Duality Theorem we first need a definition.

▶ **Definition 13.** *Let κ be a symmetric set function on U and $\mathcal{A} \subseteq 2^U$.*

- *A pre-decomposition (T, γ) is* over *$\mathcal{A}$ if $At(T, \gamma) \subseteq \mathcal{A}$.*
- *A κ-tangle $\mathcal{T}$* avoids *$\mathcal{A}$ if $\mathcal{T} \cap \mathcal{A} = \emptyset$.*

The Duality Theorem states that there can not be any decomposition over a family of sets, if there is a tangle avoiding this family and vice versa. The proof yields a construction of such a decomposition. The following theorem is a precise formulation of Theorem 1.

▶ **Theorem 14** (Duality Theorem of Submodular Bounded Functions). *Let κ be a maximum-submodular connectivity function on U. Let $\mathcal{A} \subseteq 2^U$ such that $\mathcal{A}$ is closed under taking subsets and $\mathrm{Sing}(U) \subseteq \mathcal{A}$, where $\mathrm{Sing}(U)$ is the set of all singletons from U. Then there is a decomposition of width less than k over $\mathcal{A}$ if and only if there is no κ-tangle of order k that avoids $\mathcal{A}$.*

Assuming the theorem holds, we can directly derive the following corollary using that every branch decomposition of U is complete, thus is over $\mathrm{Sing}(U)$ and every κ-tangle avoids $\mathrm{Sing}(U)$ by definition.

▶ **Corollary 15.** *Let κ be a maximum-submodular connectivity function on U. It holds that*

$$\mathrm{tn}(\kappa) = \mathrm{bw}(\kappa).$$

Looking at the proof of duality for submodular connectivity functions as it is presented in [9], we see that they do not use any properties of the set function, besides symmetry and a transformation from a pre-decomposition into a decomposition of equal width. Therefore, we can adapt that proof if we are able to do a similar transformation. The following lemma shows how to achieve exactness at every node of a pre-decomposition.

▶ **Lemma 16** (Exactness Lemma). *Let κ be a maximum-submodular connectivity function on U and (T, γ) be a pre-decomposition of U. Then there is a mapping $\gamma' \colon \vec{E}(T) \to 2^U$ such that (T, γ') is a decomposition of U satisfying*

- *$\mathrm{wd}(T, \gamma') \leq \mathrm{wd}(T, \gamma)$ and*
- *$\gamma'(\ell) \subseteq \gamma(\ell)$, for all leaves $\ell \in L(T)$.*

Proof. We iteratively construct γ' from γ, keeping the invariants

- $\mathrm{wd}(T, \gamma') \leq \mathrm{wd}(T, \gamma)$ and
- $\gamma'(\ell) \subseteq \gamma(\ell)$, for all leaves $\ell \in L(T)$.

We pick an arbitrary leaf $\ell_{start} \in L(T)$ and set $\gamma'(\ell_{start}, s) \coloneqq \gamma(\ell_{start}, s)$ and $\gamma'(s, \ell_{start}) \coloneqq \gamma(s, \ell_{start})$ for $s \in N(\ell_{start})$. If T only consists of at most two nodes we are done, since (T, γ') is already a decomposition. Otherwise, we traverse the tree with breadth-first search starting at ℓ_{start}. If we reach a node $s \in V(T) \backslash L(T)$ with predecessor $t \in V(T)$, we do the following. Let $u_1, u_2 \in N(s)$ be the successors of s and define $X \coloneqq \gamma'(s, t)$ and $Y_i \coloneqq \gamma(s, u_i)$, for $i = 1, 2$.

If $X \cap (Y_1 \cup Y_2) \neq \emptyset$ we update Y_i to $Y_i \cap \overline{X}$, for $i = 1, 2$. This step is consistent with the invariants as $\kappa(Y_i \cap \overline{X}) \leq \max(\kappa(Y_i \cap \overline{X}), \kappa(Y_i \cup \overline{X})) \leq \max(\kappa(Y_i), \kappa(X))$, where the second inequality holds due to maximum-submodularity, and $Y_i \cap \overline{X} \subseteq \gamma(s, u_i)$, for $i = 1, 2$.

If $Y_1 \cap Y_2 \neq \emptyset$, update Y_1 to $Y_1 \cap \overline{Y_2}$. This step is again consistent with the invariants as $\kappa(Y_1 \cap \overline{Y_2}) \leq \max(\kappa(Y_1 \cap \overline{Y_2}), \kappa(Y_1 \cup \overline{Y_2})) \leq \max(\kappa(Y_1), \kappa(Y_2))$, where the second inequality holds due to maximum-submodularity, and $Y_1 \cap \overline{Y_2} \subseteq \gamma(s, u_1)$.

Set $\gamma'(s, u_i) \coloneqq Y_i$ and $\gamma'(u_i, s) \coloneqq \overline{\gamma'(s, u_i)}$, for $i = 1, 2$. After these steps we know that γ' is exact at s and we do not change γ' for any predecessor of s.

When we reach a leaf $\ell \in L(T)$, we do not change γ' and continue with the next node in the breadth-first search.

This construction yields the desired mapping. ◀

The construction above may result in a tree with empty leafs. But such a leaf can be easily removed by deleting it and its neighbor, connecting the resulting open edges.

Now, we are ready to prove the Duality Theorem for Submodular Bounded Functions.

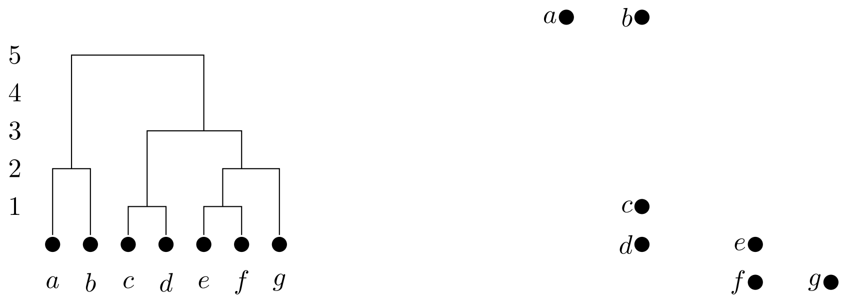
Proof of Theorem 14, see [9]. For the forward direction, we let (T, γ) be a decomposition of U over $\mathcal{A}$ of width less than k. Suppose, for contradiction, $\mathcal{T}$ is a κ-tangle of order k that avoids $\mathcal{A}$. We orient the edges $E(T)$ such that they point in the direction of the set that is contained in the tangle. Such a set always exists as the width is less then k. Thus, formally we orient $(s,t) \in E(T)$ towards t if $\gamma(s,t) \in \mathcal{T}$, and towards s if $\gamma(t,s) = \overline{\gamma(s,t)} \in \mathcal{T}$. As in every oriented tree, there is at least one node $t \in V(T)$ such that all edges incident to t are oriented towards t. If $t \in L(T)$ then $\gamma(t) \in \mathcal{A}$ and $\gamma(t) \in \mathcal{T}$ which contradicts the assumption that $\mathcal{T}$ avoids $\mathcal{A}$. Thus, t is an internal node with $N(t) = \{u_1, u_2, u_3\}$. But since all $\gamma(t, u_i)$ are mutually disjoint and all $\gamma(u_1, t) \in \mathcal{T}$ this contradicts (T.2) as $\gamma(t, u_1) \cup \gamma(t, u_2) \cup \gamma(t, u_3) = U$ and thus $\gamma(u_1, t) \cap \gamma(u_2, t) \cap \gamma(u_3, t) = \emptyset$. It follows that such a κ-tangle can not exist and the forward direction holds.

For the backward direction assume there is no κ-tangle of order k that avoids $\mathcal{A}$. We will construct a pre-decomposition (T, γ) of U over $\mathcal{A}$ of width less than k. Using the Exactness Lemma and since $\mathcal{A}$ is closed under taking subsets it follows that a decomposition of U over $\mathcal{A}$ exists.

We construct such a pre-decomposition (T, γ) inductively on the number of sets $X \subseteq U$ with $\kappa(X) < k$ and neither $X \in \mathcal{A}$ nor $\overline{X} \in \mathcal{A}$.

In the base case, for all $X \subseteq U$ with $\kappa(X) < k$, holds $X \in \mathcal{A}$ or $\overline{X} \in \mathcal{A}$. We define $\mathcal{Y} := \{\overline{X} \mid X \in \mathcal{A} \text{ with } \kappa(X) < k\}$. We know that $\mathcal{Y}$ can not be a tangle, as we assumed that there is no tangle of order k. Since (T.0) and (T.1) hold by assumption on $\mathcal{A}$, either (T.2) or (T.3) have to be false. If $\mathcal{Y}$ violates (T.2) there are three sets $Y_1, Y_2, Y_3 \in \mathcal{Y}$ such that $Y_1 \cap Y_2 \cap Y_3 = \emptyset$. Then $\overline{Y_1}, \overline{Y_2}, \overline{Y_3} \in \mathcal{A}$ and $\overline{Y_1} \cup \overline{Y_2} \cup \overline{Y_3} = U$. We set $T := (\{\ell_1, \ell_2, \ell_3, t\}, \{(\ell_i, t) \mid i = 1,2,3\})$, $\gamma(t, \ell_i) := \overline{Y_i} \in \mathcal{A}$ and $\gamma(\ell_i, t) := Y_i$. Then, (T, γ) is a pre-decomposition of U over $\mathcal{A}$. If $\mathcal{Y}$ violates (T.3) there is some $x \in U$ such that $\{x\} \in \mathcal{Y}$ and thus $\overline{\{x\}} \in \mathcal{A}$. Since $\mathrm{Sing}(U) \subseteq \mathcal{A}$ we have $\{x\} \in \mathcal{A}$. We take $T := (\{s,t\}, \{(s,t)\})$ and $\gamma(s,t) := \{x\}$, $\gamma(t,s) := \overline{\{x\}}$. Then (T, γ) is a pre-decomposition of U over $\mathcal{A}$.

In the inductive step, we have some $X \subseteq U$ with $\kappa(X) < k$ and neither $X \in \mathcal{A}$ nor $\overline{X} \in \mathcal{A}$. We chose X' such that $|X'|$ is minimal with respect to the conditions above. We set $\mathcal{A}^1 := \mathcal{A} \cup 2^{X'}$ and $\mathcal{A}^2 := \mathcal{A} \cup 2^{\overline{X'}}$. By the induction hypothesis there are pre-decompositions (T^1, γ^1) over $\mathcal{A}^1$ and (T^2, γ^2) over $\mathcal{A}^2$. If $\mathrm{At}(T_i, \gamma_i) \subseteq \mathcal{A}$ than (T_i, γ_i) is a pre-decomposition over $\mathcal{A}$ and we are done. Otherwise, we can assume that (T^1, γ^1) is a decomposition, due to the Exactness Lemma 16, thus the $\gamma^1(\ell)$ for all $\ell \in L(T^1)$ are unique. There is some $\ell^1 \in L(T^1)$ such that $\gamma^1(\ell^1) \notin \mathcal{A}$. As the width of (T^1, γ^1) is less than k and no true subset $X'' \subset X'$ fulfills $\kappa(X'') < k$ and neither $X'' \in \mathcal{A}$ nor $\overline{X''} \in \mathcal{A}$, we know that $\gamma^1(\ell^1) = X'$ and that it is the only leaf with this condition. We denote its neighbor by s^1. Let us now consider all $\ell_1^2, \dots, \ell_m^2 \in L(T^2)$ with $\gamma^2(\ell_i^2) \notin \mathcal{A}$. We know that, for all ℓ_i^2, we have $\gamma^2(\ell_i^2) \subseteq \overline{X'}$. We consider all s_i^2 with $N(\ell_i^2) = \{s_i^2\}$. We modify γ^2 by setting $\gamma^2(s_i^2, \ell_i^2) := \overline{X'}$ and $\gamma^2(\ell_i^2, s_i^2) := X'$. The result will still be a pre-decomposition. Then, we construct a pre-decomposition (T, γ) of U over $\mathcal{A}$. We take m disjoint copies (T_i^1, γ_i^1) of (T^1, γ^1). We



(a) The dendogram corresponding to the data points using single linkage.

(b) Data points used to compute the dendogram.

Figure 1 An example of a dendogram. The distance function used is $\ell_{SL}(X,Y) := \min_{x\in X,y\in Y} \|x-y\|$, where $\|\cdot\|$ is the Euclidean norm. This distance function is used in single linkage.

define

$$V(T) := \bigcup_{1\leq i\leq m} V(T_i^1)\backslash\{\ell_i^1\} \cup V(T^2)\backslash\{\ell_1^2,\dots,\ell_m^2\}$$

and take the union of all edge sets where ℓ_i^1 is replaced by s_i^2 and ℓ_i^2 is replaced by s_i^1. Then, we define $\gamma\colon E(T)\to 2^U$ by

$$\gamma(s,t) := \begin{cases} X' & \text{if } (s,t) = (s_i^1, s_i^2) \text{ for some } 1\leq i\leq m, \\ \overline{X'} & \text{if } (s,t) = (s_i^2, s_i^1) \text{ for some } 1\leq i\leq m, \\ \gamma^1(s,t) & \text{if } s,t\in V(T_i^1) \text{ for some } 1\leq i\leq m, \\ \gamma^2(s,t) & \text{if } s,t\in V(T^2). \end{cases}$$

Then, (T,γ) is a pre-decomposition of U over $\mathcal{A}$ of width less than k. ◀

5 Minimum Distance Function and Hierarchical Clustering

To establish the connection between the δ_{d}-tangles and hierarchical clustering, we use Agglomerative Hierarchical Clustering via single linkage on dissimilarity inputs. A dissimilarity input is an instance, where a small function value describes a large similarity between the points. The result of a hierarchical clustering algorithm is a dendogram. For an arbitrary set U, $\mathcal{P}(U)$ denotes the *set of all partitions* of U.

▶ **Definition 17** ([2]). *A* dendogram *over a finite set* $U = \{x_1,\dots,x_n\}$ *is a function* $\theta\colon [0,\infty)\to\mathcal{P}(U)$*, satisfying the following conditions:*

1. $\theta(0) = \{\{x_1\},\dots,\{x_n\}\}$*,*
2. *there exists* t_0 *such that* $\theta(t) = \{U\}$ *for all* $t\geq t_0$*,*
3. *if* $r\leq s$ *then* $\theta(r)$ *is a* refinement *of* $\theta(s)$*, that is for every* $\mathcal{B}\in\theta(r)$ *there is some* $\mathcal{B}'\in\theta(s)$ *such that* $\mathcal{B}\subseteq\mathcal{B}'$*, and*
4. *for all* r *there exists* $\epsilon>0$ *such that* $\theta(r)=\theta(t)$ *for all* $t\in[r,r+\epsilon]$.

The first and second condition ensure that the trivial partitions are part of the dendogram, with the single elements having the smallest possible value and the whole set giving an upper bound. The third condition states that every partition results from a merge of sets contained in a more refined partition. The last condition is a bit technical, and ensures that θ is right continuous. An example of a dendogram can be seen in Figure 1. We allow more than one cluster to merge in one step, as introduced and analyzed by Carlson and Mémoli [2]. The single linkage clustering in this framework works as follows.

▶ **Definition 18** ([2]). *Let (U, d) be a metric space and let $\ell_{SL}\colon 2^U \times 2^U \to \mathbb{R}$ be the single linkage function on U defined by*

$$\ell_{SL}(\mathcal{A}, \mathcal{B}) := \min_{a \in \mathcal{A}, b \in \mathcal{B}} \mathrm{d}(a, b).$$

Define a sequence of distances $R_0, R_1, R_2, \ldots \in [0, \infty)$ and a corresponding sequence of partitions $\Theta_0, \Theta_1, \Theta_2, \ldots \in \mathcal{P}(U)$ by:

- *$R_0 = 0$ and $\Theta_0 = \{\{x_1\}, \ldots, \{x_n\}\}$, with $U = \{x_1, \ldots, x_n\}$,*
- *$R_{i+1} := \min_{\mathcal{B}, \mathcal{B}' \in \Theta_i} \ell_{SL}(\mathcal{B}, \mathcal{B}')$*
- *$\Theta_{i+1} := \Theta_i / \sim_{R_{i+1}}$, where $\mathcal{B} \sim_{R_{i+1}} \mathcal{B}'$ if there exists a sequence of blocks of distance at most R_{i+1}, thus $\mathcal{B} = \mathcal{B}_1, \ldots, \mathcal{B}_s = \mathcal{B}' \in \Theta_i$ with $\ell_{SL}(\mathcal{B}_k, \mathcal{B}_{k+1}) \leq R_{i+1}$, for $k = 1, \ldots, s-1$.*

Then the dendogram for single linkage *is defined by*

$$\theta^{\ell_{SL}}(r) := \Theta_{i(r)},$$

where $i(r) := \max\{i \mid R_i \leq r\}$.

A less technical way to describe this is, that we start with distance $R_0 = 0$ and the partition into single elements. Then we inductively compute the smallest pairwise distance of any two points separated by the partition Θ_i, store this as the next distance value R_{i+1} and merge the corresponding sets to achieve a new partition Θ_{i+1}. We repeat this step until all sets are merged. The resulting dendogram can be interpreted as a decomposition of the universe into its δ_{d}-tangles, where the non-singular blocks of the dendogram correspond to the tangles. The following theorem is a precise formulation of Theorem 2.

▶ **Theorem 19.** *Let (U, d) be a metric space.*

1. *For every $r \in \mathbb{R}$ and every block $\mathcal{B} \in \theta^{\ell_{SL}}(r)$ with $|\mathcal{B}| > 1$,*

   $$\mathcal{T} := \{X \subseteq U \mid \delta_{\mathrm{d}}(X) < \exp(-r),\ \mathcal{B} \subseteq X\}$$

   is a δ_{d}-tangle of U of order $\exp(-r)$*.*
2. *For every δ_{d}-tangle $\mathcal{T}$ of U of order k we can identify a block $\mathcal{B} \in \theta^{\ell_{SL}}(-\log(k))$ with $|\mathcal{B}| > 1$ such that*

   $$\mathcal{T} = \{X \subseteq U \mid \delta_{\mathrm{d}}(X) < k,\ \mathcal{B} \subseteq X\}.$$

Proof. Using the same arguments as in Example 10 the first statement holds. For the second statement one needs the equivalence relation $\sim_r$ on U, where $x \sim_r y$ if and only if there is a sequence of elements $x = x_1, \ldots, x_s = y \in U$ such that $\mathrm{d}(x_i, x_{i+1}) \leq r$. Carlson and Mémoli [2] have shown that the blocks of $\theta^{\ell_{SL}}(r)$ are exactly the equivalence classes $U/{\sim_r}$.

Let $\mathcal{T}$ be a δ_{d}-tangle of order k. Using Corollary 12 we can find a connected component C in the graph $G = (U, {\delta_{\mathrm{d}}}^k)$ such that $C \subseteq X$, for all $X \in \mathcal{T}$. Looking at the definition ${\delta_{\mathrm{d}}}^k := \{(u, v) \mid \exp(-\mathrm{d}(u, v)) \geq k\}$ we see that two elements $u, v \in U$ are connected in G if and only if for their distance holds $\mathrm{d}(u, v) \leq -\log(k)$, thus $u \sim_{-\log(k)} v$. It follows that the equivalence classes of $U/{\sim_{-\log(k)}}$ are exactly the same as the connected components of G. Thus, there is a block $C = \mathcal{B} \in \theta^{\ell_{SL}}(-\log(k))$ that fulfills the requirement. ◀

▶ Remark 20. Let us consider two popular hierarchical clustering algorithms, average linkage and complete linkage. The algorithm is the same as in Definition 18, but the linkage function ℓ changes. The distance of two sets for complete linkage equals the maximum distance of any point from one set to any point from the other set, thus $\ell_{CL}(X,Y) := \max_{x\in X, y\in Y} \mathrm{d}(x,y)$. Using the same trick as for single linkage, a natural related connectivity function is $\kappa_{\mathrm{d}}(X) := \min_{x\in X, y\in \overline{X}} \exp(-\mathrm{d}(x,y))$, for $X \in 2^U \setminus \{\emptyset, U\}$, and $\kappa_{\mathrm{d}} = 0$, otherwise. This function is maximum-submodular and using the Manhattan distance it is even submodular. But in general, for an arbitrary partition P, it holds that

$$\min_{X,Y\in P} \max_{x\in X, y\in Y} \mathrm{d}(x,y) \neq -\log(\max_{X\in P} \min_{x\in X, x'\in \overline{X}} \exp(-\mathrm{d}(x,x'))),$$

as $\ell_{CL}(X,\overline{X}) = \max_{Y\in P} \ell_{CL}(X,Y)$, for arbitrary $X \in P$. This is different for single linkage. It holds that for any partition P of the universe we have

$$\min_{X,Y\in P} \ell_{SL}(X,Y) = -\log(\max_{X\in P} \delta_{\mathrm{d}}(X)),$$

as $\ell_{SL}(X,\overline{X}) = \min_{Y\in P} \ell_{SL}(X,Y)$, for arbitrary $X \in P$. Thus in contrast to single linkage, the optimum of complete linkage ℓ_{CL} used to compute R_{i+1} does not correspond to the optimum according to the connectivity function κ_{d}. For average linkage ($\ell_{AL}(X,Y) := \sum_{x\in X, y\in Y} \frac{\mathrm{d}(x,y)}{|X||Y|}$), a corresponding set function could be $\varphi_{\mathrm{d}}(X) := \sum_{x\in X, y\in \overline{X}} \frac{\exp(-\mathrm{d}(x,y))}{|X||\overline{X}|}$, for $X \in 2^U \setminus \{\emptyset, U\}$, and $\varphi_{\mathrm{d}} = 0$, otherwise. It is neither submodular nor maximum-submodular. Additionally in general $\ell_{AL}(X,\overline{X})$ is not directly computable from $\ell_{AL}(X,Y)$, for $X, Y \in P$ with P an arbitrary partition. To compute $\ell_{AL}(X,\overline{X})$, also the size of all $Y \in P$ is needed. We have

$$\ell_{AL}(X,\overline{X}) = \frac{\sum_{Y\in P, X\neq Y} |Y| \ell_{AL}(X,Y)}{\sum_{Y\in P, X\neq Y} |Y|}.$$

This makes it even harder to find a suitable connectivity function.

6 Conclusion

We establish a precise technical connection between tangles and hierarchical clustering. We can specify this connection for the minimum distance function and single linkage clustering. It is still an open question if there are other clustering algorithms for which we can find corresponding set functions. One of the main obstacles here is, that tangles and the corresponding set functions only look at global connectivity of some set to its converse whereas hierarchical clustering looks at local connectivity between two sets. For single linkage these two notions turned out to be the same.

Our second contribution is to show duality between tangles and branch decompositions for a new class of functions. In our view, the key transformation in the proof of the Duality Theorem is the Exactness Lemma (related to „shifting“ in [6]); this is where submodularity or maximum-submodularity comes in. To broaden the theory, it will be essential to understand under which general conditions such a transformation is possible.

References

1 Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *European Journal of Combinatorics*, 28(8):2167–2181, 2007. doi:10.1016/j.ejc.2007.04.013.

2 Gunnar Carlsson and Facundo Mémoli. Characterization, stability and convergence of hierarchical clustering methods. *Journal of Machine Learning Research*, 11(Apr):1425–1470, 2010.

3 Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical Clustering: Objective Functions and Algorithms. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 378–397. SIAM, 2018. doi:10.1137/1.9781611975031.26.

4 Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 118–127. ACM, 2016. doi:10.1145/2897518.2897527.

5 Reinhard Diestel, Fabian Hundertmark, and Sahar Lemanczyk. Profiles of Separations: in Graphs, Matroids, and Beyond. *Combinatorica*, 39(1):37–75, 2019. doi:10.1007/s00493-017-3595-y.

6 Reinhard Diestel and Sang-il Oum. Unifying Duality Theorems for Width Parameters in Graphs and Matroids (Extended Abstract). In Dieter Kratsch and Ioan Todinca, editors, *Graph-Theoretic Concepts in Computer Science - 40th International Workshop, WG 2014, Nouan-le-Fuzelier, France, June 25-27, 2014. Revised Selected Papers*, volume 8747 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2014. doi:10.1007/978-3-319-12340-0_1.

7 Reinhard Diestel and Geoff Whittle. Tangles and the Mona Lisa. *ArXiv*, 2016. arXiv:1603.06652 [math.CO]. arXiv:1603.06652.

8 Jim Geelen, Bert Gerards, and Geoff Whittle. Tangles, tree-decompositions and grids in matroids. *Journal of Combinatorial Theory, Series B*, 99(4):657–667, 2009. doi:10.1016/j.jctb.2007.10.008.

9 Martin Grohe. Tangled up in blue (a survey on connectivity, decompositions, and tangles). *ArXiv*, 2016. arXiv:1605.06704 [cs.DM].

10 Jon M Kleinberg. An impossibility theorem for clustering. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*, pages 446–453. MIT Press, 2002. URL: http://papers.nips.cc/book/advances-in-neural-information-processing-systems-15-2002.

11 Neil Robertson and Paul D Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-N.

12 Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007. doi:10.1007/s11222-007-9033-z.

Approximating the Orthogonality Dimension of Graphs and Hypergraphs

Ishay Haviv
School of Computer Science, The Academic College of Tel Aviv-Yaffo, Tel Aviv 61083, Israel

Abstract

A t-dimensional orthogonal representation of a hypergraph is an assignment of nonzero vectors in $\mathbb{R}^t$ to its vertices, such that every hyperedge contains two vertices whose vectors are orthogonal. The orthogonality dimension of a hypergraph H, denoted by $\overline{\xi}(H)$, is the smallest integer t for which there exists a t-dimensional orthogonal representation of H. In this paper we study computational aspects of the orthogonality dimension of graphs and hypergraphs. We prove that for every $k \geq 4$, it is NP-hard (resp. quasi-NP-hard) to distinguish n-vertex k-uniform hypergraphs H with $\overline{\xi}(H) \leq 2$ from those satisfying $\overline{\xi}(H) \geq \Omega(\log^{\delta} n)$ for some constant $\delta > 0$ (resp. $\overline{\xi}(H) \geq \Omega(\log^{1-o(1)} n)$). For graphs, we relate the NP-hardness of approximating the orthogonality dimension to a variant of a long-standing conjecture of Stahl. We also consider the algorithmic problem in which given a graph G with $\overline{\xi}(G) \leq 3$ the goal is to find an orthogonal representation of G of as low dimension as possible, and provide a polynomial time approximation algorithm based on semidefinite programming.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Problems, reductions and completeness; Theory of computation → Approximation algorithms analysis

Keywords and phrases orthogonal representations of hypergraphs, orthogonality dimension, hardness of approximation, Kneser and Schrijver graphs, semidefinite programming

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.39

Related Version A full version of the paper is available at `http://arxiv.org/abs/1906.05005`.

Acknowledgements We are grateful to Alexander Golovnev for useful discussions and to the anonymous reviewers for their valuable suggestions.

1 Introduction

A t-dimensional *orthogonal representation* of a hypergraph $H = (V, E)$ is an assignment of a nonzero real vector $u_v \in \mathbb{R}^t$ to every vertex $v \in V$, such that every hyperedge $e \in E$ contains two vertices $v, v' \in e$ whose vectors u_v and $u_{v'}$ are orthogonal. The *orthogonality dimension* of H, denoted by $\overline{\xi}(H)$, is the smallest integer t for which there exists a t-dimensional orthogonal representation of H.[1] The notion of orthogonal representations was introduced for graphs by Lovász [38] in the study of the Shannon capacity and was later involved in a geometric characterization of connectivity properties of graphs by Lovász, Saks, and Schrijver [40]. The orthogonality dimension over the complex field was used by de Wolf [15] in a characterization of the quantum one-round communication complexity of promise equality problems and by Cameron et al. [11] in the study of the quantum chromatic number of graphs (see also [43, 7, 8]). An extension of orthogonal representations, called orthogonal bi-representations, was introduced by Haemers [26] and has found several further applications to information theory and to theoretical computer science.

[1] Orthogonal representations of *graphs* are sometimes defined in the literature as orthogonal representations of the complement, namely, the definition requires vectors associated with *non-adjacent* vertices to be orthogonal. In this paper we have decided to use the other definition because it is extended more naturally to hypergraphs. For a graph G, one can view the notation $\overline{\xi}(G)$ as standing for $\xi(\overline{G})$.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 39; pp. 39:1–39:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Orthogonal representations can be viewed as a generalization of hypergraph vertex colorings, one of the most fundamental and extensively studied topics in graph theory. Recall that a hypergraph H is said to be c-colorable if one can assign one of c colors to every vertex of H such that no hyperedge is monochromatic. The chromatic number of H, denoted by $\chi(H)$, is the smallest integer c for which H is c-colorable. Obviously, every c-coloring of H induces a c-dimensional orthogonal representation of H by assigning the ith unit vector $e_i \in \mathbb{R}^c$ to every vertex colored by the ith color. On the other hand, given a t-dimensional orthogonal representation $(u_v)_{v \in V}$ of H one can assign to every vertex v the vector in $\{-1, 0, +1\}^t$ that consists of the signs of the entries of u_v, and since nonzero orthogonal vectors have distinct sign vectors it follows that H is 3^t-colorable. We conclude that every hypergraph H satisfies

$$\log_3 \chi(H) \leq \overline{\xi}(H) \leq \chi(H). \tag{1}$$

The upper bound in (1) can clearly be tight (take, e.g., a complete graph), and it turns out that there exist graphs whose orthogonality dimension is exponentially smaller than their chromatic number (see Proposition 7).

The current work studies the problem of approximating the orthogonality dimension of graphs and hypergraphs. This research direction was already suggested in the late eighties by Lovász et al. [40], who remarked that computing the orthogonality dimension of graphs seems to be a difficult task (see also [39]). Nevertheless, the only hardness result we are aware of for this graph parameter is the one of Peeters [42], who proved that for every $t \geq 3$ it is NP-hard to decide whether an input graph G satisfies $\overline{\xi}(G) \leq t$ (see also [7]). Before stating our hardness and algorithmic results, we overview related previous work on graph and hypergraph coloring.

1.1 Graph and Hypergraph Coloring

It is well known that the problem of deciding whether an input graph is c-colorable can be easily solved in polynomial time for $c \in \{1, 2\}$ and is NP-hard for every $c \geq 3$ [30].

In 1976, Garey and Johnson [21] have discovered an interesting connection between hardness of graph coloring and the multichromatic numbers of Kneser graphs. For integers $d \geq 2s$, the *Kneser graph* $K(d, s)$ is the graph whose vertices are all the s-subsets of $[d]$, where two sets are adjacent if they are disjoint. A *k-tuple coloring* f of a graph $G = (V, E)$ is an assignment $f(v)$ of a set of k colors to every vertex $v \in V$ such that $f(v) \cap f(v') = \emptyset$ whenever v and v' are adjacent in G. The *kth multichromatic number* of G, denoted by $\chi_k(G)$, is the smallest integer c for which G has a k-tuple coloring with c colors. Equivalently, $\chi_k(G)$ is the smallest integer c for which there exists a homomorphism from G to the Kneser graph $K(c, k)$. Note that $\chi_1(G)$ is simply the standard chromatic number $\chi(G)$. In the seventies, Stahl [45] has made the following conjecture regarding the multichromatic numbers of Kneser graphs (see also [19]).

▶ **Conjecture 1** ([45]). *For all integers k and $d \geq 2s$,*

$$\chi_k(K(d, s)) = \left\lceil \frac{k}{s} \right\rceil \cdot (d - 2s) + 2k.$$

More than forty years later, Conjecture 1 is still widely open. While the right-hand side in the conjecture is known to form an upper bound on $\chi_k(K(d, s))$ for all values of k, d and s, the conjecture was confirmed only for a few special cases. For $k = 1$, the conjecture was proved by Lovász [37] in a breakthrough application of algebraic topology confirming a conjecture by Kneser [33]. Stahl [45] proved that the conjecture holds whenever $1 \leq k \leq s$,

$d = 2s + 1$, or k is divisible by s. Garey and Johnson [21] proved the case of $s = 3$ and $k = 4$, namely, that $\chi_4(K(d,3)) = 2d - 4$, and used it in the analysis of a reduction from 3-colorability to prove that for every $c \geq 6$, it is NP-hard to distinguish between graphs G that satisfy $\chi(G) \leq c$ and those that satisfy $\chi(G) \geq 2c - 4$.

In 1993, Khanna, Linial, and Safra [32] proved that it is NP-hard to decide whether an input graph G satisfies $\chi(G) \leq 3$ or $\chi(G) \geq 5$ (see also [25]). As observed in [6], combining this result with the proof technique of [21] and the case of $s = 3$ and $k = 5$ in Conjecture 1 proved by Stahl [46] (who confirmed there the conjecture for $s \leq 3$ and all integers k and d), it follows that for every $c \geq 6$ it is NP-hard to distinguish between the cases $\chi(G) \leq c$ and $\chi(G) \geq 2c - 2$. Brakensiek and Guruswami [6] improved this result using different techniques and proved the NP-hardness of deciding whether a given graph G satisfies $\chi(G) \leq c$ or $\chi(G) \geq 2c - 1$ for all $c \geq 3$. In a recent work of Bulín, Krokhin, and Opršal [10], the latter condition was further improved to $\chi(G) \geq 2c$. We note that Dinur, Mossel, and Regev [16] proved that assuming a certain variant of the unique games conjecture, deciding whether a given graph G satisfies $\chi(G) \leq 3$ or $\chi(G) \geq c$ is NP-hard for every $c \geq 4$.

We next consider, for any constant $k \geq 3$, the problem of deciding whether an input k-uniform hypergraph (i.e., a hypergraph each of its hyperedges contains exactly k vertices) is c-colorable. While the problem can clearly be solved in polynomial time for $c = 1$, it was shown to be NP-hard for $c = 2$ and $k = 3$ by Lovász [36], motivating the study of the following gap problem: Given an n-vertex k-uniform hypergraph H, decide whether $\chi(H) \leq 2$ or $\chi(H) \geq c$. Guruswami, Håstad, and Sudan [24] proved that the problem is NP-hard for $k \geq 4$ and every constant $c \geq 3$. By combining their proof with the later PCP theorem of Moshkovitz and Raz [41], this result also follows for the super-constant $c = \Omega(\frac{\log\log n}{\log\log\log n})$. For $k = 3$, the NP-hardness was proved for every constant $c \geq 3$ by Dinur, Regev, and Smyth [17]. Their proof approach was extended in a recent work of Bhangale [3], who obtained NP-hardness for every $k \geq 4$ with the improved super-constant $c = \Omega(\log^{\delta} n)$ where $\delta > 0$ is some constant. Under the complexity assumption $\mathsf{NP} \not\subseteq \mathsf{DTIME}(2^{\mathrm{poly}(\log n)})$, several stronger hardness results are known. This includes the case of $k = 3$ and $c = \Omega(\sqrt[3]{\log\log n})$ proved in [17] and the case of $k \geq 4$ and $c = \Omega(\frac{\log n}{\log\log n})$ proved in [3]. For additional related results see, e.g., [3, Table 1] and the references therein.

On the algorithmic side, significant efforts have been made in the literature to obtain polynomial time algorithms for coloring n-vertex 3-colorable graphs using as few colors as possible. This line of research was initiated by a simple algorithm of Wigderson [47] that used $O(\sqrt{n})$ colors. In a series of increasingly sophisticated combinatorial algorithms, Blum [4] improved the number of colors to $\widetilde{O}(n^{3/8})$. Then, Karger, Motwani, and Sudan [29] introduced an algorithm based on a semidefinite relaxation and improved the number of colors to $\widetilde{O}(n^{1/4})$. Combining the combinatorial approach of [4] and the semidefinite relaxation of [29], Blum and Karger [4, 5] improved it to $\widetilde{O}(n^{3/14})$, which was later improved by Arora, Chlamtac, and Charikar [2] and by Chlamtac [13] to $\widetilde{O}(n^{0.2111})$ and $\widetilde{O}(n^{0.2072})$ respectively. The combinatorial component of these algorithms was recently improved by Kawarabayashi and Thorup [31], who reduced the number of colors to $\widetilde{O}(n^{0.19996})$. Halperin et al. [27] have obtained analogue results for coloring n-vertex c-colorable graphs for all constants $c \geq 4$, e.g., for $c = 4$ there exists an efficient algorithm that uses $\widetilde{O}(n^{7/19})$ colors.

For hypergraphs, there exists a simple combinatorial algorithm that given an n-vertex k-uniform 2-colorable hypergraph finds in polynomial time a coloring with $\widetilde{O}(n^{1-1/k})$ colors, as was shown independently by Alon et al. [1] and by Chen and Frieze [12]. For $k = 3$, this algorithm was combined in [1, 12] with the semidefinite programming approach of [29] to obtain a better bound of $\widetilde{O}(n^{2/9})$, which was later improved to $\widetilde{O}(n^{1/5})$ by Krivelevich, Nathaniel, and Sudakov [35]. We note, however, that Alon et al. [1] have provided evidence that the powerful semidefinite approach cannot be applied to coloring k-uniform hypergraphs for $k \geq 4$.

1.2 Our Contribution

The present paper offers hardness and algorithmic results on the orthogonality dimension of graphs and hypergraphs. We first mention that known hardness results on the chromatic number can be used to derive hardness results for the orthogonality dimension. Indeed, the inequalities given in (1) yield that for every integers t_1 and t_2, NP-hardness of deciding whether an input k-uniform hypergraph H satisfies $\chi(H) \leq t_1$ or $\chi(H) \geq t_2$ immediately implies the NP-hardness of deciding whether it satisfies $\overline{\xi}(H) \leq t_1$ or $\overline{\xi}(H) \geq \log_3 t_2$. In particular, the hardness results of [17, 25] imply that for all constants $k \geq 3$ and $t \geq 3$, it is NP-hard to decide whether an input k-uniform hypergraph H satisfies $\overline{\xi}(H) \leq 2$ or $\overline{\xi}(H) \geq t$. For $k = 2$, such a result follows from [16] under a variant of the unique games conjecture. However, for super-constant hardness gaps this implication leads to an exponential loss. In particular, it follows from [3] that for every $k \geq 4$ it is NP-hard to distinguish n-vertex k-uniform hypergraphs H with $\overline{\xi}(H) \leq 2$ from those satisfying $\overline{\xi}(H) \geq \Omega(\log \log n)$. We prove that this exponential loss can be avoided.

▶ **Theorem 2.** *Let $k \geq 4$ be a fixed integer.*

1. *There exists a constant $\delta > 0$ for which it is NP-hard to decide whether an input n-vertex k-uniform hypergraph H satisfies $\overline{\xi}(H) \leq 2$ or $\overline{\xi}(H) \geq \log^{\delta} n$.*
2. *Assuming $\mathsf{NP} \nsubseteq \mathsf{DTIME}(n^{O(\log \log n)})$, for every constant $c > 0$ there is no polynomial time algorithm that decides whether an input n-vertex k-uniform hypergraph H satisfies $\overline{\xi}(H) \leq 2$ or $\overline{\xi}(H) \geq c \cdot \frac{\log n}{\log \log n}$.*

We next consider the hardness of approximating the orthogonality dimension of *graphs*. Our result involves a generalization of orthogonal representations of graphs defined as follows. A t-dimensional *orthogonal k-subspace representation* of a graph $G = (V, E)$ is an assignment of a subspace $U_v \subseteq \mathbb{R}^t$ with $\dim(U_v) = k$ to every vertex $v \in V$, such that the subspaces U_v and $U_{v'}$ are orthogonal whenever v and v' are adjacent in G. For a graph G, let $\overline{\xi}_k(G)$ denote the smallest integer t for which there exists a t-dimensional orthogonal k-subspace representation of G. Note that $\overline{\xi}_1(G) = \overline{\xi}(G)$ for every graph G. We prove the following result.

▶ **Theorem 3.** *For every graph F, it is NP-hard to decide whether an input graph G satisfies $\overline{\xi}(G) \leq \overline{\xi}_3(F)$ or $\overline{\xi}(G) \geq \overline{\xi}_4(F)$.*

With Theorem 3 in hand, it is of interest to find graphs F for which $\overline{\xi}_4(F)$ is large compared to $\overline{\xi}_3(F)$. We consider here, in light of Conjecture 1, the behavior of the $\overline{\xi}_k$ parameters on the Kneser graphs $K(d, s)$. For $k = 1$, it was recently shown that the orthogonality dimension and the chromatic number coincide on Kneser graphs [28]. We further observe, as an application of a result of Bukh and Cox [9], that the values of χ_k and $\overline{\xi}_k$ coincide on the Kneser graphs $K(d, s)$ for every k divisible by s (that is, for all integers $\ell \geq 1$ and $d \geq 2s$, $\overline{\xi}_{\ell \cdot s}(K(d, s)) = \chi_{\ell \cdot s}(K(d, s)) = \ell \cdot d$, and in particular $\overline{\xi}_3(K(d, 3)) = d$; see Corollary 17). It would be natural to ask whether this is also the case for $k = 4$ and $s = 3$.

▶ **Question 4.** *Is it true that for every $d \geq 6$, $\overline{\xi}_4(K(d, 3)) = 2d - 4$?*

A positive answer to Question 4 would imply that for every $t \geq 6$, it is NP-hard to decide whether an input graph G satisfies $\overline{\xi}(G) \leq t$ or $\overline{\xi}(G) \geq 2t-4$, analogously to the hardness result of [21] for the chromatic number.[2]

We finally consider the algorithmic problem in which given an n-vertex k-uniform hypergraph H with constant orthogonality dimension, the goal is to find an orthogonal representation of H of as low dimension as possible. It is not difficult to show that a hypergraph H satisfies $\overline{\xi}(H) \leq 2$ if and only if it is 2-colorable. Hence, by the algorithm of Krivelevich et al. [35], given an n-vertex 3-uniform hypergraph H with $\overline{\xi}(H) \leq 2$ it is possible to efficiently find a coloring of H that uses $\widetilde{O}(n^{1/5})$ colors, and, in particular, to obtain an orthogonal representation of H of this dimension. For graphs, the first nontrivial case is where we are given as input an n-vertex graph G with $\overline{\xi}(G) \leq 3$, for which we prove the following result.

▶ **Theorem 5.** *There exists a randomized polynomial time algorithm that given an n-vertex graph G satisfying $\overline{\xi}(G) \leq 3$, finds a coloring of G that uses at most $\widetilde{O}(n^{0.2413})$ colors. In particular, the algorithm finds an orthogonal representation of G of dimension $\widetilde{O}(n^{0.2413})$.*

In fact, we prove a stronger statement than that of Theorem 5, allowing the input graph G to satisfy $\overline{\xi}_k(G) \leq 3k$ for some integer k (rather than the special case $k = 1$; see Theorem 23).

1.3 Overview of Proofs

Hardness of Approximating the Orthogonality Dimension of Hypergraphs

Theorem 2 is proved in two steps. In the first, we show that approximating the orthogonality dimension of k-uniform hypergraphs becomes harder as the uniformity parameter k grows, and in the second we prove the hardness result for 4-uniform hypergraphs. By combining the two, Theorem 2 follows. We elaborate below on each of these two steps.

The uniformity reduction

Our goal is to show that for every $k_1 \leq k_2$, one can efficiently transform a given k_1-uniform hypergraph H_1 to a k_2-uniform hypergraph H_2 so that $\overline{\xi}(H_1) = \overline{\xi}(H_2)$. We borrow a reduction used in [24] for hypergraph coloring and prove that it preserves the orthogonality dimension. For simplicity of presentation, let us consider here the case of $k_1 = 2$ and $k_2 = 4$. Given an n-vertex graph $G = (V, E)$ we construct a 4-uniform hypergraph H whose vertex set consists of ℓ copies $V_1, \ldots, V_\ell$ of V. For $i \in [\ell]$, let E_i denote the collection of 2-subsets of V_i that correspond to the edges of G. The hyperedges of H are defined as all possible unions of pairs of sets picked from distinct collections E_i and E_j.

As a warm-up, we observe that for a sufficiently large ℓ, say $\ell = n+1$, we have $\chi(G) = \chi(H)$. Indeed, if G is c-colorable then the c-coloring of G applied to each of the copies of V in H implies that H is c-colorable. On the other hand, if G is not c-colorable then for every coloring of H by c colors, every graph (V_i, E_i) contains a monochromatic edge. By $\ell > c$, there are $i \neq j$ and sets $e_1 \in E_i$ and $e_2 \in E_j$ such that all vertices of $e_1 \cup e_2$ share the same color. This implies the existence of a monochromatic hyperedge in H, hence H is not c-colorable.

We next show that for a sufficiently large ℓ we have $\overline{\xi}(G) = \overline{\xi}(H)$. The first direction is equally easy, namely, if G has a t-dimensional orthogonal representation then by assigning its vectors to every copy of V in H we get a t-dimensional orthogonal representation of

[2] It can be shown, using a result of [9], that every $d \geq 6$ satisfies $\overline{\xi}_4(K(d,3)) \geq \lceil 4d/3 \rceil$ (see Lemma 16). Combining this bound with Theorem 3, it follows that for every $t \geq 6$ it is NP-hard to decide whether an input graph G satisfies $\overline{\xi}(G) \leq t$ or $\overline{\xi}(G) \geq \lceil 4t/3 \rceil$.

H. For the other direction, assume that G satisfies $\overline{\xi}(G) > t$, and suppose for the sake of contradiction that $\overline{\xi}(H) \leq t$, i.e., there exists a t-dimensional orthogonal representation $(u_v)_{v \in V(H)}$ of H. By $\overline{\xi}(G) > t$, for every $i \in [\ell]$ there are two vertices $a_i, b_i \in V_i$ adjacent in G whose vectors are not orthogonal, that is, $\langle u_{a_i}, u_{b_i} \rangle \neq 0$. Now, it suffices to show that for some $i \neq j$ the four vectors $u_{a_i}, u_{b_i}, u_{a_j}, u_{b_j}$ are pairwise non-orthogonal, as this would imply a contradiction to the fact that $\{a_i, b_i, a_j, b_j\}$ is a hyperedge of H. It is not difficult to see that for some $i \neq j$ the vectors u_{a_i} and u_{b_j} are not orthogonal. Indeed, let $M_1 \in \mathbb{R}^{\ell \times t}$ be the matrix whose rows are the vectors u_{a_i} for $i \in [\ell]$, and let $M_2 \in \mathbb{R}^{\ell \times t}$ be the matrix whose rows are the vectors u_{b_i} for $i \in [\ell]$. Consider the matrix $M \in \mathbb{R}^{\ell \times \ell}$ defined by $M = M_1 \cdot M_2^T$, and notice that its diagonal entries are all nonzero (because $\langle u_{a_i}, u_{b_i} \rangle \neq 0$ for every i) and that its rank is at most t. Assuming that $\ell > t$, the matrix M must have some nonzero non-diagonal entry (otherwise its rank is ℓ), implying that $\langle u_{a_i}, u_{b_j} \rangle \neq 0$ for some $i \neq j$. This, however, still does not complete the argument, since it might be the case that for these indices i and j, one of the inner products $\langle u_{a_i}, u_{a_j} \rangle$, $\langle u_{b_i}, u_{a_j} \rangle$, and $\langle u_{b_i}, u_{b_j} \rangle$ is zero, avoiding the contradiction.

To overcome this difficulty, we use a symmetrization argument showing that the assumption $\overline{\xi}(H) \leq t$ implies that H has some t'-dimensional orthogonal representation $(w_v)_{v \in V(H)}$, where t' is not too large, with the following symmetry property: For every i and j, if $\langle w_{a_i}, w_{b_j} \rangle \neq 0$ then the inner products $\langle w_{a_i}, w_{a_j} \rangle$, $\langle w_{b_i}, w_{a_j} \rangle$, and $\langle w_{b_i}, w_{b_j} \rangle$ are all nonzero. With such an orthogonal representation, applying the above argument with $\ell > t'$ would certainly imply a contradiction and complete the proof. We achieve the symmetry property for $t' = t^4$ using vector tensor products. Namely, we assign to every vertex a_i the vector $w_{a_i} = u_{a_i} \otimes u_{b_i} \otimes u_{a_i} \otimes u_{b_i}$, to every vertex b_i the vector $w_{b_i} = u_{a_i}^{\otimes 2} \otimes u_{b_i}^{\otimes 2}$, and to every other vertex v the vector $w_v = u_v^{\otimes 4}$. It is straightforward to verify that $(w_v)_{v \in V(H)}$ forms a t'-dimensional orthogonal representation of H. Moreover, by standard properties of tensor products we have

$$\langle w_{a_i}, w_{b_j} \rangle = \langle u_{a_i}, u_{a_j} \rangle \cdot \langle u_{b_i}, u_{a_j} \rangle \cdot \langle u_{a_i}, u_{b_j} \rangle \cdot \langle u_{b_i}, u_{b_j} \rangle,$$

which can be used to obtain that if $\langle w_{a_i}, w_{b_j} \rangle \neq 0$ then the other three inner products $\langle w_{a_i}, w_{a_j} \rangle$, $\langle w_{b_i}, w_{a_j} \rangle$, and $\langle w_{b_i}, w_{b_j} \rangle$ are nonzero as well. This completes the proof sketch for $k_1 = 2$ and $k_2 = 4$. For the proof of the general case, we generalize the tensor-based argument to k-tuples of vertices and extend the matrix reasoning applied above using bounds on off-diagonal Ramsey numbers.

Hardness for 4-uniform hypergraphs

We next consider the hardness of approximating the orthogonality dimension of 4-uniform hypergraphs. A significant difficulty in proving such a result lies at the challenge of proving strong lower bounds on the orthogonality dimension. For the sake of comparison, in most hardness proofs for hypergraph coloring the lower bound on the chromatic number of the hypergraph H constructed by the reduction is shown by an upper bound on its independence number $\alpha(H)$ and the standard inequality $\chi(H) \geq \frac{|V(H)|}{\alpha(H)}$. This approach cannot be used for the orthogonality dimension, which in certain cases can be exponentially smaller than this ratio (see Proposition 7). Exceptions of this approach, where the lower bound on the chromatic number is not proved via the independence number, are the works of Dinur et al. [17] and Bhangale [3] on which we elaborate next.

A standard technique in proving hardness of approximation results is to reduce from the Label Cover problem, in which given a collection of constraints over a set of variables the goal is to decide whether there exists an assignment that satisfies all the constraints or any assignment satisfies only a small fraction of them. In such reductions, every variable over a domain $[R]$ is encoded via an error-correcting code known as the long code, and the constraints are replaced by "inner" constraints designed for the specific studied problem. One way to view the long code is as the graph whose vertices are all subsets of $[R]$ where two sets are adjacent if they are disjoint [18]. In the hardness proof of [17] for the chromatic number of 3-uniform hypergraphs, this graph was replaced by the induced subgraph that consists only of subsets of a given size (i.e., a Kneser graph), where a label $\alpha \in [R]$ is encoded by the 2-coloring of the vertices according to whether the sets contain α or not. The analysis of [17] is crucially based on the large chromatic number of Kneser graphs [37] and on the property that every coloring of Kneser graphs with number of colors smaller than their chromatic number enforces a large color class that includes a monochromatic edge. The latter property was proved in [17] using the chromatic number of the Schrijver graph [44], a vertex-critical subgraph of the Kneser graph. The approach of [17] was recently extended by Bhangale [3], who used in his long code construction only the vertices of the Schrijver graph. The fact that this subgraph has much fewer vertices and yet large chromatic number has led to improved hardness factors. However, for the analysis to work the "inner" constraints had to include four vertices, and this is the reason that the result was obtained for 4-uniform hypergraphs (and not for 3-uniform hypergraphs as in [17]).

In the current work we prove the hardness of approximating the orthogonality dimension of 4-uniform hypergraphs using the reduction applied in [3] for hypergraph coloring. While we achieve the same hardness factors as in [3], the analysis relies on several different ideas and tools. This includes the aforementioned symmetrization argument for orthogonal representations, a lower bound of Golovnev et al. [23] on the sparsity of low rank matrices with nonzero entries on the diagonal, and the orthogonality dimension of Schrijver graphs determined in [28] (see Theorem 8).

Hardness of Approximating the Orthogonality Dimension of Graphs

Theorem 3 relates the hardness of approximating the orthogonality dimension of graphs to orthogonal subspace representations. Our starting point is the NP-hardness of deciding whether an input graph G satisfies $\overline{\xi}(G) \leq 3$ [42]. Following an approach of Garey and Johnson [21], our reduction constructs a graph G' defined as the lexicographic product of some fixed graph F and the input graph G. Namely, we replace every vertex of F by a copy of G and replace every edge of F by a complete bipartite graph between the vertex sets associated with its endpoints (see Definition 11). We then show that if $\overline{\xi}(G) \leq 3$ then G' has a $\overline{\xi}_3(F)$-dimensional orthogonal representation, whereas if $\overline{\xi}(G) \geq 4$ the orthogonality dimension of G' is at least $\overline{\xi}_4(F)$. It would be interesting to figure out the best hardness factors that Theorem 3 can yield (see Question 4). We note, though, that our approach is limited to multiplicative hardness gaps bounded by 2, as it is easy to see that every graph F satisfies $\overline{\xi}_4(F) \leq \overline{\xi}_3(F) + \overline{\xi}_1(F) \leq 2 \cdot \overline{\xi}_3(F)$.

Coloring Graphs with Orthogonality Dimension Three

Consider the problem in which given an n-vertex graph G satisfying $\overline{\xi}(G) \leq 3$, the goal is to find an orthogonal representation of G of as low dimension as possible. Employing an approach of [14], we attempt to find a coloring of G with a small number of colors, as this in particular gives an orthogonal representation of the same dimension. As mentioned before, for every $c \geq 3$ there are known efficient algorithms for coloring n-vertex c-colorable graphs, however, our only guarantee on G is that its orthogonality dimension is at most 3.

Interestingly, it follows from a theorem of Kochen and Specker [34] (see also [22]) that the largest possible chromatic number of such a graph is 4. It follows that given an n-vertex graph G with $\overline{\xi}(G) \leq 3$, one can simply apply the efficient algorithm of [27] for coloring 4-colorable graphs to obtain a coloring of G by $\widetilde{O}(n^{\gamma})$ colors where $\gamma = 7/19 \approx 0.368$.

To improve on this bound, we show an efficient algorithm that finds a large independent set in a given graph G satisfying $\overline{\xi}(G) \leq 3$. We consider two cases according to the maximum degree in the graph. If G has a vertex of large degree then the algorithm finds a large independent set in its neighborhood. This can be done since the assumption $\overline{\xi}(G) \leq 3$ implies that the neighborhood of every vertex of G is 2-colorable (just as for 3-colorable graphs). Otherwise, in case that all the degrees in G are small, we use an algorithm of Karger et al. [29] based on a semidefinite relaxation of the chromatic number, called the *vector chromatic number*. Our analysis relies on a result by Lovász [38] relating the (strict) vector chromatic number of graphs to their orthogonality dimension. Now, by repeatedly omitting independent sets in G, we obtain a coloring that uses $\widetilde{O}(n^{1/4})$ colors. This can be slightly improved to $\widetilde{O}(n^{0.2413})$ by applying the refined analysis of Arora et al. [2] for the rounding algorithm of [29].

As already mentioned, our algorithm can handle any graph G that satisfies $\overline{\xi}_k(G) \leq 3k$ for some integer k, rather than for $k = 1$ (see Theorem 23). The generalized analysis involves a connection, recently proved by Bukh and Cox [9], between the (strict) vector chromatic number and the graph parameters $\overline{\xi}_k$.

1.4 Outline

The rest of the paper is organized as follows. In Section 2, we provide some background on the orthogonality dimension and on the Kneser and Schrijver graphs. In Section 3, we prove our hardness and algorithmic results on the orthogonality dimension of graphs, confirming Theorems 3 and 5. The analysis of the uniformity reduction and the proof of Theorem 2 can be found in the full version of the paper.

2 Preliminaries

2.1 Orthogonality Dimension

We define the orthogonality dimension of hypergraphs over a general field.

▶ **Definition 6.** *A t-dimensional* orthogonal representation *of a hypergraph $H = (V, E)$ over a field $\mathbb{F}$ is an assignment of a vector $u_v \in \mathbb{F}^t$ with $\langle u_v, u_v \rangle \neq 0$ to every vertex $v \in V$, such that for every hyperedge $e \in E$ there are two vertices $v, v' \in e$ satisfying $\langle u_v, u_{v'} \rangle = 0$. The* orthogonality dimension *of a hypergraph $H = (V, E)$ over $\mathbb{F}$, denoted by $\overline{\xi}(H, \mathbb{F})$, is the smallest integer t for which there exists a t-dimensional orthogonal representation of H over $\mathbb{F}$. For the real field $\mathbb{R}$, we let $\overline{\xi}(H)$ stand for $\overline{\xi}(H, \mathbb{R})$.*

The following proposition shows that there are graphs whose orthogonality dimension is exponentially smaller than their chromatic number.

▶ **Proposition 7.** *There exists an explicit family of graphs G_t such that $\overline{\xi}(G_t) \leq t$ and $\chi(G_t) \geq 2^{\Omega(t)}$.*

Proof. Let t be an integer divisible by 4. Consider the graph $G_t = (V, E)$ whose vertices are all the $\frac{t}{2}$-subsets of $[t]$ where two sets $A, B \in V$ are adjacent if their intersection size satisfies $|A \cap B| = \frac{t}{4}$. Notice that by $|A| = |B| = \frac{t}{2}$, this condition is equivalent to $|A \triangle B| = \frac{t}{2}$.

Assign to every vertex A the vector $u_A \in \{\pm 1\}^t$ where $(u_A)_i = +1$ if $i \in A$ and $(u_A)_i = -1$ otherwise. We claim that $(u_A)_{A \in V}$ is an orthogonal representation of G_t. Indeed, for every adjacent vertices $A, B \in V$ we have

$$\langle u_A, u_B \rangle = (-1) \cdot |A \triangle B| + (t - |A \triangle B|) = t - 2 \cdot |A \triangle B| = 0.$$

This implies that $\overline{\xi}(G_t) \leq t$. On the other hand, by a celebrated result of Frankl and Rödl [20], $\alpha(G_t) \leq 2^{c \cdot t}$ for some $c < 1$, implying that

$$\chi(G_t) \geq \frac{|V|}{\alpha(G_t)} \geq \frac{\binom{t}{t/2}}{2^{c \cdot t}} \geq 2^{(1-c-o(1)) \cdot t},$$

completing the proof. ◀

2.2 Kneser and Schrijver Graphs

For integers $d \geq 2s$, the *Kneser graph* $K(d,s)$ is the graph whose vertices are all the s-subsets of $[d]$, where two sets are adjacent if they are disjoint.

A set $A \subseteq [d]$ is said to be *stable* if it does not contain two consecutive elements modulo d (that is, if $i \in A$ then $i+1 \notin A$, and if $d \in A$ then $1 \notin A$). In other words, a stable subset of $[d]$ is an independent set in the cycle C_d with the numbering from 1 to d along the cycle. For integers $d \geq 2s$, the *Schrijver graph* $S(d,s)$ is the graph whose vertices are all the stable s-subsets of $[d]$, where two sets are adjacent if they are disjoint.

The orthogonality dimension of the Kneser and Schrijver graphs was determined in [28] using topological methods.

▶ **Theorem 8** ([28]). *For every $d \geq 2s$, $\overline{\xi}(K(d,s)) = \overline{\xi}(S(d,s)) = d - 2s + 2$.*

The number of vertices in $K(d,s)$ is clearly $\binom{d}{s}$. We need the following simple bound, given in [17], on the number of vertices in $S(d,s)$.

▶ **Lemma 9** ([17]). *For every $d \geq 2s$, the number of vertices in $S(d,s)$ is at most $\binom{d}{d-2s}$.*

3 The Orthogonality Dimension of Graphs

In this section we focus on the orthogonality dimension of graphs and prove Theorems 3 and 5.

3.1 Orthogonal Subspace Representations

We generalize the notion of orthogonal representations over the real field by assigning to every vertex a subspace of a given dimension, so that adjacent vertices are assigned to orthogonal subspaces.

▶ **Definition 10.** *A t-dimensional* orthogonal k-subspace representation *of a graph $G = (V,E)$ is an assignment of a subspace $U_v \subseteq \mathbb{R}^t$ with $\dim(U_v) = k$ to every vertex $v \in V$, such that the subspaces U_v and $U_{v'}$ are orthogonal whenever v and v' are adjacent in G. For a graph G, let $\overline{\xi}_k(G)$ denote the smallest integer t for which there exists a t-dimensional orthogonal k-subspace representation of G.*

Clearly, $\overline{\xi}(G) = \overline{\xi}_1(G)$ for every graph G. It is also easy to see that the multichromatic numbers of graphs, defined in Section 1.1, bound the parameters $\overline{\xi}_k$ from above, namely, $\overline{\xi}_k(G) \leq \chi_k(G)$ for every G and k.

3.2 Hardness

In this section we prove Theorem 3, namely that for every graph F, it is NP-hard to decide whether an input graph G satisfies $\overline{\xi}(G) \leq \overline{\xi}_3(F)$ or $\overline{\xi}(G) \geq \overline{\xi}_4(F)$. The proof employs the notion of lexicographic product of graphs, defined as follows.

▶ **Definition 11.** *The* lexicographic product *of the graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, denoted by $G_1 \bullet G_2$, is the graph whose vertex set is $V_1 \times V_2$ where two vertices (x_1, y_1) and (x_2, y_2) are adjacent if either $\{x_1, x_2\} \in E_1$ or $x_1 = x_2$ and $\{y_1, y_2\} \in E_2$.*

One can view the graph $G_1 \bullet G_2$ as the graph obtained from G_1 by replacing every vertex by a copy of G_2 and replacing every edge by a complete bipartite graph between the vertex sets associated with its endpoints. We need the following property of the orthogonality dimension of lexicographic products of graphs.

▶ **Lemma 12.** *For every two graphs G_1 and G_2, $\overline{\xi}(G_1 \bullet G_2) = \overline{\xi}_k(G_1)$ where $k = \overline{\xi}(G_2)$.*

Proof. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs and denote $k = \overline{\xi}(G_2)$.

We first prove that $\overline{\xi}(G_1 \bullet G_2) \geq \overline{\xi}_k(G_1)$. Denote $t = \overline{\xi}(G_1 \bullet G_2)$, then there exists a t-dimensional orthogonal representation $(u_{(x,y)})_{(x,y) \in V_1 \times V_2}$ of $G_1 \bullet G_2$. For every $x \in V_1$, let U_x denote the subspace of $\mathbb{R}^t$ spanned by all vectors $u_{(x,y)}$ with $y \in V_2$. By the definition of $G_1 \bullet G_2$, the subspaces U_x and $U_{x'}$ are orthogonal whenever x and x' are adjacent in G_1. Further, for every $x \in V_1$, the restriction of the given orthogonal representation to the copy of G_2 associated with x forms an orthogonal representation of G_2, so by $k = \overline{\xi}(G_2)$ it follows that $\dim(U_x) \geq k$. This implies that there exists a t-dimensional orthogonal k-subspace representation of G_1, hence $\overline{\xi}_k(G_1) \leq t$, as required.

We next prove that $\overline{\xi}(G_1 \bullet G_2) \leq \overline{\xi}_k(G_1)$. Denote $t = \overline{\xi}_k(G_1)$, then there exists a t-dimensional orthogonal k-subspace representation $(U_x)_{x \in V_1}$ of G_1. By $k = \overline{\xi}(G_2)$, there exists a k-dimensional orthogonal representation $(u_y)_{y \in V_2}$ of G_2. For every $x \in V_1$, the fact that $\dim(U_x) = k$ implies that there exists an orthogonal linear transformation T_x from $\mathbb{R}^k$ onto U_x. We assign to every vertex $(x, y) \in V_1 \times V_2$ of $G_1 \bullet G_2$ the nonzero vector $w_{(x,y)} = T_x(u_y) \in \mathbb{R}^t$. We claim that this is a t-dimensional orthogonal representation of $G_1 \bullet G_2$. To see this, let (x_1, y_1) and (x_2, y_2) be two adjacent vertices in $G_1 \bullet G_2$. If x_1 and x_2 are adjacent in G_1 then the subspaces U_{x_1} and U_{x_2} are orthogonal, hence the vectors $w_{(x_1,y_1)} \in U_{x_1}$ and $w_{(x_2,y_2)} \in U_{x_2}$ are orthogonal as well. Otherwise, $x_1 = x_2$ and the vertices y_1 and y_2 are adjacent in G_2. This implies that the vectors u_{y_1} and u_{y_2} are orthogonal, and since T_{x_1} preserves inner products it follows that $T_{x_1}(u_{y_1})$ and $T_{x_1}(u_{y_2})$ are orthogonal, and we are done. ◀

Equipped with Lemma 12, we are ready to prove Theorem 3.

Proof of Theorem 3. Fix a graph F. We reduce from the NP-hard problem of deciding whether an input graph G satisfies $\overline{\xi}(G) \leq 3$ [42]. The reduction maps an input graph G to the lexicographic product $G' = F \bullet G$ of F and G. The graph G' can clearly be constructed in polynomial time. The correctness of the reduction follows from Lemma 12. Indeed, we have $\overline{\xi}(G') = \overline{\xi}_k(F)$ for $k = \overline{\xi}(G)$, so if $\overline{\xi}(G) \leq 3$ then $\overline{\xi}(G') \leq \overline{\xi}_3(F)$ and if $\overline{\xi}(G) \geq 4$ then $\overline{\xi}(G') \geq \overline{\xi}_4(F)$. ◀

3.3 Algorithm

Before presenting our algorithm, we provide some background on the vector chromatic number of graphs.

3.3.1 Vector Chromatic Number

Consider the following two relaxations of the chromatic number of graphs, due to Karger, Motwani, and Sudan [29].

▶ **Definition 13.** *For a graph $G = (V, E)$ the* vector chromatic number *of G, denoted by $\chi_{\mathrm{v}}(G)$, is the minimal real value of $\kappa > 1$ for which there exists an assignment of a unit real vector u_v to every vertex $v \in V$ such that $\langle u_v, u_{v'} \rangle \leq -\frac{1}{\kappa-1}$ whenever v and v' are adjacent in G.*

▶ **Definition 14.** *For a graph $G = (V, E)$ the* strict vector chromatic number *of G, denoted by $\chi_{\mathrm{v}}^{(\mathrm{s})}(G)$, is the minimal real value of $\kappa > 1$ for which there exists an assignment of a unit real vector u_v to every vertex $v \in V$ such that $\langle u_v, u_{v'} \rangle = -\frac{1}{\kappa-1}$ whenever v and v' are adjacent in G.*

It is well known and easy to verify that for every graph G, $\chi_{\mathrm{v}}(G) \leq \chi_{\mathrm{v}}^{(\mathrm{s})}(G) \leq \chi(G)$. Karger et al. [29] have obtained the following algorithmic result.

▶ **Theorem 15** ([29]). *There exists a randomized polynomial time algorithm that given an n-vertex graph G with maximum degree at most Δ and $\chi_{\mathrm{v}}(G) \leq \kappa$ for some $\kappa \geq 2$, finds an independent set of size $\widetilde{\Omega}(\frac{n}{\Delta^{1-2/\kappa}})$.*

Note that the well-known Lovász ϑ-function, introduced in [38], is known to satisfy $\vartheta(G) = \chi_{\mathrm{v}}^{(\mathrm{s})}(\overline{G})$ for every graph G [29]. Combining this with a result of [38], it follows that the orthogonality dimension forms an upper bound on the strict vector chromatic number, that is, $\overline{\xi}(G) \geq \chi_{\mathrm{v}}^{(\mathrm{s})}(G)$ for every graph G. This was recently generalized by Bukh and Cox as follows (see [9, Proposition 23]).

▶ **Lemma 16** ([9]). *For every graph G and an integer k, $\overline{\xi}_k(G) \geq k \cdot \chi_{\mathrm{v}}^{(\mathrm{s})}(G)$.*

We derive that the graph parameters $\overline{\xi}_k$ and χ_k coincide on Kneser graphs $K(d, s)$ whenever k is divisible by s.

▶ **Corollary 17.** *For all integers $\ell \geq 1$ and $d \geq 2s$, $\overline{\xi}_{\ell \cdot s}(K(d, s)) = \ell \cdot d$.*

Proof. For the upper bound on $\overline{\xi}_{\ell \cdot s}(K(d, s))$, recall that Conjecture 1 was confirmed for $k = \ell \cdot s$ in [45], hence $\overline{\xi}_{\ell \cdot s}(K(d, s)) \leq \chi_{\ell \cdot s}(K(d, s)) = \ell \cdot d$. For the lower bound, combine Lemma 16 with the fact that $\chi_{\mathrm{v}}^{(\mathrm{s})}(K(d, s)) = \frac{d}{s}$ (see [38]), to get that $\overline{\xi}_{\ell \cdot s}(K(d, s)) \geq \ell \cdot d$. ◀

3.3.2 The Algorithm

We present an efficient algorithm that given a graph G that satisfies $\overline{\xi}(G) \leq 3$ (or, more generally, $\overline{\xi}_k(G) \leq 3k$ for some k), finds a coloring of G with relatively few colors. We use the following simple claim of Blum [4] which reduces the algorithmic task of coloring a graph to the algorithmic task of finding a large independent set in it.

▶ **Claim 18** ([4]). *Let $\mathcal{G}$ be a graph family which is closed under taking induced subgraphs, let $c_1, c_2 > 1$ be arbitrary fixed constants, and let $f : \mathbb{N} \to \mathbb{N}$ be any non-decreasing function satisfying $c_1 \cdot f(n) \leq f(2n) \leq c_2 \cdot f(n)$ for all sufficiently large n. Then if there exists a (randomized) polynomial time algorithm which finds an independent set of size $f(n)$ in any n-vertex graph $G \in \mathcal{G}$, then there exists a (randomized) polynomial time algorithm which finds an $O(\frac{n}{f(n)})$-coloring of any n-vertex graph $G \in \mathcal{G}$.*

We need the following two simple lemmas.

▶ **Lemma 19.** *Let $G = (V, E)$ be a graph such that $\overline{\xi}_k(G) \leq 2k$ for some integer k. Then G is 2-colorable.*

Proof. Let $G = (V, E)$ be a graph satisfying $\overline{\xi}_k(G) \leq 2k$, and let $(U_v)_{v \in V}$ be a $2k$-dimensional orthogonal k-subspace representation of G. It suffices to prove that every connected component of G is 2-colorable. Fix a vertex v in some connected component of G, and observe that there exists a unique subspace of $\mathbb{R}^{2k}$ of dimension k orthogonal to U_v. This implies that the orthogonal subspace representation of G provides a 2-coloring of the connected component of v, where the vertices of even distance from v are assigned to U_v and the vertices of odd distance from v are assigned to its orthogonal complement $U_v^{\perp}$, so we are done. ◀

For a graph G and a vertex v, let $N(v)$ denote the neighborhood of v in G and let $G[N(v)]$ denote the subgraph of G induced by $N(v)$.

▶ **Lemma 20.** *Let $G = (V, E)$ be a graph such that $\overline{\xi}_k(G) \leq 3k$ for some integer k. Then for every vertex $v \in V$ the subgraph $G[N(v)]$ is 2-colorable.*

Proof. Let $G = (V, E)$ be a graph satisfying $\overline{\xi}_k(G) \leq 3k$, and let $(U_v)_{v \in V}$ be a $3k$-dimensional orthogonal k-subspace representation of G. Let $v \in V$ be a vertex in G. The subspace U_v is orthogonal to all subspaces $U_{v'}$ with $v' \in N(v)$. By $\dim(U_v) = k$, the orthogonal complement of U_v in $\mathbb{R}^{3k}$ has dimension $2k$. It follows that $G[N(v)]$ admits a $2k$-dimensional orthogonal k-subspace representation, hence by Lemma 19 it is 2-colorable, as required. ◀

We are ready to prove the following result.

▶ **Theorem 21.** *There exists a randomized polynomial time algorithm that given an n-vertex graph G that satisfies $\overline{\xi}_k(G) \leq 3k$ for some k, finds a coloring of G that uses at most $\widetilde{O}(n^{1/4})$ colors. In particular, the algorithm finds an orthogonal representation of G of dimension $\widetilde{O}(n^{1/4})$.*

Proof. By Claim 18 it suffices to show that there exists a randomized polynomial time algorithm that given an n-vertex graph $G = (V, E)$ with $\overline{\xi}_k(G) \leq 3k$ for some k, finds in G an independent set of size $\widetilde{\Omega}(n^{3/4})$. We consider two possible cases. Suppose first that there exists a vertex $v \in V$ in G whose degree is at least $\Delta = n^{3/4}$. Then by Lemma 20 the subgraph $G[N(v)]$ is 2-colorable, so we can find an independent set in G of size at least $\frac{\Delta}{2}$ by finding in polynomial time a 2-coloring of $G[N(v)]$ and taking a largest color class. Otherwise, the maximum degree of G is at most Δ. By Lemma 16 we have

$$\chi_{\mathrm{v}}(G) \leq \chi_{\mathrm{v}}^{(\mathrm{s})}(G) \leq \frac{\overline{\xi}_k(G)}{k} \leq 3,$$

so by Theorem 15 we can find in polynomial time an independent set of size $\widetilde{\Omega}(\frac{n}{\Delta^{1/3}}) \geq \widetilde{\Omega}(n^{3/4})$, and we are done. ◀

To improve the number of used colors, we employ the following result that stems from the analysis by Arora et al. [2] of the semidefinite relaxation of [29]. (For an explicit statement, see [14, Lemma 4.12], applied with $\sigma = 0.5$, $c \approx 0.04843726$, and $\delta \approx 0.7587$.)

▶ **Theorem 22** ([2]). *There exists a randomized polynomial time algorithm that given an n-vertex graph G with maximum degree at most $\Delta = n^{0.7587}$ and $\chi_{\mathrm{v}}(G) \leq 3$, finds an independent set in G of size at least $\widetilde{\Omega}(n \cdot \Delta^{-0.3179}) \geq \widetilde{\Omega}(n^{0.7587})$.*

By applying Theorem 22 instead of Theorem 15 in the proof of Theorem 21, we obtain the following slight improvement, confirming Theorem 5.

▶ **Theorem 23.** *There exists a randomized polynomial time algorithm that given an n-vertex graph G that satisfies $\overline{\xi}_k(G) \leq 3k$ for some k, finds a coloring of G that uses at most $\widetilde{O}(n^{0.2413})$ colors.*

References

1 Noga Alon, Pierre Kelsen, Sanjeev Mahajan, and Ramesh Hariharan. Approximate Hypergraph Coloring. *Nord. J. Comput.*, 3(4):425–439, 1996. Preliminary version in SWAT'96.

2 Sanjeev Arora, Eden Chlamtac, and Moses Charikar. New approximation guarantee for chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 215–224, 2006.

3 Amey Bhangale. NP-hardness of coloring 2-colorable hypergraph with poly-logarithmically many colors. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 15:1–15:11, 2018.

4 Avrim Blum. New Approximation Algorithms for Graph Coloring. *J. ACM*, 41(3):470–516, 1994.

5 Avrim Blum and David R. Karger. An $O(n^{3/14})$-Coloring Algorithm for 3-Colorable Graphs. *Inf. Process. Lett.*, 61(1):49–53, 1997.

6 Joshua Brakensiek and Venkatesan Guruswami. New Hardness Results for Graph and Hypergraph Colorings. In *Proceedings of the 31st Conference on Computational Complexity (CCC)*, pages 14:1–14:27, 2016.

7 Jop Briët, Harry Buhrman, Debbie Leung, Teresa Piovesan, and Florian Speelman. Round Elimination in Exact Communication Complexity. In *Proceedings of the 10th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC)*, volume 44 of *LIPIcs*, pages 206–225, 2015.

8 Jop Briët and Jeroen Zuiddam. On the orthogonal rank of Cayley graphs and impossibility of quantum round elimination. *Quantum Information & Computation*, 17(1&2):106–116, 2017.

9 Boris Bukh and Christopher Cox. On a Fractional Version of Haemers' Bound. *IEEE Trans. Inform. Theory*, 65(6):3340–3348, 2019.

10 Jakub Bulín, Andrei A. Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 602–613, 2019.

11 Peter J. Cameron, Ashley Montanaro, Michael W. Newman, Simone Severini, and Andreas J. Winter. On the Quantum Chromatic Number of a Graph. *Electr. J. Comb.*, 14(1), 2007.

12 Hui Chen and Alan M. Frieze. Coloring Bipartite Hypergraphs. In *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 345–358, 1996.

13 Eden Chlamtac. Approximation Algorithms Using Hierarchies of Semidefinite Programming Relaxations. In *Proceedings of the 48th Symposium on Foundations of Computer Science (FOCS)*, pages 691–701, 2007.

14 Eden Chlamtáč and Ishay Haviv. Linear Index Coding via Semidefinite Programming. *Combinatorics, Probability & Computing*, 23(2):223–247, 2014. Preliminary version in SODA'12.

15 Ronald de Wolf. Quantum Computing and Communication Complexity. PhD thesis, Universiteit van Amsterdam, 2001.

16 Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional Hardness for Approximate Coloring. *SIAM Journal on Computing*, 39(3):843–873, 2009. Preliminary version in STOC'06.

17 Irit Dinur, Oded Regev, and Clifford D. Smyth. The Hardness of 3-Uniform Hypergraph Coloring. *Combinatorica*, 25(5):519–535, 2005. Preliminary version in FOCS'02.

18 Irit Dinur and Shmuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005. Preliminary version in STOC'02.

19 Peter Frankl and Zoltán Füredi. Extremal problems concerning Kneser graphs. *J. Comb. Theory, Ser. B*, 40(3):270–284, 1986.

20 Peter Frankl and Vojtěch Rödl. Forbidden intersections. *Trans. Amer. Math. Soc.*, 300(1):259–286, 1987.

21 Michael R. Garey and David S. Johnson. The Complexity of Near-Optimal Graph Coloring. *J. ACM*, 23(1):43–49, 1976.

22 Chris D. Godsil and Joseph Zaks. Colouring the Sphere. *University of Waterloo research report*, CORR 88-12, 1988.

23 Alexander Golovnev, Oded Regev, and Omri Weinstein. The Minrank of Random Graphs. *IEEE Trans. Inform. Theory*, 64(11):6990–6995, 2018. Preliminary version in RANDOM'17.

24 Venkatesan Guruswami, Johan Håstad, and Madhu Sudan. Hardness of Approximate Hypergraph Coloring. *SIAM J. Comput.*, 31(6):1663–1686, 2002. Preliminary version in FOCS'00.

25 Venkatesan Guruswami and Sanjeev Khanna. On the Hardness of 4-Coloring a 3-Colorable Graph. *SIAM J. Discrete Math.*, 18(1):30–40, 2004. Preliminary version in CCC'00.

26 Willem H. Haemers. An upper bound for the Shannon capacity of a graph. In *Algebraic methods in graph theory, Vol. I, II (Szeged, 1978)*, volume 25 of *Colloq. Math. Soc. János Bolyai*, pages 267–272. North-Holland, Amsterdam, 1981.

27 Eran Halperin, Ram Nathaniel, and Uri Zwick. Coloring k-colorable graphs using relatively small palettes. *J. Algorithms*, 45(1):72–90, 2002. Preliminary version in SODA'01.

28 Ishay Haviv. Topological Bounds on the Dimension of Orthogonal Representations of Graphs. *Eur. J. Comb.*, 81:84–97, 2019.

29 David R. Karger, Rajeev Motwani, and Madhu Sudan. Approximate Graph Coloring by Semidefinite Programming. *J. ACM*, 45(2):246–265, 1998. Preliminary version in FOCS'94.

30 Richard M. Karp. Reducibility Among Combinatorial Problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, pages 85–103, 1972.

31 Ken-ichi Kawarabayashi and Mikkel Thorup. Coloring 3-Colorable Graphs with Less than $n^{1/5}$ Colors. *J. ACM*, 64(1):4:1–4:23, 2017. Preliminary versions in FOCS'12 and STACS'14.

32 Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the Hardness of Approximating the Chromatic Number. *Combinatorica*, 20(3):393–415, 2000. Preliminary version in ISTCS'93.

33 Martin Kneser. Aufgabe 360. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 58(2):27, 1955.

34 Simon Kochen and E. P. Specker. The Problem of Hidden Variables in Quantum Mechanics. *Journal of Mathematics and Mechanics*, 17(1):59–87, 1967.

35 Michael Krivelevich, Ram Nathaniel, and Benny Sudakov. Approximating Coloring and Maximum Independent Sets in 3-Uniform Hypergraphs. *J. Algorithms*, 41(1):99–113, 2001. Preliminary version in SODA'01.

36 László Lovász. Coverings and colorings of hypergraphs. In *Proceedings of the 4th Southeastern Conf. on Comb.*, pages 3–12. Utilitas Math., 1973.

37 László Lovász. Kneser's Conjecture, Chromatic Number, and Homotopy. *J. Comb. Theory, Ser. A*, 25(3):319–324, 1978.

38 László Lovász. On the Shannon capacity of a graph. *IEEE Trans. Inform. Theory*, 25(1):1–7, 1979.

39 László Lovász. Chapter 5: Orthogonal representations and their dimension. *Lecture notes: Selected topics in graph theory*, 2016. Available at `https://http://web.cs.elte.hu/ lovasz/kurzusok/orth16-2.pdf`.

40 László Lovász, Michael Saks, and Alexander Schrijver. Orthogonal representations and connectivity of graphs. *Linear Algebra and its Applications*, 114/115:439–454, 1989. Special Issue Dedicated to Alan J. Hoffman.

41 Dana Moshkovitz and Ran Raz. Two-query PCP with subconstant error. *J. ACM*, 57(5):29:1–29:29, 2010. Preliminary version in FOCS'08.

42 René Peeters. Orthogonal representations over finite fields and the chromatic number of graphs. *Combinatorica*, 16(3):417–431, 1996.

43 Giannicola Scarpa and Simone Severini. Kochen-Specker sets and the rank-1 quantum chromatic number. *IEEE Trans. Inform. Theory*, 58(4):2524–2529, 2012.

44 Alexander Schrijver. Vertex-critical subgraphs of Kneser graphs. *Nieuw Arch. Wiskd.*, 26(3):454–461, 1978.

45 Saul Stahl. n-Tuple colorings and associated graphs. *J. Comb. Theory, Ser. B*, 20(2):185–203, 1976.

46 Saul Stahl. The multichromatic numbers of some Kneser graphs. *Discrete Mathematics*, 185(1-3):287–291, 1998.

47 Avi Wigderson. Improving the Performance Guarantee for Approximate Graph Coloring. *J. ACM*, 30(4):729–735, 1983. Preliminary version in STOC'82.

Domination Above r-Independence: Does Sparseness Help?

Carl Einarson
Royal Holloway, University of London, UK
einarson.carl@gmail.com

Felix Reidl
Birkbeck, University of London, UK
f.reidl@dcs.bbk.ac.uk

Abstract

Inspired by the potential of improving tractability via gap- or above-guarantee parametrisations, we investigate the complexity of Dominating Set when given a suitable lower-bound witness. Concretely, we consider being provided with a maximal r-independent set X (a set in which all vertices have pairwise distance at least $r+1$) along the input graph G which, for $r \geq 2$, lower-bounds the minimum size of any dominating set of G. In the spirit of gap-parameters, we consider a parametrisation by the size of the "residual" set $R := V(G) \setminus N[X]$.

Our work aims to answer two questions: How does the constant r affect the tractability of the problem and does the restriction to sparse graph classes help here? For the base case $r = 2$, we find that the problem is paraNP-complete even in apex- and bounded-degree graphs. For $r = 3$, the problem is W[2]-hard for general graphs but in FPT for nowhere dense classes and it admits a linear kernel for bounded expansion classes. For $r \geq 4$, the parametrisation becomes essentially equivalent to the natural parameter, the size of the dominating set.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Dominating Set, Above Guarantee, Kernel, Bounded Expansion, Nowhere Dense

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.40

Related Version A full version of the paper is available at `https://arxiv.org/abs/1906.09180`

Acknowledgements We thank our anonymous reviewer for helpfully pointing out how to achieve a linear kernel in Theorem 14.

Introduction

The research of *above/below guarantee* parameters as first used by Mahajan and Raman [22] was an important step towards studying problems whose natural parameters provided only trivial and unsatisfactory answers. Case in point, the motivation for Mahajan and Raman was the observation that every CNF-SAT formula with m clauses trivially has an assignment that satisfies $\geq \lceil m/2 \rceil$ clauses, thus question for the maximum number of satisfied clauses is only interesting if $k > \lceil m/2 \rceil$, which of course renders the parametrised approach unnecessary. They therefore proposed to study parametrisations "above guarantee": going with the previous example, we would ask to satisfy $\lceil m/2 \rceil + k$ clauses or 'k above guarantee'. After some isolated results in that direction (e.g. [15, 17]) the programme took up steam after Mahajan *et al.* presented several results and pointers in new directions [23] (e.g. [3, 16, 18, 17]). In particular, Cygan *et al.* broke new ground for Multiway Cut and Vertex Cover with algorithms that run in $O^*(4^k)$ time, where k is the *gap parameter* between an appropriate LP-relaxation and the integral optimum [5]. Lokshtanov *et al.* improved the Vertex Cover case to $O^*(2.3146^k)$ using a specialized branching algorithm [21].

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 40; pp. 40:1–40:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The latter result highlights an important realization: these alternative, smaller parameters might not only provide the means to investigate problems without "good" natural parameters, it might also provide us with faster algorithms in practise! Gap- and above-guarantee parameters are attractive because there is a reasonable chance that they are small in real-world scenarios, something we often cannot expect from natural parameters.

To the best of our knowledge, so far no gap-parameter results are known for domination problems and an above/below-guarantee result is only known in bounded-degree graphs [23]. This is probably due to the fact that there are no simple "natural" upper/lower bounds and in the case of gap-parameters the LP-dualities do not provide much purchase. We therefore explore this topic under the most basic assumptions: we are provided with a witness for a lower bound on the domination number *as input* and consider parametrisations that arise from this additional information. In the case of Dominating Set, the witness takes the form of a 2*-independent set*, that is, a set in which all vertices have pairwise distance at least three. Note that this approach also captures a form of duality: the LP-dual of dominating set describes a 2-independent set, however, in general the two optima are arbitrarily far apart. Recently, Dvořák highlighted this connection [7] and proved that in certain sparse classes the gap between the dual optima is bounded by a constant.

Thus, assume we are given a maximal 2-independent set X alongside the input graph G. A parametrisation by $|X|$ would go against the spirit of gap-parameters, instead we parametrise by the size of *residual set* $R := V(G) \setminus N[X]$, that is, all vertices that lie at distance two from X (since X is maximal, no vertex can have distance three or more). We choose this particular parameter for two reasons:

(i) For $|R| = 0$ the problem is decidable in polynomial time since the domination number of the graph is precisely $|X|$.

(ii) The set $X \cup R$ is a dominating set of G.

The first property is of course an important pre-requisite for the problem to be in FPT under this parametrisation, while the second property guarantees us that the dominating set size lies in-between $|X|$ and $|X| + |R|$.

Our first investigatory dimension is the constant $r = 2$ in the 2-independent set: intuitively, increasing the minimum distance between vertices in X increases the size of the parameter $|R|$ and imposes more structure on the input instance. Our second dimension encompasses an approach that has been highly successful in improving tractability of domination problems: restricting the inputs to sparse graphs. While Dominating Set is W[2]-complete in general graphs, Alber *et al.* showed that it is fpt in planar graphs [1]; Alon and Gutner later proved that assuming degeneracy is sufficient [4]. Philip, Raman, and Sikdar extended this result yet further to graphs excluding a fixed bi-clique and also proved that it admits a polynomial kernel [25]. A related line of research was the hunt for *linear* kernels in sparse classes. Beginning with such a kernel on planar graphs by Alber, Fellows, and Niedermeier [2], results on apex-minor free graphs [10], graphs excluding a minor [11] and classes excluding a topological minor [12] were soon proven. Recently, a linear kernel for graphs of bounded expansion [6] (and an almost-linear kernel for nowhere dense graphs [9]) has subsumed all previous results.

Our investigation of Dominating Set parametrised above an r-independent set, for $r \geq 2$, led us to the following results. For $r = 2$, the problem is paraNP-complete already for $|R| = 1$, squashing all hope for an FPT or even XP algorithm. This also holds true if the inputs are restricted to sparse graph classes (apex-graphs/graphs of maximum degree six).

For $r = 3$, the problem is W[2]-hard in general graphs but admits an XP-algorithm. In nowhere dense and bounded expansion classes, it is fixed-parameter tractable. We further show, in the probably most technical part of this paper, that it admits a linear kernel in bounded expansion classes.

Finally, for $r \geq 4$ the problem remains W[2]-hard in general graphs and essentially degenerates to Dominating Set (hence, all the above mentioned results in sparse classes translate in the parametrisation above r-independence).

1 Preliminaries

A set $X \subseteq V(G)$ is *r-independent* if each pair of distinct vertices in X have distance at least $r{+}1$, thus an independent set is 1-independent. We write $N(v)$ and $N[v]$, respectively, for the neighbourhood and the closed neighbourhood of a vertex v. We extend this notation to sets as follows: for $X \subseteq V(G)$ we let $N(X)$ be all vertices not in X that have a neighbour in X and $N[X] := X \cup N(X)$. We let $N^i(X)$ be all vertices not in X that are at most distance i from any vertex in X and we let $N^i[X] = X \cup N^i(X)$. A vertex set $Z \subseteq V(G)$ is *dominated* by a set $D \subseteq V(G)$ if for every vertex $z \in Z$ we have $N[z] \cap D \neq \emptyset$, D is then called a *Z-dominator*. We let $\mathbf{ds}(G)$ denote the size of a minimum dominating set of G.

Dominating Set above r-independence

Input: A graph G, a maximal r-independent set $X \subseteq V(G)$, an integer p.
Parameter: The size of the residual set $R := V(G) \setminus N[X]$.
Problem: Does G have a dominating set of size p?

Note that $X \cup R$ is trivially a dominating set and that, for $r \geq 2$, it holds that $\mathbf{ds}(G) \geq |X|$, thus we will tacitly assume in the following that $|X| \leq p \leq |X| + |R|$ since all other instances are trivial.

We will frequently invoke the terms *bounded expansion* and *nowhere dense* to describe graph classes. The definitions of these terms requires the introduction of several concepts which will not be useful for the remainder of the paper, we refer the reader to the book by Nešetřil and Ossona de Mendez [24]. In this context, it is important to know that bounded expansion classes generalize most structurally sparse classes (planar, bounded genus, bounded degree, H-minor free, H-topological minor free) and nowhere dense classes contain bounded expansion classes in turn. The following lemma and propositions for those two sparse graph classes will be needed in the remainder of this paper:

▶ **Lemma 1** (Twin class lemma [13, 26]). *For every bipartite nowhere dense class there exists a constant ω and a function $f(s) = O(s^{o(1)})$ such that for every member $G = (X, Y, E)$ of the class it holds that*

1. $|\{u \mid \deg(u) > 2\tau\}_{u \in Y}| \leq 2\tau \cdot |X|$, *and*
2. $|\{N(u)\}_{u \in Y}| \leq (\min\{4^\tau, \omega(e\tau)^\omega\} + 2\tau) \cdot |X|$.

where $\tau = f(|X|) = O(|X|^{o(1)})$. If G is from a bounded expansion class, τ can be assumed a constant as well.

We will also frequently invoke the following result regarding first-order (FO) model checking in bounded expansion and nowhere dense classes:

▶ **Proposition 2** (Dvořák, Král, and Thomas [8]). *For every bounded expansion class, the first-order model checking problem is solvable in linear fpt-time parametrised by the size of the input formula.*

This result has since been extended to nowhere dense classes as well. Here, *almost linear fpt-time* means running time of the form $O(f(k) \cdot n^{1+o(1)})$ for some function f.

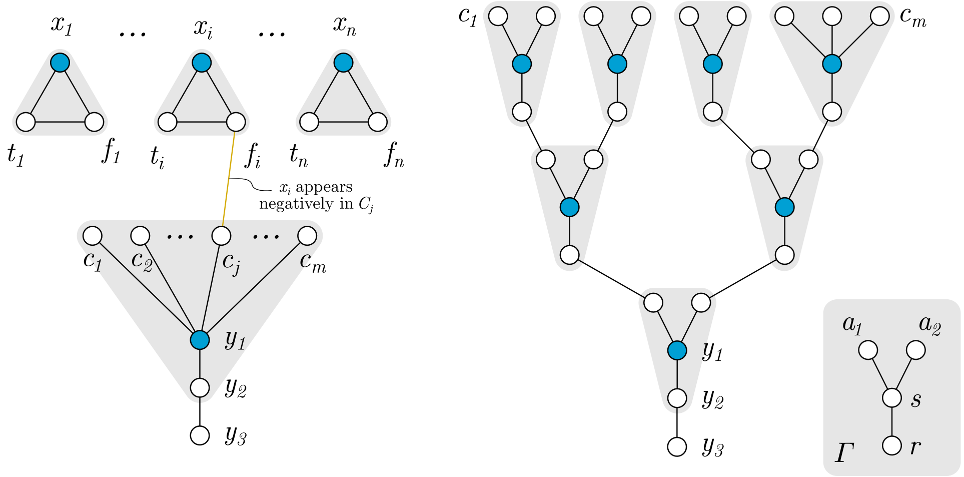


Figure 1 Sketch of reduction from 3SAT to Dominating Set above 2-independent set. The left side shows the basic reduction, the right side shows the bounded-degree replacement gadget for the clause part, with the tree-gadget Γ highlighted on the bottom right. The 2-independent set X is coloured blue and $N[X]$ is shaded in grey. In both constructions the set R consists only of y_3.

▶ **Proposition 3** (Grohe, Kreutzer, and Siebertz [14]). *For every nowhere dense class, he first-order model checking problem is solvable in almost linear fpt-time parametrised by the size of the input formula.*

2 Above 2-independence: hard as nails

In this section we will show that when we let $r = 2$, we find that the problem is paraNP-complete for $|R| = 1$, hence this parametrisation does not even admit an XP-algorithm. In the following we first present a reduction from 3SAT and then discuss how to modify it to reduce into sparse graph classes.

Since X is a (maximal) 2-independent set, we know that each vertex in R is a neighbour of some vertex in $N(X)$, otherwise we could add this vertex to X. Let us now describe the reduction. Let ϕ be 3SAT-instance with variables $x_1, \dots, x_n$ and clauses $C_1, \dots, C_m$. We construct G as follows (*cf.* Figure 1):

1. For each variable x_i, add a triangle with vertices x_i, t_i, f_i.
2. For each clause C_j add a vertex c_j. If the variable x_i occurs positively in C_j, add the edge $c_j t_i$; if it occurs negatively, add the edge $c_j f_i$.
3. Add a single vertex y_1 to the graph and connect it to each clause variable c_i. Add two further vertices y_2, y_3 and add the edges $y_1 y_2$ and $y_2 y_3$.

We further set $X := \{x_1, \dots, x_n, y_1\}$ as our 2-independent set; notice that the only vertex not contained in $N[X]$ is y_3. Hence, $R := \{y_3\}$.

▶ **Lemma 4.** *ϕ is satisfiable iff G has a dominating set of size $|X|$.*

Proof. Assume ϕ is satisfiable and fix one satisfying assignment I. We construct a dominating set D as follows: if x_i is true under I, add t_i to D; otherwise add f_i. Since I satisfies every clause of ϕ the dominating set so far dominates every clause vertex and, of course, every variable gadget. The remaining undominated vertices are y_1, y_2, y_3, thus adding y_2 to D yields a dominating set of G of size $|X|$.

In the other direction, assume that D is a dominating set of G of size $|X|$. Since y_3 is a pendant vertex we can assume that $y_2 \in D$ (if y_3 would be in D we could exchange it for y_2). That leaves $|X| - 1 = n$ vertices in D, precisely the number of variable-gadgets. Since every variable gadget must include at least one vertex of D, we conclude the every such gadget contains precisely one dominating vertex. Since that depletes our budget, no other vertex is contained in D.

By the usual exchange argument we may assume that the dominating vertex in each variable gadget is either f_i or t_i and not x_i for $1 \leq i \leq n$; hence the dominating vertices inside the variable gadgets encode a variable assignment I_D of ϕ. Finally, note that the clause vertices are not dominated by y_2 and y_1 is not contained in D. Hence, they must be completely dominated by vertices contained in the variable gadgets. Then, by construction, the assignment I_D satisfies ϕ and the claim follows. ◀

We conclude that 3SAT many-one reduces to Dominating Set above 2-independence already with $|R| = 1$. We obtain the following two corollaries that demonstrate that sparseness cannot help tractability here:

▶ **Corollary 5.** Dominating Set *above* 2-*independence is* paraNP-*complete in apex-graphs.*

Proof. We use the above construction but reduce from a planar variant of 3SAT. To ensure that we can construct variable-gadgets without edge crossings, we choose to reduce from Lichtenstein's Planar 3SAT variant [20] which ensures that the following graph G' derived from the Planar 3SAT instance ϕ is planar:

1. Every variable x_i of ϕ is represented by two literal vertices t_i, f_i with the edge $t_i f_i \in G'$
2. Each clause C_j is represented by a vertex c_j. If the variable x_i occurs positively in C_j, the edge $c_j t_i$ exits; if it occurs negatively, the edge $c_j f_i$ exists.

To complete G' to G we have to add the vertices x_i and connect them to t_i, f_i. This is clearly possible without breaking planarity (picture placing x_i on the middle of the line segment $f_i t_i$ and moving it perpendicular by a small amount, then the edges $x_i f_i$ and $x_i t_i$ can be embedded without crossing other edges). The vertices y_2, y_3 can be placed anywhere; finally the vertex y_1 will break planarity (the embedding does not guarantee that the clause vertices lie on the outer face of the graph) and we conclude that G is indeed an apex-graph. ◀

▶ **Corollary 6.** Dominating Set *above* 2-*independence is* paraNP-*complete in graphs of maximum degree six.*

Proof. We reduce from (3,4)SAT (NP-hardness shown in [27]) in which every clause has size three and every variable occurs in at most four clauses. We use the above construction with one modification. Instead of connecting all clause vertices to one vertex we create a bounded-degree tree with the clause-vertices as its leaves.

We begin by partitioning the clause vertices in pairs P_i; if there is an odd number of vertices the last group will have three vertices. Then, for each group $P_i = \{c, c'\}$, we add two vertices s^i, r^i, connect the clause-vertices c and c' to s^i, and add the edge $s^i r^i$. We further add each s^i to our 2-independent set X and then create a set L_1 consisting of each r^i. Now we iteratively construct the next level of the tree, starting with $L := L_1$:

1. If $L = \{r^1\}$, create a single vertex y_3, connect it to r^1 and finish, otherwise proceed with the next step.
2. Partition the vertices in L into ℓ groups $\{l_i, r_i\}$ of pairs. If $|L|$ is odd, the last group will be a triple $\{l_i, c_i, r_i\}$ instead.
3. For each group, create a tree-gadget Γ^i, with vertices $\{a_1, a_2, s^i, r^i\}$ (and a_3 if the group contains a third vertex), and edges $l_i a_1, r_i a_2$, $(c_i a_3)$, $a_1 s^i$, $a_2 s^i$, $(a_3 s^i)$, and $s^i r^i$.
4. Add each s^i to X, let L now be the set of all r^i (for $1 \leq i \leq \ell$) and continue with Step 1.

Figure 1, on the right side, shows an example of this construction. We note that, in the last tree-gadget, s^i and r^i are the same vertices as y_1 and y_2 respectively in the figure. We conclude the construction by adding each x_i from the variable-gadgets to X and setting $p := |X|$. Notice that the only vertex not contained in $N[X]$ is y_3 and thus $R := \{y_3\}$.

Since each variable in (3,4)SAT can be in up to four clauses, the maximum degree for t_i and f_i is six. All clause-vertices have degree at most four and all other types of vertices have a degree not higher than that, hence the claimed degree-bound holds. It is left to show that ϕ is satisfiable iff there is a dominating set of size $p = |X|$ in the graph.

Let us assume that ϕ is satisfiable and fix one satisfying assignment I. We construct a dominating set as follows, beginning in the same way as in Lemma 4: if x_i is true under I, add t_i to D, otherwise add f_i. Since I satisfies every clause of ϕ the dominating set so far dominates every clause vertex and every variable gadget. Now, the remaining undominated vertices are the tree-vertices, and our remaining budget is $|X| - n$ which is equal to the amount of tree-gadgets. Since every clause vertex is already dominated we can, for each tree-gadget $\Gamma^i = \{a_1, a_2, s^i, r^i\}$, add r^i to the dominating set. This will dominate s^i and the corresponding a_1 or a_2 in the tree-gadget below it, hence we can dominate all vertices of the graph within the budget $|X|$.

In the other direction, assume that D is a dominating set of G of size $|X|$. Since y_3 is a pendant vertex we can assume that $y_2 \in D$. Thus, in the last tree-gadget r^i (y_2) is in the dominating set. This means that in order for a_1 and a_2 to be dominated, r^i in both tree-gadgets above has to be in the dominating set. This holds for all tree-gadgets, all the way up to the clause vertices. Since we now have one vertex per tree-gadget in the dominating set this leaves n vertices. Just as in Lemma 4, we note that, for each variable gadget, either t_i or f_i is in the dominating set. We know that the clause variables are not dominated by anything in the tree gadgets and thus must be dominated by the variable vertices. As stated in Lemma 4, the dominating vertices inside the variable gadgets encode a variable assignment I_D that satisfies ϕ and the claim follows. ◀

3 Above 3-independence: sparseness matters

3.1 W[2]-hardness in general graphs

In the following we present a result for Dominating Set above 3-independence, namely that it is W[2]-hard in general graphs. We show this by reduction from Colourful Dominating Set parametrised by the number of colours k:

Colourful Dominating Set parametrised by k

Input: A graph G with a vertex partition $C_0, C_1, \dots, C_k$.

Problem: Is there a set that dominates C_0 and uses exactly one vertex from each set $C_1, \dots, C_k$?

It is easy to verify that Colourful Dominating Set is W[2]-hard by reducing from Red-Blue Dominating Set: we copy the blue set k times and make each copy a colour set C_i, $1 \leq i \leq k$, and let C_0 be the red set.

▶ **Lemma 7.** Dominating Set *above r-independence is* W[2]*-hard for $r \geq 2$.*

Proof. Let $I = (G, C_0, C_1, \dots, C_k)$ be an instance of Colourful Dominating Set. We construct an instance (G', X, k) of Dominating Set above r-independence as follows (*cf.* Figure 2):

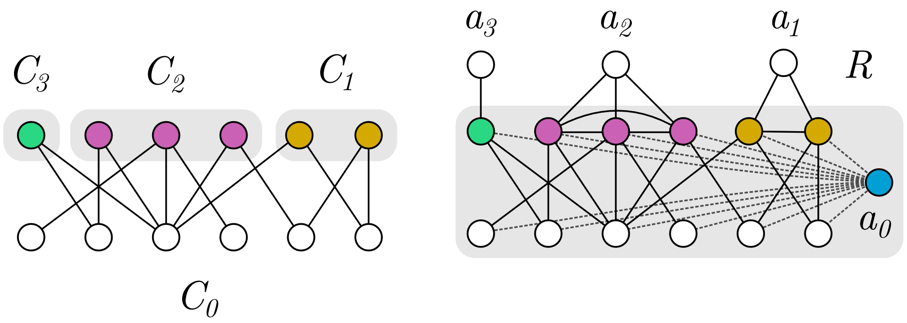


Figure 2 Sketch of reduction from Colourful Dominating Set to Dominating Set above r-independent set for $r \geq 2$. The set X contains only the vertex a_0, the remaining vertices a_i are all contained in the residual R.

1. Begin with G' equal to G; then
2. for each block C_i, $i \geq 1$, add edges to make $G[C_i]$ a complete graph and add an additional vertex a_i with neighbourhood C_i; then
3. add a vertex a_0 and connect it to all vertices in $C_0 \cup C_1 \cup \dots C_k$.

Let $X = \{a_0\}$ be the r-independent set (which it clearly is for any r) and thus $R = \{a_1, \dots, a_k\}$. Note that the graph G' trivially has a dominating set of size $k+1$; the set $\{a_0, a_1, \dots, a_k\}$. We now claim that I has a colourful dominating set of size k iff G' has a dominating set of size k.

Assume that I has a solution of size k and fix one such colourful dominating set D. Note that D in G' a) dominates all of C_0 (because it dominates C_0 in G) and b) dominates each set $C_i \cup \{a_i\}$, $i \geq 1$ (because D intersects each such C_i), and of course the vertex a_0. Hence, D is a dominating set of G' as well.

In the other direction, assume that D is a dominating set of G' of size k. Since each a_i, $i \geq 1$, is only connected to vertices in the corresponding set C_i, at least one vertex from each set $C_i \cup \{a_i\}$ needs to be in D. Since there are k such sets we conclude that D intersects each set $C_i \cup \{a_i\}$ in precisely one vertex. We can further modify any such solution to not take the a_i-vertices by taking an arbitrary vertex from C_i instead, thus assume that D has this form in the following. But then D is of course also a dominating set of size k for G, as claimed. ◀

3.2 Tractability in sparse graphs

In the following we present two positive results, namely that Dominating Set above 3-independence is fpt in nowhere dense classes and that it admits a polynomial kernel in bounded expansion classes. The algorithm further implies an XP algorithm in general graphs. The following annotated domination problem will occur as a subproblem:

Annotated Dominating Set

Input: A graph G, a subset $Y \subseteq V(G)$, a collection of vertex sets $R_1, \dots, R_\ell$, and an integer k.

Problem: Is there a set of size k that dominates $V(G) \setminus Y$ and contains at least one vertex in each R_i?

In the following algorithm we will group vertices of $N(X)$ according to their neighbourhood in R (or a subset of R). We will call those groups *R-neighbourhood classes*. We will write $\gamma(G, Y, \{R_1, \dots, R_\ell\})$ to denote the size of an optimal solution of Annotated Dominating Set.

▶ **Lemma 8.** Dominating set *above* 3*-independence can be solved in linear fpt-time in any graph class of bounded expansion.*

Proof. First we guess the intersection D_R of an optimal solution (should it exist) with R in $O(2^{|R|})$ time. Let $R' \subseteq R$ be those vertices of R that are not dominated by D_R. Define

$$\mathcal{R} := \{N(v) \cap R'\}_{v \in N(X)}$$

as the neighbourhoods induced in R' by vertices in $N(X)$. By Lemma 1, we have that $|\mathcal{R}| = O(|R'|)$ since G is from a class with bounded expansion (note that the partition into such neighbourhoods is computable in linear time using the partition-refinement data structure [19]). Accordingly, in time $O(2^{|\mathcal{R}|}) = 2^{O(|R'|)}$, we can guess a subset $\mathcal{R}' \subseteq \mathcal{R}$ such that an optimal solution covers exactly the neighbourhoods $\mathcal{R}'$ (if $\mathcal{R}'$ does not cover R' we abort this branch of the computation). We are now left with the task of choosing vertices from $N[X]$ to a) cover the neighbourhoods $\mathcal{R}'$ and b) dominate the vertices in $N[X]$ which are not dominated by D_R.

We introduce the following notation to ease our task: for a collection of R-neighbourhoods $\mathcal{S} \subseteq \mathcal{R}'$ and a set $Y \subseteq N(X)$, let $\mathcal{S}^{-1}(Y) := \{Y_i \subseteq Y \mid N(y) = S \text{ for all } y \in Y_i\}_{S \in \mathcal{S}}$. That is, $\mathcal{S}^{-1}(Y)$ contains those R'-neighbourhood classes in Y whose neighbourhood is contained in $\mathcal{S}$. Let $x_1, \ldots, x_\ell$ be an ordering of X and let $H_i := G[N[x_i]]$; we will describe a dynamic-programming algorithm over the ordering $x_1, \ldots, x_\ell$. Let $T_i[\mathcal{S}]$ be the minimum size of a partial solution in $N[\{x_1, \ldots, x_i\}]$ that covers the neighbourhoods $\mathcal{S} \subseteq \mathcal{R}'$ and together with D_R dominates all of $G[\bigcup_{1 \leq j \leq i} N[x_j]]$. We initialize $T_0[\mathcal{S}] := \infty$ for all $\emptyset \neq \mathcal{S} \subseteq \mathcal{R}'$ and $T_0[\emptyset] := 0$, then compute the following entries with the recurrence[1]

$$T_{i+1}[\mathcal{S}] := \min_{\mathcal{S}_1 \cup \mathcal{S}_2 = \mathcal{S}} \Big(T_i[\mathcal{S}_1] + \gamma\big(H_{i+1}, N(D_R) \cap N[x_{i+1}], \mathcal{S}_2^{-1}(N(x_{i+1}))\big)\Big).$$

Note that $\gamma(\ldots)$ is the minimum size of a set that dominates $N[x_i] \setminus N(D_R)$ while choosing at least one vertex from each member of $S_2^{-1}(N(x_i))$ (if $\emptyset \in S_2^{-1}(N(x_i)$ we assume that $\gamma(\ldots) = \infty$). The latter constraint corresponds to dominating the neighbourhoods of $\mathcal{S}_2$ in R' by using vertices from $N[x_i]$. Once the DP table T_ℓ has been computed, the size of an optimal solution is the value in $T_\ell[\mathcal{R}']$.

It remains to be noted that every neighbourhood graph H_i admits a dominating set of size one, hence the annotated dominating set has size at most $|\mathcal{S}_2| + 1 = O(|R|)$. Thus, the problem of finding an annotated dominating set is FO-expressible by a formula of size $O(|R|)$ and we can solve the subproblem of computing $\gamma(\ldots)$ in time $O(f(|R|) \cdot |N[x_i]|)$ for some function f using Proposition 2. As a result, we obtain a linear dependence on the input size (note that $|E| = O(|V|)$) in the running time and thus the problem is solvable in linear fpt-time. ◀

The same proof works for nowhere dense classes by applying Proposition 3 instead of Proposition 2:

▶ **Corollary 9.** Dominating set *above* 3*-independence can be solved in almost linear fpt-time in any nowhere dense class.*

We finally note that the algorithm described in the proof of Lemma 8 only needs a black-box fpt-algorithm for Annotated Dominating Set to run in fpt time, thus it is very likely that Dominating set above 3-independence is in FPT for other highly structured but not

[1] A proof for the correctness of the recurrence can be found in the full version available at `https://arxiv.org/abs/1906.09180`

necessarily sparse graph classes. We can run the same algorithm on general graphs to obtain an XP-algorithm using the bound $|\mathcal{R}| \leq 2^{|R'|}$ and a simple brute-force in XP-time on the Annotated Dominating Set subinstance during the DP.

▶ **Corollary 10.** Dominating set *above* 3*-independence is in* XP*.*

3.3 Kernelization in sparse graphs

Let us now set up the necessary machinery for the kernelization. A *boundaried graph* $^\circ G$ is a tuple (G, R) where G is a graph and $R \subseteq V(G)$ is the *boundary.* We also write $\partial^\circ G$ to denote the boundary. For a graph G and an induced subgraph H, the boundary $\partial_G H \subseteq V(H)$ are those vertices of H that have neighbours in $V(G) \setminus V(H)$. Thus for every subgraph H of G there is a naturally associated boundaried graph $\partial H = (H, \partial_G H)$.

For a boundaried graph $^\circ H$ and subsets $A, B \subseteq \partial^\circ H$ a set $D \subseteq V(^\circ H)$ is an (A,B)-dominator of $^\circ H$ if $D \cap \partial^\circ H = A$ and D dominates the set $(V(^\circ H) \setminus \partial^\circ H) \cup B$. We let $\mathbf{ds}(^\circ H, A, B)$ denote the size of a minimum (A,B)-dominator of $^\circ H$. A *replacement* for H is a boundaried graph (H', B) with $H[B] = H'[B]$. The operation of *replacing* H *by* H' *in* G, written as $G[H \to H']$, consist of removing the vertices $V(H) \setminus B$ from G, then adding H' to G with B in H' identified with B in G (we assume that the vertices $V(H') \setminus B$ do not occur in G).

▶ **Lemma 11.** *Let* (G, X, p) *be an instance of* Dominating Set *above* 3*-independence where* G *is from a bounded expansion class. Let* $x \in X$*,* $H_x = G[N^2[x]]$ *and* $R' = N^2[x] \cap R$*. Then, in fpt-time with parameter* $|R'|$*, we can compute a replacement* H'_x *for* H_x *of size* $O(4^{|R'|}|R'|)$ *such that* $\mathbf{ds}(G[H_x \to H'_x]) = \mathbf{ds}(G)$*. Moreover, the replacement* H'_x *is a subgraph of* H_x *and contains* x*.*

Proof. For every pair of subsets $A, B \in R'$ we compute a minimal (A,B)-dominator $S_{A,B}$ for $^\circ H_x = (H_x, R')$. Since this problem is expressible by an FO-formula of size $O(|R'|)$, we can employ Proposition 2 to compute the set $S_{A,B}$ in linear fpt-time with parameter $|R'|$. Note that $S_{A,B}$ will, besides the vertices in A, contain at most $|B|+1$ additional vertices, since $|B|$ vertices suffice to dominate B and the vertex x dominates all of $V(H_x) \setminus R$.

Let $S = \bigcup_{A,B \subseteq R'} S_{A,B} \cup R' \cup \{x\}$ be the union of all such computed solutions and the boundary. By construction, $|S| \leq (4^{|R'|}+1)|R'|+1$. Since all (A,B)-dominators live inside S, we can safely remove the edges from H_x that do not have any endpoint in S, call the resulting graph $\tilde{H}_x$. Let $Y := V(\tilde{H}_x) \setminus S$ and let $Y_1, \ldots, Y_t$ be a partition of Y into S-twin classes (thus all vertices in Y_i have the same neighbourhood in S and no other class has this neighbourhood). Note that Y is an independent set in $\tilde{H}_x$.

Let H'_x be obtained from $\tilde{H}_x$ by removing all but two representatives from every twin-class, denote those by $Y'_i \subseteq Y_i$. Clearly, every (A,B)-dominator of $\tilde{H}_x$ is still an (A,B)-dominator of H'_x, we need to proof the other direction. Let D' be an (A,B)-dominator of H'. If $D' \subseteq S$ then D' is still an (A,B)-dominator for H. The same holds if D' intersects every twin-class in at most one vertex: each such class either has size one, in which case it has size one in $\tilde{H}_x$ as well, or it has size two and the vertex not contained in D' is dominated by a vertex in S. In either case, D' dominates all of Y and thus is an (A,B)-dominator of $\tilde{H}_x$ and therefore of H_x. Thus, assume that D' fully contains some class Y'_i. Clearly, only one vertex of Y'_i is enough to dominate $N(Y'_i)$ (and potentially vertices in B), thus we can modify D' by picking the central vertex x instead to dominate the other vertex of Y'_i (which of course also dominates all of Y_i in $\tilde{H}_x$). This can, of course, only happen once, otherwise we would reduce the size of the supposedly minimal set D'. This leads us back to the previous case and we conclude that there exists an (A,B)-dominator of $\tilde{H}_x$ and thus H_x of equal size.

We conclude that $\mathbf{ds}({}^{\circ}H_x, A, B) = \mathbf{ds}({}^{\circ}H'_x, A, B)$ for every choice of $A, B \subseteq R'$; which implies that $\mathbf{ds}(G[H_x \to H'_x]) = \mathbf{ds}(G)$. Finally, by the twin-class lemma, $|H'_x| = O(|S|) = O(4^{|R'|}|R'|)$ since τ is a constant in bounded expansion classes. ◀

▶ **Lemma 12.** *Let (G, X, p) be an instance of* Dominating Set *above 3-independence where G is from a bounded expansion class. Let $R' \subseteq R$ be a subset and let $X' \subseteq X$ be those vertices $x \in X$ with $N^2(x) \cap R = R'$. Let H be the induced subgraph on $R' \cup \bigcup_{x \in X'} N[x]$. Assuming X' is not empty we can, in fpt-time with parameter $|R'|$, compute a replacement H' for H of size $O(|R'|^{2|R'|+2}4^{|R'|})$ alongside an offset c such that $\mathbf{ds}(G[H' \to H]) = \mathbf{ds}(G) - c$. Moreover, the replacement H' is a subgraph of H.*

Proof. Let $X' := \{x_1, \ldots, x_\ell\}$ and define the graphs $H_i := G[N^2[x_i]]$ for $1 \le i \le \ell$. We first we apply Lemma 11 to replace every subgraph H_i by a subgraph H'_i of size $O(4^{|R'|}|R'|)$ in linear fpt-time with parameter R'. For simplicity, let us call the resulting graph G and relabel the graphs H'_i to H_i (note that Lemma 11 ensures that dominating set size does not change and that X is still a 3-independent set of the resulting graph).

For every $x_i \in X'$ we compute a *characteristic vector* χ_i indexed by pairs of subsets of R' with the following semantic: for $A, B \subseteq R'$ we set $\chi_i[A, B] = \mathbf{ds}({}^{\circ}H_i, A, B)$. If $\chi_i[A, B]$ is larger than $|A| + |B| + 1$ we simply set $\chi_i[A, B] = \infty$.

Note that the constraint subproblem to compute $\chi_i[A, B]$ is FO-expressible, by a formula of size $O(|R'|)$, thus we can compute the vectors χ_i for $1 \le i \le \ell$ in linear time fpt-time with parameter $O(|R'|)$ using Proposition 2. Let $\equiv_\chi$ be the equivalence relation over the graphs H_i defined as $H_i \equiv_\chi H_j \iff \chi_i = \chi_j$ and let $\mathcal{H} := \{H_i\}_{1 \le i \le \ell} / \equiv_\chi$ be the corresponding partition into equivalence classes under $\equiv_\chi$. Note that $|\mathcal{H}| \le (|R'| + 1)^{2|R'|}$ since that is the number of possible characteristic vectors.

The construction of H' is now simple: in every equivalence class $\mathcal{C} \in \mathcal{H}$ we select $\min\{|\mathcal{C}|, |R|\}$ subgraphs and remove the rest; clearly H' is a subgraph of H of the claimed size. We let the offset c to be equal to the number of subgraphs removed in this way.

We are left to show that $\mathbf{ds}(G[H' \to H]) = \mathbf{ds}(G) - c$. Consider any minimal dominating set D for G. We call a graph H_i *interesting* under D if D intersects H_i in any vertex besides x_i.

▷ Claim 13. There exists a solution D' of size equal to D under which at most $|R'|$ graphs per equivalence class $\mathcal{C} \in \mathcal{H}$ are interesting.

Proof. Consider any such class $\mathcal{C} \in \mathcal{H}$. If $|\mathcal{C}| \le |R'|$ we are done, so assume otherwise. Let $R'' \subseteq R'$ be the set of vertices that are dominated through vertices in graphs contained in $\mathcal{C}$ and select up to $|R''|$ many graphs that together already dominate R''; we let D' to be equivalent to D on these graphs. Any graph $H \in \mathcal{C}$ *not* selected in this way only needs to dominate itself and we add its centre vertex $V(H) \cap X$ to D'. Since every graph needs to intersect any dominating set in at least one vertex, $|D'| \le |D|$ and since D is minimal we must have $|D'| = |D|$. Finally, only the $|R''| \le |R'|$ selected graphs are interesting under D', as claimed. ◁

Thus, let us assume in the following that D is such a minimal solution under which at most $|R'|$ graphs per class $\mathcal{C} \in \mathcal{H}$ are interesting. For such a solution D of H, we construct a solution D' of H' of size $|D| - c$ as follows. For a class $\mathcal{C} \in \mathcal{H}$, let $\mathcal{C}' \subseteq \mathcal{C}$ be those graphs that are contained in H' and let $\mathcal{I} \subseteq \mathcal{C}$ be the graphs that are interesting under D. Since $|\mathcal{C}| \le |\mathcal{C}'|$, we can pair every graph $H_i \in \mathcal{C}$ with a graph $H_{i'} \in \mathcal{C}'$. Fix such a pair $H_i, H_{i'}$, let $A = R' \cap D$ and let B be those vertices of R' that are exclusively dominated vertices in H_i. Since $\chi_i[A, B] = \chi_{i'}[A, B]$, there must exist a set of $|D \cap V(H_i)|$ vertices in $H_{i'}$ that dominate $H_{i'}$ and B. If we repeat this construction for every graph $H_i \in \mathcal{C}$ with their respective pair in $\mathcal{C}'$ and then pick the centre vertex $X \cap V(H_j)$ for all $H_j \in \mathcal{C}' \setminus \mathcal{C}$, then the resulting set D' has size $\leq |D| - c$ and dominates all of H'.

The same proof works in reverse if we start with a dominating set D' of H' to construct a dominating set D of H with $|D| \leq |D'| + c$; thus we conclude that $\mathbf{ds}(G[H' \to H]) = \mathbf{ds}(G) - c$ and the claim follows. ◀

▶ **Theorem 14.** Dominating Set *above* 3*-independence has a linear kernel in bounded expansion graphs.*

Proof. Let (G, X, p) be the input instance and let $x_1, \ldots, x_\ell$ be the members of the 3-independent set X. We define the graphs $H_i := G[N^2[x_i]]$ and their respective R-neighbours $R_i := N^2[x_i] \cap R$ for $1 \leq i \leq \ell$. Let τ be as in Lemma 1. We partition the graphs $\{H_i\}_{1 \leq i \leq \ell}$ into two sets $\mathcal{L}$, $\mathcal{S}$ where $H_i \in \mathcal{L}$ iff $|R_i| > 2\tau$; and $\mathcal{S}$ contains all remaining graphs.

Let us first reduce $\mathcal{S}$. Let $\mathcal{R} := \{R_i \mid H_i \in \mathcal{S}\}$ be the R-neighbourhoods of the graphs collected in $\mathcal{S}$. By the twin class lemma, $|\mathcal{R}| \leq (4^\tau + 2\tau)|R|$, however, for each member $R' \in \mathcal{R}$ we might have many graphs in $\mathcal{S}$ that intersect R in precisely this set R'.

Fix $R' \in \mathcal{R}$ for now and let $\mathcal{S}[R']$ be those graphs of $\mathcal{S}$ that intersect R in R'. Let $H_{R'} := G[\bigcup_{H \in \mathcal{S}[R']} V(H)]$ be the joint graph of the subgraphs in $\mathcal{S}[R']$. Since $|R'| \leq 2\tau$, and τ is a constant depending only on the graph class, we can apply Lemma 12 to compute a replacement $H'_{R'}$ of size $O(|R'|^{2|R'|+2} 4^{|R'|})$ with offset c in polynomial time. We apply the replacement $G[H_{R'} \to H'_{R'}]$ and decrease p by c. Repeating this procedure for all $R' \in \mathcal{R}$ yields a graph G_1 (a subgraph of G) in which the small graphs $\mathcal{S}$ in total contain at most $|\mathcal{R}| \cdot O(|\tau|^{2\tau+2} 4^\tau) = O(|R|)$ vertices, a 3-independent set $X' \subseteq X$ of G_1, and a new input p'. This concludes our reduction for $\mathcal{S}$.

Let us now deal with $\mathcal{L}$ in G_1. By the twin class lemma, $|\mathcal{L}| \leq 2\tau|R|$. But then $|X'|$ has size bounded in $O(|R|)$ and we conclude that the size of a minimal dominating set for G_1 is bounded by $O(|R|)$, hence we assume that p' is bounded by $O(|R|)$ (otherwise the instance is positive and we can output a trivial instance). We now apply the existing linear kernel [6] for Dominating Set to G_1. The output of the kernelization is a subgraph of the original graph, hence we collect the remaining vertices of the 3-independent set $X'' \subseteq X'$ in order to output a well-formed instance (G'', X'', p''). This concludes the proof. ◀

4 Above 4-independence: simple domination

In the case where the lower-bound set X is 4-independent we now have that for distinct $x, x' \in X$ it holds that $N^2(x) \cap N^2(x') = \emptyset$, thus for each $x \in X$ the set $N^2(x) \cap R$ can only be dominated from R and $N(x)$. Let us call an instance (G, X) *reduced* if for every $x \in X$ the intersection $N^2(x) \cap R$ is non-empty. We can easily pre-process our input instance to enforce this property: if such an x would exist we can simply remove $N[x]$ from G to obtain an equivalent instance. In a reduced instance the parameter $|R|$ is necessarily big compared to $|X|$:

▶ **Observation 15.** *For every reduced instance* (G, X) *of* Dominating Set *above* 4*-independence it holds that* $|R| \geq |X|$.

▶ **Corollary 16.** *Let* $\mathcal{G}$ *be a graph class for which* Dominating Set *is in* FPT*. Then* Dominating Set *above* 4*-independence is in* FPT *for* $\mathcal{G}$ *as well.*

▶ **Corollary 17.** *Let $\mathcal{G}$ be a graph class for which* Dominating Set *admits a polynomial kernel. Then* Dominating Set *above 4-independence admits a polynomial kernel for $\mathcal{G}$ as well.*

On the other hand, we showed in Section 3.1 that the problem remains W[2]-hard for any $r \geq 2$ in general graphs.

5 Conclusion

We considered Dominating Set parametrised by the residual of a given r-independent set and investigated how the value of r and the choice of input graph classes affect its tractability. We observed that the tractability does improve from $r = 2$ to $r = 3$ as it goes from being paraNP-complete to 'merely' W[2]-hard and at least admits an XP-algorithm. Larger values of r, however, do not increase the tractability as the problem becomes essentially equivalent to Dominating Set.

If we consider sparse classes (bounded expansion and nowhere dense), the improvement in tractability from $r = 2$ to $r = 3$ is much more pronounced; changing from paraNP-complete to FPT and even admitting a linear kernel in bounded expansion classes. We very much believe that the kernel can be extended to nowhere dense classes, but leave that quite technical task as an open question.

References

1 J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed Parameter Algorithms for DOMINATING SET and Related Problems on Planar Graphs. *Algorithmica*, 33(4):461–493, 2002.

2 J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for Dominating Set. *JACM*, 51:363–384, 2004.

3 N. Alon, G. Gutin, E. Jung Kim, S. Szeider, and A. Yeo. Solving MAX-r-SAT Above a Tight Lower Bound. *Algorithmica*, 61(3):638–655, 2011.

4 N. Alon and S. Gutner. Linear Time Algorithms for Finding a Dominating Set of Fixed Size in Degenerated Graphs. *Algorithmica*, 54(4):544–556, 2009.

5 M. Cygan, M. Pilipczuk, M. Pilipczuk, and J.O. Wojtaszczyk. On multiway cut parameterized above lower bounds. *TOCT*, 5(1):3, 2013.

6 P.G. Drange, M. Sortland Dregi, F. V. Fomin, S. Kreutzer, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, F. Reidl, F. Sánchez Villaamil, S. Saurabh, S. Siebertz, and S. Sikdar. Kernelization and Sparseness: the Case of Dominating Set. In *STACS*, volume 47 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

7 Z. Dvořák. On distance r-dominating and $2r$-independent sets in sparse graphs. *CoRR*, abs/1710.10010, 2017. arXiv:1710.10010.

8 Zdenek Dvorak, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013.

9 K. Eickmeyer, A. C. Giannopoulou, S. Kreutzer, O. Kwon, M. Pilipczuk, R. Rabinovich, and S. Siebertz. Neighborhood Complexity and Kernelization for Nowhere Dense Classes of Graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 63:1–63:14, 2017.

10 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and Kernels. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 503–510, 2010. doi:10.1137/1.9781611973075.43.

11 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 Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 82–93. SIAM, 2012. `doi:10.1137/1.9781611973099.7`.

12 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Linear kernels for (connected) dominating set on graphs with excluded topological subgraphs. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 92–103. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. `doi:10.4230/LIPIcs.STACS.2013.92`.

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. *J. Comput. Syst. Sci.*, 84:219–242, 2017.

14 M. Grohe, S. Kreutzer, and Siebertz S. Deciding First-Order Properties of Nowhere Dense Graphs. *J. ACM*, 64(3):17:1–17:32, 2017.

15 G. Gutin, E.J. Kim, M. Lampis, and V. Mitsou. Vertex cover problem parameterized above and below tight bounds. *Theory of Computing Systems*, 48(2):402–410, 2011.

16 G. Gutin, E.J. Kim, M. Mnich, and A. Yeo. Betweenness parameterized above tight lower bound. *Journal of Computer and System Sciences*, 76(8):872–878, 2010.

17 G. Gutin, A. Rafiey, S. Szeider, and A. Yeo. The linear arrangement problem parameterized above guaranteed value. *Theory of Computing Systems*, 41(3):521–538, 2007.

18 G. Gutin, L. van Iersel, M. Mnich, and A. Yeo. All Ternary Permutation Constraint Satisfaction Problems Parameterized above Average Have Kernels with Quadratic Numbers of Variables. In *Proc. of the 18th ESA*, volume 6346 of *Lecture Notes in Computer Science*, pages 326–337. Springer, 2010.

19 M. Habib, C. Paul, and L. Viennot. A Synthesis on Partition Refinement: A Useful Routine for Strings, Graphs, Boolean Matrices and Automata. In *STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science, Paris, France, February 25-27, 1998, Proceedings*, volume 1373 of *Lecture Notes in Computer Science*, pages 25–38. Springer, 1998. `doi:10.1007/BFb0028546`.

20 D. Lichtenstein. Planar Formulae and Their Uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

21 D. Lokshtanov, NS Narayanaswamy, V. Raman, MS Ramanujan, and S. Saurabh. Faster parameterized algorithms using linear programming. *TALG*, 11(2):15, 2014.

22 M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999.

23 M. Mahajan, V. Raman, and S. Sikdar. Parameterizing above or below guaranteed values. *Journal of Computer and System Sciences*, 75(2):137–153, 2009.

24 J. Nešetřil and P. Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

25 G. Philip, V. Raman, and S. Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Transactions on Algorithms*, 9(1), 2012.

26 F. Reidl. *Structural sparseness and complex networks*. Dr., Aachen, Techn. Hochsch., Aachen, 2016. Aachen, Techn. Hochsch., Diss., 2015.

27 C. A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

Reducing the Domination Number of Graphs via Edge Contractions

Esther Galby
Department of Informatics, University of Fribourg, Fribourg, Switzerland
esther.galby@unifr.ch

Paloma T. Lima
Department of Informatics, University of Bergen, Bergen, Norway
paloma.lima@uib.no

Bernard Ries
Department of Informatics, University of Fribourg, Fribourg, Switzerland
bernard.ries@unifr.ch

Abstract

In this paper, we study the following problem: given a connected graph G, can we reduce the domination number of G by at least one using k edge contractions, for some fixed integer $k \geq 0$? We show that for $k \leq 2$, the problem is coNP-hard. We further prove that for $k = 1$, the problem is W[1]-hard parameterized by the size of a minimum dominating set plus the mim-width of the input graph, and that it remains NP-hard when restricted to P_9-free graphs, bipartite graphs and $\{C_3, \dots, C_\ell\}$-free graphs for any $\ell \geq 3$. Finally, we show that for any $k \geq 1$, the problem is polynomial-time solvable for P_5-free graphs and that it can be solved in FPT-time and XP-time when parameterized by tree-width and mim-width, respectively.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases domination number, blocker problem, graph classes

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.41

1 Introduction

In a graph modification problem, we are usually interested in modifying a given graph G, via a small number of operations, into some other graph G' that has a certain desired property. This property often describes a certain graph class to which G' must belong. Such graph modification problems allow to capture a variety of classical graph-theoretic problems. Indeed, if for instance only k vertex deletions are allowed and G' must be a stable set or a clique, we obtain the Stable Set or Clique problem, respectively.

Now, instead of specifying a graph class to which G' should belong, we may ask for a specific graph parameter π to decrease. In other words, given a graph G, a set $\mathcal{O}$ of one or more graph operations and an integer $k \geq 1$, the question is whether G can be transformed into a graph G' by using at most k operations from $\mathcal{O}$ such that $\pi(G') \leq \pi(G) - d$ for some *threshold* $d \geq 0$. Such problems are called *blocker problems* as the set of vertices or edges involved can be viewed as "blocking" the parameter π. Notice that identifying such sets may provide important information about the structure of the graph G.

Blocker problems have been well studied in the literature (see for instance [1, 2, 3, 4, 6, 9, 14, 15, 16, 17, 19]) and their relations to other well-known graph problems have been presented (see for instance [9, 15]). So far, the literature mainly focused on the following graph parameters: the chromatic number, the independence number, the clique number, the matching number and the vertex cover number. Furthermore, the set $\mathcal{O}$ consisted of a single graph operation, namely either vertex deletion, edge contraction, edge deletion or edge

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 41; pp. 41:1–41:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

addition. Since these blocker problems are usually NP-hard in general graphs, a particular attention has been paid to their computational complexity when restricted to special graph classes.

In this paper, we focus on another parameter, namely the domination number γ, and we restrict $\mathcal{O}$ to a single graph operation, the edge contraction. More specifically, let $G = (V, E)$ be a graph. The *contraction* of an edge $uv \in E$ removes vertices u and v from G and replaces them by a new vertex that is made adjacent to precisely those vertices that were adjacent to u or v in G (without introducing self-loops nor multiple edges). We say that a graph G can be *k-contracted* into a graph G', if G can be transformed into G' by a sequence of at most k edge contractions, for an integer $k \geq 1$ (note that contracting an edge cannot increase the domination number). We will be interested in the following problem, where $k \geq 1$ is a fixed integer.

k-Edge Contraction(γ)
Instance: A connected graph $G = (V, E)$
Question: Can G be k-edge contracted into a graph G' such that $\gamma(G') \leq \gamma(G) - 1$?

In other words, we are interested in a blocker problem with parameter γ, graph operations set $\mathcal{O}$ = {edge contraction} and threshold $d = 1$. Notice that if $\gamma(G) = 1$, that is, G contains a dominating vertex, then G is always a No-instance for k-Edge Contraction(γ). Reducing the domination number using edge contractions was first considered in [13]; given a graph $G = (V, E)$, the authors denote by $ct_\gamma(G)$ the minimum number of edge contractions required to transform G into a graph G' such that $\gamma(G') \leq \gamma(G) - 1$ and prove that for a connected graph G such that $\gamma(G) \geq 2$, we have $ct_\gamma(G) \leq 3$. It follows that a connected graph G with $\gamma(G) \geq 2$ is always a Yes-instance of k-Edge Contraction(γ), if $k \geq 3$. The authors [13] further give necessary and sufficient conditions for $ct_\gamma(G)$ to be equal to 1, respectively 2.

▶ **Theorem 1** ([13]). *For a connected graph G, the following holds.*

(i) $ct_\gamma(G) = 1$ *if and only if there exists a minimum dominating set in G that is not a stable set.*

(ii) $ct_\gamma(G) = 2$ *if and only if every minimum dominating set in G is a stable set and there exists a dominating set D in G of size $\gamma(G) + 1$ such that $G[D]$ contains at least two edges.*

To the best of our knowledge, a systematic study of the computational complexity of k-Edge Contraction(γ) has not yet been attempted in the literature. We here initiate such a study as it has been done for other parameters and several graph operations. Our paper is organised as follows[1]. In Section 2, we present definitions and notations that are used throughout the paper. In Section 3, we prove the (co)NP-hardness of k-Edge Contraction(γ) for $k = 1, 2$. We further show that 1-Edge Contraction(γ) is W[1]-hard parameterized by the size of a minimum dominating set plus the mim-width of the input graph, and that it remains NP-hard when restricted to P_9-free graphs, bipartite graphs and $\{C_3, \ldots, C_l\}$-free graphs for any $l \geq 3$. Finally, we present in Section 4 some positive results; in particular, we show that for any $k \geq 1$, k-Edge Contraction(γ) is polynomial-time solvable for P_5-free graphs and that it can be solved in FPT-time and XP-time when parameterized by tree-width and mim-width, respectively.

[1] Proofs marked by ♠ are omitted due to space constraints.

2 Preliminaries

Throughout the paper, we only consider finite, undirected, connected graphs that have no self-loops or multiple edges. We refer the reader to [8] for any terminology and notation not defined here and to [7] for basic definitions and terminology regarding parameterized complexity.

Let $G = (V, E)$ be a graph and let $u \in V$. We denote by $N_G(u)$, or simply $N(u)$ if it is clear from the context, the set of vertices that are adjacent to u i.e., the *neighbors* of u, and let $N[u] = N(u) \cup \{u\}$. Two vertices $u, v \in V$ are said to be *true twins* (resp. *false twins*), if $N[u] = N[v]$ (resp. if $N(u) = N(v)$).

For a family $\{H_1, \dots, H_p\}$ of graphs, G is said to be $\{H_1, \dots, H_p\}$*-free* if G has no induced subgraph isomorphic to a graph in $\{H_1, \dots, H_p\}$; if $p = 1$ we may write H_1-free instead of $\{H_1\}$-free. For a subset $V' \subseteq V$, we let $G[V']$ denote the subgraph of G *induced* by V', which has vertex set V' and edge set $\{uv \in E \mid u, v \in V'\}$.

We denote by $d_G(u, v)$, or simply $d(u, v)$ if it is clear from the context, the length of a shortest path from u to v in G. Similarly, for any subset $V' \subseteq V$, we denote by $d_G(u, V')$, or simply $d(u, V')$ if it is clear from the context, the minimum length of a shortest path from u to some vertex in V' i.e., $d(u, V') = \min_{v \in V'} d(u, v)$.

For a vertex $v \in V$, we write $G - v = G[V \setminus \{v\}]$ and for a subset $V' \subseteq V$ we write $G - V' = G[V \setminus V']$. For an edge $e \in E$, we denote by $G \backslash e$ the graph obtained from G by contracting the edge e. The k*-subdivision* of an edge uv consists in replacing it by a path u-v_1-…-v_k-v, where $v_1, \dots, v_k$ are new vertices.

For $n \geq 1$, the path and cycle on n vertices are denoted by P_n and C_n respectively. A graph is *bipartite* if every cycle contains an even number of vertices.

A subset $S \subseteq V$ is called a *stable set* of G if any two vertices in S are nonadjacent; we may also say that S is *stable*. A subset $D \subseteq V$ is called a *dominating set*, if every vertex in $V \setminus D$ is adjacent to at least one vertex in D; the *domination number* $\gamma(G)$ is the number of vertices in a minimum dominating set. For any $v \in D$ and $u \in N[v]$, v is said to *dominate* u (in particular, v dominates itself); furthermore, u is a *private neighbor of* v *with respect to* D if u has no neighbor in $D \backslash \{v\}$. We say that D *contains an edge* (or more) if the graph $G[D]$ contains an edge (or more). The Dominating Set problem is to test whether a given graph G has a dominating set of size at most ℓ, for some given integer $\ell \geq 0$.

3 Hardness results

In this section, we present hardness results for the k-Edge Contraction(γ) problem. Recall that for $k \geq 3$, the problem is trivial; we show that for $k = 1, 2$, it becomes (co)NP-hard. To this end, we introduce the following problem.

Contraction Number(γ,k)

Instance: A connected graph $G = (V, E)$.

Question: Is $ct_\gamma(G) = k$?

▶ **Theorem 2.** Contraction Number(γ,3) *is* NP*-hard.*

Proof. We reduce from 1-in-3 Positive 3-Sat, where each variable occurs only positively, each clause contains exactly three positive literals, and we want a truth assignment such that each clause contains exactly one true variable. This problem is known to be NP-complete [11]. Given an instance Φ of this problem, with variable set X and clause set C, we construct an equivalent instance G_Φ of Contraction Number(γ,3) as follows. For any variable $x \in X$, we introduce a copy of C_3, which we denote by G_x, with two distinguished *truth vertices* T_x and F_x (see Fig. 1); in the following, the third vertex of G_x is denoted by u_x. For any clause $c \in C$ containing variables x_1, x_2 and x_3, we introduce the gadget G_c depicted in Fig. 1 (where it is connected to the corresponding variable gadgets). The vertex set of the clique K_c corresponds to the set of subsets of size 1 of $\{x_1, x_2, x_3\}$ (hence the notation); for any $i \in \{1,2,3\}$, the vertex x_i (resp. x'_i) is connected to every vertex $v_S \in K_c$ such that $x_i \notin S$ (resp. $x_i \in S$). Finally, for $i = 1,2,3$, we add an edge between t_i (resp. x'_i) and the truth vertex T_{x_i} (resp. F_{x_i}). Our goal now is to show that Φ is satisfiable if and only if $ct_\gamma(G_\Phi) = 3$. In the remainder of the proof, given a clause $c \in C$, we denote by x_1, x_2 and x_3 the variables occuring in c and thus assume that t_i (resp. x'_i) is adjacent to T_{x_i} (resp. F_{x_i}) for $i \in \{1,2,3\}$. Let us first start with some easy observations.

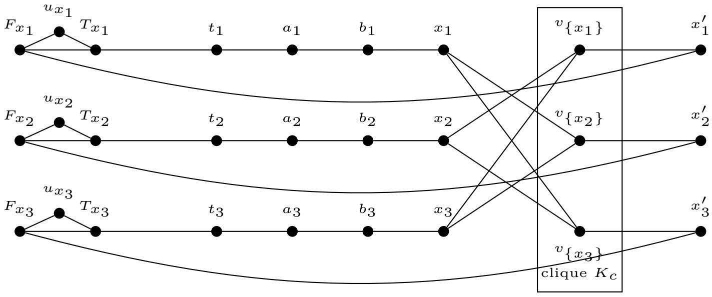


Figure 1 The gadget G_c together with G_{x_i}, $i = 1,2,3$, for a clause $c \in C$ containing variables x_1, x_2 and x_3 (the rectangle indicates that the corresponding set of vertices induces a clique).

▶ **Observation 1.** *Let D be a dominating set of G_Φ. Then for any $x \in X$, $|D \cap V(G_x)| \geq 1$ and for any $c \in C$, $|D \cap V(G_c)| \geq 4$. In particular, $|D| \geq |X| + 4|C|$.*

Clearly, for any $x \in X$, $|D \cap V(G_x)| \geq 1$ since u_x must be dominated. Also, in order to dominate vertices a_1, a_2, a_3 and $v_{\{x_1\}}$ in some gadget G_c, we need at least 4 distinct vertices, since their neighborhoods are pairwise disjoint and so, $|D \cap V(G_c)| \geq 4$, for any $c \in C$.

▶ **Observation 2.** *Let D be a dominating set of G_Φ. For any clause gadget G_c and $i \in \{1,2,3\}$, $D \cap \{a_i, b_i, x_i\} \neq \emptyset$.*

This immediately follows from the fact that every vertex b_i needs to be dominated and its neighbors are a_i and x_i for $i \in \{1,2,3\}$.

▶ **Observation 3.** *Let D be a dominating set of G_Φ. For any clause gadget G_c, if $|D \cap V(G_c)| = 4$, then $D \cap \{t_i, x'_i\} = \emptyset$ and $|D \cap \{a_i, b_i, x_i\}| = 1$, for any $i \in \{1,2,3\}$.*

If $t_i \in D$ for some $i \in \{1,2,3\}$, then it follows from Observation 2 that $|D \cap \{a_j, b_j, x_j\}| = 1$ for any $j \in \{1,2,3\}$. This implies that at least two vertices among x_1, x_2 and x_3 belong to D for otherwise there would exist $j \in \{1,2,3\}$ such that $v_{\{x_j\}}$ is not dominated. In particular, there must exist $j \neq i$ such that $x_j \in D$; but then, a_j is not dominated. Similarly, if $x'_i \in D$ for some $i \in \{1,2,3\}$, it follows from Observation 2 that $|D \cap \{a_j, b_j, x_j\}| = 1$ for any $j \in \{1,2,3\}$. But then, in order to dominate the vertices of K_c, either $x_i \in D$ in which case a_i is not dominated; or $\{x_j, j \neq i\} \subset D$ and a_j with $j \neq i$, is not dominated.

Now suppose that $|D \cap \{a_i, b_i, x_i\}| \geq 2$ for some $i \in \{1,2,3\}$. Then by Observation 2, we conclude that $|D \cap \{a_k, b_k, x_k\}| = 1$ for $k \neq i$ and $|D \cap \{a_i, b_i, x_i\}| = 2$. This implies that

$D \cap V(K_c) = \emptyset$ for otherwise we would have $|D \cap V(G_c)| \geq 5$. But then, since $x'_i \notin D$, D must contain at least two vertices among x_1, x_2 and x_3 in order to dominate the vertices of K_c; in particular, there exists $j \neq i$ such that $x_j \in D$ and so, a_j is not dominated.

▶ **Observation 4.** *Let D be a minimum dominating set of G_Φ and suppose that $ct_\gamma(G_\Phi) = 3$. Then for any vertices $u, v \in D$, we have $d(u, v) \geq 3$.*

Indeed, if u, v are adjacent, we conclude by Theorem 1(i) that $ct_\gamma(G_\Phi) = 1$; and if u, v are at distance 2 then $D \cup \{w\}$, where w is the vertex on a shortest path from u to v, contains two edges and we conclude by Theorem 1(ii) that $ct_\gamma(G_\Phi) = 2$.

▶ **Observation 5.** *Let D be a minimum dominating set of G_Φ and suppose that $ct_\gamma(G_\Phi) = 3$. Then for any clause gadget G_c and $i \in \{1, 2, 3\}$, $a_i \in D$ if and only if $T_{x_i} \notin D$.*

This readily follows from Observation 4. Further note that we may assume that for any $i \in \{1, 2, 3\}$, $a_i \in D$ if and only if $F_{x_i} \in D$; $T_{x_i} \notin D$ is equivalent to $\{F_{x_i}, u_{x_i}\} \cap D \neq \emptyset$ and if $T_{x_i} \notin D$, we may always replace D by $(D \backslash \{u_{x_i}\}) \cup \{F_{x_i}\}$.

▶ **Observation 6.** *Let D be a minimum dominating set of G_Φ and suppose that $ct_\gamma(G_\Phi) = 3$. Then for any clause gadget G_c, $|D \cap \{a_1, a_2, a_3\}| \leq 2$.*

If it weren't the case then, by Observation 4, no x_i or b_i $(i = 1, 2, 3)$ would belong to D. But since x_1, x_2 and x_3 must be dominated, it follows that $D \cap V(K_c) \neq \emptyset$ and by Observation 5, we conclude that D contains two vertices at distance two (namely, $v_{\{x_i\}} \in D \cap V(K_c)$ and F_{x_i} for some $i \in \{1, 2, 3\}$), which contradicts Observation 4.

▶ **Observation 7.** *Let D be a minimum dominating set of G_Φ and suppose that $ct_\gamma(G_\Phi) = 3$. Then for any clause gadget G_c, $|D \cap \{b_1, b_2, b_3\}| \leq 1$.*

Indeed, if we assume, without loss of generality, that $b_1, b_2 \in D$, then by Observation 4, $D \cap V(K_c) = \emptyset$. It then follows from Observation 4 that $x'_3 \in D$ for otherwise $V_{\{x_3\}}$ would not be dominated. But then $D \cap V(G_{x_3}) = \emptyset$ by Observation 4, which contradicts Observation 1.

▶ **Claim 1.** *$\gamma(G_\Phi) = |X| + 4|C|$ if and only if $ct_\gamma(G_\Phi) = 3$.*

Assume that $\gamma(G_\Phi) = |X| + 4|C|$ and consider a minimum dominating set D of G_Φ. We first show that D is a stable set which would imply that $ct_\gamma(G_\Phi) > 1$ (see Theorem 1(i)). First note that Observation 1 implies that $|D \cap V(G_x)| = 1$ and $|D \cap V(G_c)| = 4$, for any variable $x \in X$ and any clause $c \in C$. It then follows from Observation 3 that no truth vertex is dominated by some vertex t_i or x'_i in some clause gadget G_c with $i \in \{1, 2, 3\}$; in particular, this implies that there can exist no edge in D having one endvertex in some gadget G_x $(x \in X)$ and the other in some gadget G_c $(c \in C)$. Hence, it is enough to show that for any $c \in C$, $D \cap V(G_c)$ is a stable set.

Now consider a clause gadget G_c. It follows from Observation 3 that if there exists $i \in \{1, 2, 3\}$ such that $a_i \notin D$ then $b_i \in D$ since a_i must be dominated (also note that by Observation 3, if $a_i \in D$ then $b_i \notin D$). Hence, for any $i \in \{1, 2, 3\}$, exactly one of a_i and b_i belongs to D. But then, by Observation 3 and since $|D \cap V(G_c)| = 4$, we immediately conclude that $D \cap V(G_c)$ is a stable set and so, D is a stable set.

Now, suppose to the contrary that $ct_\gamma(G_\Phi) = 2$ i.e., there exists a dominating set D' of G_Φ of size $\gamma(G_\Phi) + 1$ containing two edges e and e' (see Theorem 1(ii)). First assume that there exists $x \in X$ such that $|D' \cap V(G_x)| = 2$. Then, for any $x' \neq x$, $|D' \cap V(G_{x'})| = 1$; and for any $c \in C$, $|D' \cap V(G_c)| = 4$ which by Observation 3 implies that $\{t_i, x'_i\} \cap D' = \emptyset$ for any $i \in \{1, 2, 3\}$. Since as shown previously, $D' \cap V(G_c)$ is then a stable set, it follows that D' contains at most one edge, a contradiction.

Thus, there must exist some $c \in C$ such that $|D' \cap V(G_c)| = 5$. We then claim that $\{a_1, a_2, a_3\} \not\subset D'$. Indeed, since $x_1, x_2, x_3, v_{\{x_1\}}, v_{\{x_2\}}$ and $v_{\{x_3\}}$ must be dominated, $D' \cap V(K_c) \neq \emptyset$ (otherwise, at least three additional vertices of G_c would be required to dominate x_1, x_2 and x_3), say $v_{\{x_1\}} \in D'$ without loss of generality. But then, $|N[x_1] \cap D'| = 1$ as x_1 must be dominated and $|D' \cap V(G_c)| = 5$ and so, D' contains at most one edge. Therefore, there must exist $i \in \{1,2,3\}$ such that $a_i \notin D'$, say $a_1 \notin D'$ without loss of generality. Then, since a_1 must be dominated, either $t_1 \in D'$ or $b_1 \in D'$.

Assume first that t_1 belongs to D' (note that $\{b_1, x_1\} \cap D' \neq \emptyset$ by Observation 2). We then claim that either e or e' has an endvertex in $\{a_j, b_j, x_j\}$ for some $j \neq 1$. Indeed, if it weren't the case, then t_1 would be an endvertex of neither e nor e' for otherwise $T_{x_1} \in D'$ which implies that $D' \cap \{v_{\{x_1\}}, x'_1\} \neq \emptyset$ as $|D' \cap V(G_{x_1})| = 1$ and x'_1 should be dominated. But then, D' contains at most one edge as $5 = |D' \cap V(G_c)| \geq |\{t_1\}| + |D' \cap \{b_1, x_1\}| + |D' \cap \{v_{\{x_1\}}, x'_1\}| + |D' \cap \{a_j, b_j, x_j, j \neq 1\}| \geq 1 + 1 + 1 + 2$ and neither e nor e' has an endvertex in $\{a_j, b_j, x_j\}$ for some $j \neq 1$ by assumption, a contradiction. Since e and e' have at most one common endvertex, it then follows that $|D' \cap V(G_c)| \geq |\{t_1\}| + |D' \cap \{a_j, b_j, x_j, j \neq 1\}| + 3 \geq 1 + 2 + 3$, a contradiction. Thus, either e or e' has an endvertex in $\{a_j, b_j, x_j\}$ for some $j \neq 1$, say $j = 2$ without loss of generality. Suppose that x_2 is an endvertex of e. Then the other endvertex of e should be b_2 for otherwise it belongs to K_c and thus, a_2 would not be dominated. But then, we conclude by Observation 2 and the fact that $|D' \cap V(G_c)| = 5$, that D' contains only one edge. Thus, $e = a_2 b_2$ or $e = a_2 t_2$ and since $v_{\{x_1\}}$ must be dominated, necessarily $x_3 \in D'$; but then, a_3 is not dominated. Therefore, it must be that b_1 belongs to D'; and we conclude similarly that if a_2 (resp. a_3) is not in D' then b_2 (resp. b_3) belongs to D'.

Now, since $t_1, a_1 \notin D'$, it follows that $T_{x_1} \in D'$ for otherwise t_1 would not be dominated. But $|D' \cap V(G_x)| = 1$ and so, $F_{x_1} \notin D'$; thus, $D' \cap \{x'_1, v_{\{x_1\}}\} \neq \emptyset$ as x'_1 must be dominated and we may assume, without loss of generality, that in fact, $v_{\{x_1\}} \in D'$. Then, if $D' \cap \{v_{\{x_2\}}, v_{\{x_3\}}\} = \emptyset$, necessarily $F_{x_2}, F_{x_3} \in D'$; indeed, since $|D' \cap V(G_c)| = 5$, at least one among x'_2 and x'_3 does not belong to D', say x'_2 without loss of generality. But if $x'_3 \in D'$, then exactly one of a_j and b_j, for $j \neq 1$ belongs to D' (recall that if $a_j \notin D'$ then $b_j \in D'$) and therefore, D' contains at most one edge. Thus, $F_{x_2}, F_{x_3} \in D'$ which implies that $D' \cap \{t_j, a_j\} \neq \emptyset$ for $j \neq 1$ as t_j must be dominated. But by Observation 2 and the fact that $|D' \cap V(G_c)| = 5$, we have that $|D' \cap \{t_2, t_3\}| \leq 1$ and so, D' contains at most one edge. Thus, $D' \cap \{v_{\{x_2\}}, v_{\{x_3\}}\} \neq \emptyset$ and since by Observation 2 $|D' \cap V(K_c)| \leq 2$, we conclude that in fact $|D' \cap V(K_c)| = 2$. But then, exactly one among a_j and b_j belongs to D' for $j \neq 1$ and so, D' contains only one edge. Consequently, no such dominating set D' exists and thus, $ct_\gamma(G_\Phi) = 3$.

Conversely, assume that $ct_\gamma(G_\Phi) = 3$ and consider a minimum dominating set D of G_Φ. It readily follows from Observations 1 and 4 that for any variable $x \in X$, $|D \cap V(G_x)| = 1$. Now consider a clause gadget G_c. Then, by Observation 4, we obtain that $t_i \notin D$ (resp. $x'_i \notin D$) for $i \in \{1,2,3\}$, as otherwise it would be within distance at most 2 from the vertex in D belonging to the gadget G_{x_i}.

Now since for any $i \in \{1,2,3\}$, $t_i \notin D$, if $a_i \notin D$ then $b_i \in D$ as a_i must be dominated (also note that by Observation 4, if $a_i \in D$ then $b_i \notin D$). Thus, by Observations 6 and 7, we conclude that for any clause gadget G_c, $|D \cap \{a_1, a_2, a_3\}| = 2$ and $|D \cap \{b_1, b_2, b_3\}| = 1$, say $a_1, a_2, b_3 \in D$ without loss of generality. But then, $v_{\{x_3\}}$ must belong to D; indeed, since $b_3 \in D$, it follows that $T_{x_3} \in D$ for otherwise t_3 is not dominated. Observation 4 then implies that $x'_3 \notin D$ and thus, it can only be dominated by $v_{\{x_3\}}$. But then, it follows from Observation 5 that every vertex in G_c is dominated and we conclude that $|D \cap V(G_c)| = 4$ by minimality of D. Consequently, $|D| = |X| + 4|C|$ which concludes the proof of Claim 1.

▶ **Claim 2.** $\gamma(G_\Phi) = |X| + 4|C|$ *if and only if* Φ *is satisfiable.*

Assume first that $\gamma(G_\Phi) = |X| + 4|C|$ and consider a minimum dominating set D of G_Φ. We construct a truth assignment from D satisfying Φ as follows. For any $x \in X$, if $T_x \in D$, set x to true; otherwise, set x to false. We claim that each clause $c \in C$ has exactly one true variable. Indeed, it follows from Observation 1 that $|D \cap V(G_c)| = 4$ for any $c \in C$, and from Claim 1 that $ct_\gamma(G_\Phi) = 3$. But then, by Observation 3, for any $i \in \{1,2,3\}$, $a_i \notin D$ if and only if $b_i \in D$ (a_i would otherwise not be dominated). It then follows from Observations 6 and 7 that $|D \cap \{a_1,a_2,a_3\}| = 2$ and $|D \cap \{b_1,b_2,b_3\}| = 1$ for any $c \in C$; but by Observation 5 we conclude that $b_i \in D$ if and only if $T_{x_i} \in D$, which proves our claim.

Conversely, assume that Φ is satisfiable and consider a truth assignment satisfying Φ. We construct a dominating set D of G_Φ as follows. If variable x is set to true, we add T_x to D; otherwise, we add F_x to D. For any clause $c \in C$ and $i \in \{1,2,3\}$, if $T_{x_i} \in D$, then add b_i to D; otherwise, add a_i to D. Since every clause has exactly one true variable, it follows that $|D \cap \{b_1,b_2,b_3\}| = 1$ and $|D \cap \{a_1,a_2,a_3\}| = 2$; finally add $v_{\{x_i\}}$ to D where $b_i \in D$. Now clearly $|D \cap V(G_c)| = 4$ and every vertex in G_c is dominated. Thus, $|D| = |X| + 4|C|$ and so by Observation 1, $\gamma(G_\Phi) = |X| + 4|C|$, which concludes this proof.

Now combining Claims 1 and 2, we have that Φ is satisfiable if and only if $ct_\gamma(G_\Phi) = 3$ which completes the proof of Theorem 2. ◀

By observing that for any graph G, G is a YES-instance for CONTRACTION NUMBER(γ,3) if and only if G is a NO-instance for 2-EDGE CONTRACTION(γ), we deduce the following corollary from Theorem 2.

▶ **Corollary 3.** 2-EDGE CONTRACTION(γ) *is* coNP*-hard.*

It is thus coNP-hard to decide whether $ct_\gamma(G) \leq 2$ for a graph G; and in fact, it is NP-hard to decide whether equality holds, as stated in the following.

▶ **Theorem 4 (♠).** CONTRACTION NUMBER(γ,2) *is* NP*-hard.*

We finally consider the case $k = 1$.

▶ **Theorem 5.** 1-EDGE CONTRACTION(γ) *is* NP*-hard even when restricted to* P_t*-free graphs, with* $t \geq 9$.

Proof. We reduce from DOMINATING SET: given an instance (G,ℓ) of this problem, we construct an equivalent instance G' of 1-EDGE CONTRACTION(γ) as follows. We denote by $\{v_1,\dots,v_n\}$ the vertex set of G. The graph G' consists of $\ell+1$ copies of G, denoted by $G_0,\dots,G_\ell$, connected in such a way that for any $1 \leq i \leq \ell$ and $1 \leq k \leq n$, the copies $v_k^i \in V(G_i)$ and $v_k^0 \in V(G_0)$ of a vertex v_k of G are true twins in the subgraph of G' induced by $V(G_0) \cup V(G_i)$; and for any $1 \leq i,j \leq \ell$ and $1 \leq k \leq n$, the copies $v_k^i \in V(G_i)$ and $v_k^j \in V(G_j)$ of a vertex v_k of G are false twins in the subgraph of G' induced by $\bigcup_{1 \leq p \leq \ell} V(G_p)$. Next, we add $\ell+1$ pairwise nonadjacent vertices $x_1,\dots,x_{\ell+1}$, which are made adjacent to every vertex in G_0; x_i is further made adjacent to every vertex in G_i, for all $1 \leq i \leq \ell$. Finally, we add a vertex y adjacent to only $x_{\ell+1}$ (see Fig. 2). Note that the fact that for all $1 \leq k \leq n$ and $1 \leq i,j \leq \ell$, v_k^i and v_k^j (resp. v_k^i and v_k^0) are false (resp. true) twins within the graph induced by $\bigcup_{1 \leq p \leq \ell} V(G_p)$ (resp. $V(G_0) \cup V(G_i)$) is not made explicit on Fig. 2 for the sake of readability. In the following, we denote by $X = \{x_1,\dots,x_{\ell+1}\}$ and $V = \bigcup_{0 \leq p \leq \ell} V(G_p)$. We now claim the following.

▶ **Claim 3.** $\gamma(G') = \min\{\gamma(G)+1, \ell+1\}$.

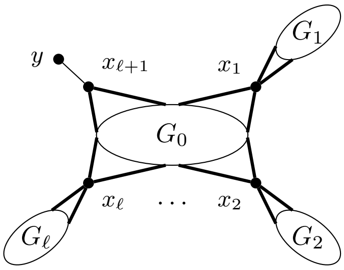


Figure 2 The graph G' (thick lines indicate that the vertex x_i is adjacent to every vertex in G_0 and G_i, for $i = 1, \dots, \ell + 1$).

It is clear that $\{x_1, \dots, x_{\ell+1}\}$ is a dominating set of G'; thus, $\gamma(G') \leq \ell + 1$. If $\gamma(G) \leq \ell$ and $\{v_{i_1}, \dots, v_{i_k}\}$ is a minimum dominating set of G, it is easily seen that $\{v^0_{i_1}, \dots, v^0_{i_k}, x_{\ell+1}\}$ is a dominating set of G'. Thus, $\gamma(G') \leq \gamma(G) + 1$ and so, $\gamma(G') \leq \min\{\gamma(G) + 1, \ell + 1\}$. Now, suppose to the contrary that $\gamma(G') < \min\{\gamma(G) + 1, \ell + 1\}$ and consider a minimum dominating D' set of G'. We first make the following simple observation.

▶ **Observation 8.** *For any dominating set D of G', $D \cap \{y, x_{\ell+1}\} \neq \emptyset$.*

Now, since $\gamma(G') < \ell + 1$, there exists $1 \leq i \leq \ell$ such that $x_i \notin D'$ (otherwise, $\{x_1, \dots, x_\ell\} \subset D'$ and combined with Observation 8, D' would be of size at least $\ell + 1$). But then, $D'' = D' \cap V$ must dominate every vertex in G_i, and so $|D''| \geq \gamma(G)$. Since $|D''| \leq |D'| - 1$ (recall that $D' \cap \{y, x_{\ell+1}\} \neq \emptyset$), we then have $\gamma(G) \leq |D'| - 1$, a contradiction. Thus, $\gamma(G') = \min\{\gamma(G) + 1, \ell + 1\}$.

We now show that (G, ℓ) is a Yes-instance for Dominating Set if and only if G' is a Yes-instance for 1-Edge Contraction(γ).

First assume that $\gamma(G) \leq \ell$. Then, $\gamma(G') = \gamma(G) + 1$ by the previous claim, and if $\{v_{i_1}, \dots, v_{i_k}\}$ is a minimum dominating set of G, then $\{v^0_{i_1}, \dots, v^0_{i_k}, x_{\ell+1}\}$ is a minimum dominating set of G' which is not stable. Hence, by Theorem 1(i), G' is a Yes-instance for 1-Edge Contraction(γ).

Conversely, assume that G' is a Yes-instance for 1-Edge Contraction(γ) i.e., there exists a minimum dominating set D' of G' which is not stable (see Theorem 1(i)). Then, Observation 8 implies that there exists $1 \leq i \leq \ell$ such that $x_i \notin D'$; indeed, if it weren't the case, then by Claim 3 we would have $\gamma(G') = \ell + 1$ and thus, D' would consist of $x_1, \dots, x_\ell$ and either y or $x_{\ell+1}$. In both cases, D' would be stable, a contradiction. It follows that $D'' = D' \cap V$ must dominate every vertex in G_i and thus, $|D''| \geq \gamma(G)$. But $|D''| \leq |D'| - 1$ (recall that $D' \cap \{y, x_{\ell+1}\} \neq \emptyset$) and so by Claim 3, $\gamma(G) \leq |D'| - 1 \leq (\ell + 1) - 1$ that is, (G, ℓ) is a Yes-instance for Dominating Set.

▷ Claim 6 (♠). If G is a $2K_2$-free graph, then G' is a P_9-free graph.

Since Dominating Set is NP-complete on $2K_2$-free graphs [5], the above claim concludes the proof of Theorem 5. ◀

Given the NP-hardness of 1-Edge Contraction(γ) and its close relation to Dominating Set, it is natural to consider the complexity of the problem when parameterized by the size of a minimum dominating set of the input graph. In the following, we show that 1-Edge Contraction(γ) is W[1]-hard when parameterized by $\gamma + mimw$, where $mimw$ denotes the maximum induced matching-width parameter (in short, mim-width). For a formal definition and basic properties of this width measure we refer the reader to [18]. We first state two simple facts regarding the mim-width of a graph.

▶ **Observation 9.** *Let G be a graph and $u, v \in V(G)$ be two vertices that are true (resp. false) twins in G. Then $mimw(G - v) = mimw(G)$.*

▶ **Observation 10.** *Let G be a graph and $v \in V(G)$. Then $mimw(G) \leq mimw(G - v) + 1$.*

▶ **Theorem 7.** 1-Edge Contraction(γ) *is* W[1]*-hard parameterized by $\gamma + mimw$.*

Proof. We give a parameterized reduction from Dominating Set parameterized by solution size plus mim-width, which is a problem that was recently shown to be W[1]-hard by Fomin et al. [10]. Given an instance (G, ℓ) of Dominating Set, the construction of the equivalent instance G' for 1-Edge Contraction(γ) is the same as the one introduced in the proof of Theorem 5; and it is there shown that G is a Yes-instance for Dominating Set if and only if G' is a Yes-instance for 1-Edge Contraction(γ). Now, note that G' can be obtained from G by the addition of true twins (the set $V(G_1)$), the addition of false twins (the sets $V(G_2), \ldots, V(G_\ell)$), and the addition of $\ell + 2$ vertices $(x_1, \ldots, x_{\ell+1}, y)$. By Observation 9, the addition of true (resp. false) twins does not increase the mim-width of a graph and, by Observation 10, the addition of a vertex can only increase the mim-width of G by one; thus, $mimw(G') \leq mimw(G) + \ell + 2$ and since $\gamma(G') \leq \ell + 1$ by Claim 3, we conclude that $mimw(G') + \gamma(G') \leq mimw(G) + 2\ell + 3$. ◀

In order to obtain complexity results for further graph classes, let us now consider subdivisions of edges.

▶ **Lemma 8 (♠).** *Let G be a graph and let G' be the graph obtained by 3-subdividing every edge of G. Then G is a* Yes*-instance for* 1-Edge Contraction(γ) *if and only if G' is a* Yes*-instance for* 1-Edge Contraction(γ)*.*

By 3-subdividing every edge of a graph G sufficiently many times, we deduce the following two corollaries from Lemma 8.

▶ **Corollary 9.** 1-Edge Contraction(γ) *is* NP*-hard when restricted to bipartite graphs.*

▶ **Corollary 10.** *For any $\ell \geq 3$,* 1-Edge Contraction(γ) *is* NP*-hard when restricted to $\{C_3, \ldots, C_\ell\}$-free graphs.*

We finally observe that, even if an edge is given, deciding whether contracting this particular edge decreases the domination number is unlikely to be solvable in polynomial time as shown in the following result.

▶ **Theorem 11.** *There exists no polynomial-time algorithm deciding whether contracting a given edge decreases the domination number, unless* P = NP*.*

Proof. We denote by Edge Contraction(γ) the problem that takes as an input a graph $G = (V, E)$ and an edge $e \in E$, and asks whether $\gamma(G \backslash e) \leq \gamma(G) - 1$. We show that if Edge Contraction(γ) can be solved in polynomial time, then Dominating Set can also be solved in polynomial time. Since Dominating Set is a well-known NP-complete problem, the result follows.

Let (G, ℓ) be an instance for Dominating Set and let e be an edge of G. We run the polynomial time algorithm for Edge Contraction(γ) to determine if $\gamma(G \setminus e) = \gamma(G) - 1$; we then have two possible scenarios.

Case 1. (G, e) is a YES-instance for EDGE CONTRACTION(γ). Since $\gamma(G \setminus e) = \gamma(G) - 1$, we know that G has a dominating set of size ℓ if and only if $G \setminus e$ has a dominating set of size $\ell - 1$. Hence, we obtain that $(G \setminus e, \ell - 1)$ is an equivalent instance for DOMINATING SET.

Case 2. (G, e) is a NO-instance for EDGE CONTRACTION(γ). Since $\gamma(G \setminus e) = \gamma(G)$, we know that G has a dominating set of size ℓ if and only if $G \setminus e$ has a dominating set of size ℓ. In this case, we obtain that $(G \setminus e, \ell)$ is an equivalent instance for DOMINATING SET.

In both cases, the ensuing equivalent instance has one less vertex. Thus, by applying the polynomial-time algorithm for EDGE CONTRACTION(γ) at most n times, we obtain a trivial instance for DOMINATING SET and can therefore correctly determine its answer. ◀

4 Algorithms

We now deal with cases in which k-EDGE CONTRACTION(γ) is tractable, for $k = 1, 2$. A first simple approach to the problem, from which we obtain Proposition 12, is based on brute force.

▶ **Proposition 12.** *For $k = 1, 2$, k-*EDGE CONTRACTION(γ) *can be solved in polynomial time for a graph class $\mathcal{C}$, if either*

(a) *$\mathcal{C}$ is closed under edge contractions and* DOMINATING SET *can be solved in polynomial time on $\mathcal{C}$; or*

(b) *for every $G \in \mathcal{C}$, $\gamma(G) \leq q$, where q is some fixed constant; or*

(c) *$\mathcal{C}$ is the class of $(H + K_1)$-free graphs, where $|V_H| = q$ is a fixed constant and k-*EDGE CONTRACTION(γ) *is polynomial-time solvable on H-free graphs.*

Proof. In order to prove item (a), it suffices to note that if we can compute $\gamma(G)$ and $\gamma(G \setminus e)$, for any edge e of G, in polynomial time, then we can determine whether a graph G is a YES-instance for 1-EDGE CONTRACTION(γ) in polynomial time (we may proceed in a similar fashion for 2-EDGE CONTRACTION(γ)).

For item (b), we proceed as follows. Given a graph G of $\mathcal{C}$, we first check whether G has a dominating vertex. If it is the case, then G is a NO-instance for k-EDGE CONTRACTION(γ) for both $k = 1, 2$. Otherwise, we may consider any subset $S \subset V(G)$ with $|S| \leq q$ and check whether it is a dominating set of G. Since there are at most $\mathcal{O}(n^q)$ possible such subsets, we can determine the domination number of G and check whether the conditions given in Theorem 1 (i) or (ii) are satisfied in polynomial time.

Finally, so as to prove item (c), we provide the following algorithm that works similarly for $k = 1$ and $k = 2$. Let H and q be as stated and let G be an instance of k-EDGE CONTRACTION(γ) on $(H + K_1)$-free graphs. We first test whether G is H-free (note that this can be done in time $\mathcal{O}(n^q)$). If this is the case, we use the polynomial-time algorithm for k-EDGE CONTRACTION(γ) on H-free graphs. Otherwise, G has an induced subgraph isomorphic to H; but since G is a $(H+K_1)$-free graph, $V(H)$ must then be a dominating set of G and so, $\gamma(G) \leq q$. We then conclude by Proposition 12(b) that k-EDGE CONTRACTION(γ) is also polynomial-time solvable in this case. ◀

Proposition 12(b) provides an algorithm for 1-EDGE CONTRACTION(γ) parameterized by the size of a minimum dominating set of the input graph running in XP-time. Note that this result is optimal as 1-EDGE CONTRACTION(γ) is W[1]-hard with such parameterization from Theorem 7.

We further show that even though simple, this brute force method provides polynomial-time algorithms for a number of relevant classes of graphs, such as graphs of bounded tree-width and graphs of bounded mim-width. We first state the following result and observation.

▶ **Theorem 13.** [18] *Given a graph G and a decomposition of width t,* Dominating Set *can be solved in time $\mathcal{O}^*(3^t)$ when parameterized by tree-width, and in time $\mathcal{O}^*(n^{3t})$ when parameterized by mim-width.*

▶ **Observation 11.** $mimw(G \setminus e) \leq mimw(G) + 1$.

Indeed, note that the graph $G \setminus e$ can be obtained from G by the removal of the vertices u and v where $e = uv$, and the addition of a new vertex whose neighborhood is $N_G(u) \cup N_G(v)$. The result then follows from Observation 10 and the fact that vertex deletion does not increase the mim-width of a graph.

▶ **Proposition 14.** *Given a decomposition of width t, k-*Edge Contraction*(γ) can be solved in time $\mathcal{O}^*(3^t)$ in graphs of tree-width at most t and in time $\mathcal{O}^*(n^{3t})$ in graphs of mim-width at most t, for $k = 1, 2$.*

Proof. We use the above-mentioned brute force approach and Theorem 13. That is, for $k = 1$, the algorithm first computes $\gamma(G)$ and then computes $\gamma(G \setminus e)$ for every $e \in E(G)$. For $k = 2$, the algorithm proceeds similarly for every pair of edges. We next show that the width parameters increase by a constant when contracting at most two edges. It is a well-known fact that $tw(G \setminus e) \leq tw(G)$ and so, $tw(G \setminus \{e, f\}) \leq tw(G)$. By Observation 11, $mimw(G \setminus e) \leq mimw(G) + 1$ which implies that $mimw(G \setminus \{e, f\}) \leq mimw(G) + 2$. Also note that, given a tree (resp. mim) decomposition of width t for G, we can construct in polynomial time decompositions of width t (resp. at most $t + 2$) for $G \setminus e$ and $G \setminus \{e, f\}$. This implies that $\gamma(G \setminus e)$ and $\gamma(G \setminus \{e, f\})$ can also be computed in time $\mathcal{O}^*(3^t)$ if G is a graph of tree-width at most t, and in time $\mathcal{O}^*(n^{3t})$ if G is a graph of mim-width at most t. ◀

Proposition 14 provides an algorithm for 1-Edge Contraction(γ) parameterized by mim-width running in XP-time; this result is optimal as 1-Edge Contraction(γ) is W[1]-hard parameterized by mim-width from Theorem 7.

Since Dominating Set is polynomial-time solvable in P_4-free graphs (see [12]), it follows from Proposition 12(a) that k-Edge Contraction(γ) can also be solved efficiently in this graph class. However, Dominating Set is NP-complete for P_5-free graphs (see [5]) and thus, it is natural to examine the complexity of k-Edge Contraction(γ) for this graph class. As we next show, k-Edge Contraction(γ) is in fact polynomial-time solvable on P_5-free graphs, for $k = 1, 2$.

▶ **Lemma 15.** *If G is a P_5-free graph with $\gamma(G) \geq 3$, then $ct_\gamma(G) = 1$.*

Proof. Let $G = (V, E)$ be a P_5-free graph and D be a minimum dominating set of G. Suppose that D is a stable set and consider $u, v \in D$ such that $d(u, v) = \max_{x,y \in D} d(x, y)$. Since G is P_5-free, $d(u, v) \leq 3$ and, since D is stable, $d(u, v) \geq 2$. We distinguish two cases depending on this distance.

Case 1. $d(u, v) = 3$. Let x (resp. y) be the neighbor of u (resp. v) on a shortest path from u to v. Then, $N(u) \cup N(v) \subseteq N(x) \cup N(y)$; indeed, if a is a neighbor of u, then a is nonadjacent to v (recall that $d(u, v) = 3$) and thus, a is adjacent to either x or y for otherwise a, u, x, y and v would induce a P_5 in G. The same holds for any neighbor of v. Consequently, $(D\backslash\{u, v\}) \cup \{x, y\}$ is a minimum dominating set of G which is not stable; the result then follows from Theorem 1(i).

Case 2. $d(u, v) = 2$. Since D is stable and $d(u, v) = \max_{x,y \in D} d(x, y) = 2$, it follows that every $w \in D\backslash\{u, v\}$ is at distance two from both u and v. Let x (resp. y) be the vertex on a shortest path from u (resp. v) to some vertex $w \in D\backslash\{u, v\}$.

Suppose first that $x = y$. If every private neighbor of w with respect to D is adjacent to x then $(D\backslash\{w\}) \cup \{x\}$ is a minimum dominating set of G which is not stable; the result then follows from Theorem 1(i). We conclude similarly if every private neighbor of u or v with respect to D is adjacent to x. Thus, we may assume that w (resp. u; v) has a private neighbor t (resp. r; s) with respect to D which is nonadjacent to x. Since G is P_5-free, it then follows that r, s and t are pairwise adjacent. But then, t, r, u, x and v induce a P_5, a contradiction.

Finally, suppose that $x \neq y$ (we may also assume that $uy, vx \notin E$ as we otherwise fall back in the previous case). Then, $xy \in E$ for u, x, w, y and v would otherwise induce a P_5. Now, if a is a private neighbor of u with respect to D then a is adjacent to either x or y (a, u, x, y and v otherwise induce a P_5); we conclude similarly that any private neighbor of v with respect to D is adjacent to either x or y. If b is adjacent to both u and v but not w, then it is adjacent to x (and y) as v, b, u, x and w (u, b, v, y and w) would otherwise induce a P_5. But then, $(D\backslash\{u, v\}) \cup \{x, y\}$ is a minimum dominating set of G which is not stable; thus, by Theorem 1(i), $ct_\gamma(G) = 1$ which concludes the proof. ◀

▶ **Theorem 16.** k-Edge Contraction(γ) *is polynomial-time solvable on* P_5*-free graphs, for* $k = 1, 2$.

Proof. If G has a dominating vertex, then G is clearly a No-instance for both $k = 1, 2$. Now, for every $uv \in E(G)$, we check whether $\{u, v\}$ is a dominating set. If it is the case, then by Theorem 1(i), G is a Yes-instance for k-Edge Contraction(γ) for $k = 1, 2$. If no edge of G is dominating, we consider all the pairs of nonadjacent vertices of G. If there exists such a pair dominating G and $k = 1$ then by Theorem 1(i), we have a No-instance for 1-Edge Contraction(γ) since this implies that every minimum dominating set of G is stable. For the case $k = 2$, if G has two nonadjacent vertices dominating G, we then consider all triples of vertices of G to check whether there exists one which is dominating and contains at least two edges (see Theorem 1(ii)). Finally, both for $k = 1$ and $k = 2$, if G has no dominating set of size at most two, then by Lemma 15, G is a Yes-instance for k-Edge Contraction(γ). ◀

5 Conclusion

In this paper, we studied the k-Edge Contraction(γ) problem and provided the first complexity results. In particular, we showed that 1-Edge Contraction(γ) is NP-hard for P_t-free graphs, $t \geq 9$, but polynomial-time solvable for P_5-free graphs; it would be interesting to determine the complexity status for P_ℓ-free graphs, for $\ell \in \{6, 7, 8\}$. Similarly, the complexity of 2-Edge Contraction(γ) for P_t-free graphs, with $t \geq 6$, remains an interesting open problem.

References

1 Cristina Bazgan, Sonia Toubaline, and Zsolt Tuza. The most vital nodes with respect to independent set and vertex cover. *Discrete Applied Mathematics*, 159:1933–1946, October 2011. doi:10.1016/j.dam.2011.06.023.

2 Cristina Bazgan, Sonia Toubaline, and Daniel Vanderpooten. Critical edges for the assignment problem: Complexity and exact resolution. *Operations Research Letters*, 41:685–689, November 2013. doi:10.1016/j.orl.2013.10.001.

3 Cédric Bentz, Costa Marie-Christine, Dominique de Werra, Christophe Picouleau, and Bernard Ries. Blockers and Transversals in some subclasses of bipartite graphs : when caterpillars are dancing on a grid. *Discrete Mathematics*, 310:132–146, January 2010. doi:10.1016/j.disc.2009.08.009.

4 Cédric Bentz, Costa Marie-Christine, Dominique de Werra, Christophe Picouleau, and Bernard Ries. *Weighted Transversals and Blockers for Some Optimization Problems in Graphs*, pages 203–222. Progress in Combinatorial Optimization. ISTE-WILEY, 2012.

5 Alan A. Bertossi. Dominating sets for split and bipartite graphs. *Information Processing Letters*, 19(1):37–40, 1984. doi:10.1016/0020-0190(84)90126-1.

6 Marie-Christine Costa, Dominique de Werra, and Christophe Picouleau. Minimum d-blockers and d-transversals in graphs. *Journal of Combinatorial Optimization*, 22(4):857–872, 2011. doi:10.1007/s10878-010-9334-6.

7 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

8 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg; New York, fourth edition, 2010.

9 Öznur Yaşar Diner, Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Contraction Blockers for Graphs with Forbidden Induced Paths. In *Algorithms and Complexity*, pages 194–207, Cham, 2015. Springer International Publishing.

10 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the Tractability of Optimization Problems on H-Graphs. In *ESA 2018*, volume 112 of *LIPIcs*, pages 30:1–30:14, 2018. doi:10.4230/LIPIcs.ESA.2018.30.

11 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

12 Teresa W. Haynes, S. T. Hedetniemi, and Peter J. Slater. *Fundamentals of domination in graphs*. New York : Marcel Dekker, 1998.

13 Jia Huang and Jun-Ming Xu. Domination and Total Domination Contraction Numbers of Graphs. *Ars Combinatoria*, 94, January 2010.

14 Foad Mahdavi Pajouh, Vladimir Boginski, and Eduardo Pasiliao. Minimum Vertex Blocker Clique Problem. *Networks*, 64, August 2014. doi:10.1002/net.21556.

15 Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Reducing the Clique and Chromatic Number via Edge Contractions and Vertex Deletions. In *ISCO 2016*, volume 9849 of *LNCS*, pages 38–49, 2016. doi:10.1007/978-3-319-45587-7_4.

16 Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Blocking Independent Sets for H-Free Graphs via Edge Contractions and Vertex Deletions. In *TAMC 2017*, volume 10185 of *LNCS*, pages 470–483, 2017. doi:10.1007/978-3-319-55911-7_34.

17 Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Critical vertices and edges in H-free graphs. *Discrete Applied Mathematics*, 257:361–367, 2019. doi:10.1016/j.dam.2018.08.016.

18 Martin Vatshelle. *New width parameters of graphs*. PhD thesis, University of Bergen, Norway, 2012.

19 Öznur Yaşar Diner, Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Contraction and deletion blockers for perfect graphs and H-free graphs. *Theoretical Computer Science*, 746:49–72, 2018. doi:10.1016/j.tcs.2018.06.023.

Measuring what Matters: A Hybrid Approach to Dynamic Programming with Treewidth

Eduard Eiben
Department of Informatics, University of Bergen, Bergen, Norway
eduard.eiben@uib.no

Robert Ganian
Vienna University of Technology, Vienna, Austria
rganian@gmail.com

Thekla Hamm
Vienna University of Technology, Vienna, Austria
thekla.hamm@tuwien.ac.at

O-joung Kwon
Department of Mathematics, Incheon National University, Korea
ojoungkwon@inu.ac.kr

Abstract

We develop a framework for applying treewidth-based dynamic programming on graphs with "hybrid structure", i.e., with parts that may not have small treewidth but instead possess other structural properties. Informally, this is achieved by defining a refinement of treewidth which only considers parts of the graph that do not belong to a pre-specified tractable graph class. Our approach allows us to not only generalize existing fixed-parameter algorithms exploiting treewidth, but also fixed-parameter algorithms which use the size of a modulator as their parameter. As the flagship application of our framework, we obtain a parameter that combines treewidth and rank-width to obtain fixed-parameter algorithms for Chromatic Number, Hamiltonian Cycle, and Max-Cut.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, treewidth, rank-width, fixed-parameter algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.42

Funding E. Eiben was supported by Pareto-Optimal Parameterized Algorithms (ERC Starting Grant 715744). R. Ganian acknowledges support by the Austrian Grant Agency (FWF, project P31336 NFPC), and is also affiliated with FI MUNI, Czech Republic. O. Kwon was supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education (No. NRF-2018R1D1A1B07050294).

1 Introduction

Over the past decades, the use of structural properties of graphs to obtain efficient algorithms for NP-hard computational problems has become a prominent research direction in computer science. Perhaps the best known example of a structural property that can be exploited in this way is the tree-likeness of the inputs, formalized in terms of the decomposition-based structural parameter *treewidth* [35]. It is now well known that a vast range of fundamental problems admit so-called *fixed-parameter* algorithms parameterized by the treewidth of the input graph – that is, can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$ on n-vertex graphs of treewidth k (for some computable function f). We say that such problems are FPT parameterized by treewidth.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 42; pp. 42:1–42:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

On the other hand, dense graphs are known to have high treewidth and hence require the use of different structural parameters; the classical example of such a parameter tailored to dense graphs is *clique-width* [8]. Clique-width is asymptotically equivalent to the structural parameter *rank-width* [34], which is nowadays often used instead of clique-width due to a number of advantages (rank-width is much easier to compute [26] and can be used to design more efficient fixed-parameter algorithms than clique-width [21, 22]). While rank-width (or, equivalently, clique-width) dominates[1] treewidth and can be used to "lift" fixed-parameter algorithms designed for treewidth to well-structured dense graphs for a number of problems, there are also important problems which are FPT parameterized by treewidth but W[1]-hard (and hence probably not FPT) parameterized by rank-width. The most prominent examples of such problems are Chromatic Number [18], Hamiltonian Cycle [18], and Max-Cut [19].

Another generic type of structure used in algorithmic design is based on measuring the size of a *modulator* (i.e., a vertex deletion set) [5] to a certain graph class. Basic examples of parameters based on modulators include the vertex cover number (a modulator to edgeless graphs) [16] and the feedback vertex set number (a modulator to forests)[3]. For dense graphs, modulators to graphs of rank-width 1 have been studied [13, 30], and it is known that for every constant c one can find a modulator of size at most k to graphs of rank-width c (if such a modulator exists) in time $f(k) \cdot n$ [29]. However, the algorithmic applications of such modulators have remained largely unexplored up to this point.

Our Contribution. We develop a class of *hybrid parameters* which combines the foremost advantages of treewidth and modulators to obtain a "best-of-both-worlds" outcome. In particular, instead of measuring the treewidth of the graph itself or the size of a modulator to a graph class $\mathcal{H}$, we consider the treewidth of a (torso of a) modulator to $\mathcal{H}$. This parameter, which we simply call $\mathcal{H}$-treewidth, allows us to lift previously established tractability results for a vast number of problems from treewidth and modulators to a strictly more general setting. As our first technical contribution, we substantiate this claim with a meta-theorem that formalizes generic conditions under which a treewidth-based algorithm can be generalized to $\mathcal{H}$-treewidth; the main technical tool for the proof is an adaptation of *protrusion replacement* techniques [2, Section 4].

As the flagship application of $\mathcal{H}$-treewidth, we study the case where $\mathcal{H}$ is the class $\mathcal{R}_c$ of graphs of rank-width at most c (an arbitrary constant). $\mathcal{R}_c$-treewidth hence represents a way of lifting treewidth towards dense graphs that lies "between" treewidth and rank-width. We note that this class of parameters naturally incorporates a certain scaling trade-off: $\mathcal{R}_c$-treewidth dominates $\mathcal{R}_{c-1}$-treewidth for each constant c, but the runtime bounds for algorithms using $\mathcal{R}_c$-treewidth are worse than those for $\mathcal{R}_{c-1}$-treewidth.

Our first result for $\mathcal{R}_c$-treewidth is a fixed-parameter algorithm for computing the parameter itself. We then develop fixed-parameter algorithms for Chromatic Number, Hamiltonian Cycle and Max-Cut parameterized by $\mathcal{R}_c$-treewidth; moreover, in 2 out of these 3 cases the parameter dependencies of our algorithms are essentially tight. These algorithms represent generalizations of:

1. classical fixed-parameter algorithms parameterized by treewidth [12],
2. polynomial-time algorithms on graphs of bounded rank-width [22], and
3. (not previously known) fixed-parameter algorithms parameterized by modulators to graph classes of bounded rank-width.

[1] Parameter α dominates parameter β if for each graph class with bounded β, α is also bounded.

The main challenge for all of these problems lies in dealing with the fact that some parts of the graph need to be handled using rank-width based techniques, while for others we use treewidth-based dynamic programming. We separate these parts from each other using the notion of *nice $\mathcal{H}$-tree decompositions.* The algorithm then relies on enhancing the known dynamic programming approach for solving the problem on treewidth with a subroutine that not only solves the problem on the part of the graph outside of the modulator, but also serves as an interface by supplying appropriate records to the treewidth-based dynamic programming part of the algorithm. At its core, each of these subroutines boils down to solving an "extended" version of the original problem parameterized by the size of a modulator to $\mathcal{R}_c$; in particular, each subroutine immediately implies a fixed-parameter algorithm for the respective problem when parameterized by a modulator to constant rank-width. To give a specific example for such a subroutine, in the case of Chromatic Number one needs to solve the problem parameterized by a modulator to $\mathcal{R}_c$ where the modulator is furthermore precolored.

To avoid any doubt, we make it explicitly clear that the runtime of all of our algorithms utilizing $\mathcal{R}_c$-treewidth has a polynomial dependency on the input where the degree of this polynomial depends on c (as is necessitated by the W[1]-hardness of the studied problems parameterized by rank-width).

Related Work. Previous works have used a combination of treewidth with *backdoors*, a notion that is closely related to modulators, in order to solve non-graph problems such as Constraint Satisfaction [25], Boolean Satisfiability [24] and Integer Programming [23]. Interestingly, the main technical challenge in all of these papers is the problem of computing the parameter, while using the parameter to solve the problem is straightforward. In the algorithmic results presented in this contribution, the situation is completely reversed: the main technical challenge lies in developing the algorithms (and most notably the subroutines) for solving our targeted problems. Moreover, while the aforementioned three papers focus on solving a single problem, here we aim at identifying and exploiting structural properties that can be used to solve a wide variety of graph problems.

Other parameters which target inputs with hybrid structure include *sm-width* [36] and well-structured modulators [14]. It is not difficult to show that these are different (both conceptually and factually) from $\mathcal{R}_c$-treewidth.

2 Preliminaries

For $i \in \mathbb{N}$, let $[i]$ denote the set $\{1, \dots, i\}$. All graphs in this paper are simple and undirected. We refer to the standard textbook [11] for basic graph terminology. For $S \subseteq V(G)$, let $G[S]$ denote the subgraph of G induced by S. For $v \in V(G)$, the set of neighbors of v in G is denoted by $N_G(v)$ (or $N(v)$ when G is clear from the context). For $A \subseteq V(G)$, let $N_G(A)$ denote the set of vertices in $G - A$ that have a neighbor in A. For a vertex set A, an *A-path* is a path whose endpoints are contained in A and all the internal vertices are contained in $G - A$.

A set M of vertices in a graph G is called a *modulator* to a graph class $\mathcal{H}$ if $G - M \in \mathcal{H}$. The operation of *collapsing* a vertex set X, denoted $G \circ X$, deletes X from the graph and adds an edge between vertices $u, v \in V(G - X)$ if $uv \notin E(G)$ and there is an u-v path with all internal vertices in $G[X]$. We assume that the reader is familiar with *parameterized complexity* [9, 12], notably with notions such as FPT, W[1], treewidth and Courcelle's Theorem.

Rank-width. For a graph G and $U, W \subseteq V(G)$, let $\boldsymbol{A}_G[U, W]$ denote the $U \times W$-submatrix of the adjacency matrix over the two-element field GF(2), i.e., the entry $a_{u,w}$, $u \in U$ and $w \in W$, of $\boldsymbol{A}_G[U, W]$ is 1 if and only if $\{u, w\}$ is an edge of G. The *cut-rank* function ρ_G of a graph G is defined as follows: For a bipartition (U, W) of the vertex set $V(G)$, $\rho_G(U) = \rho_G(W)$ equals the rank of $\boldsymbol{A}_G[U, W]$.

A *rank-decomposition* of a graph G is a pair (T, μ) where T is a tree of maximum degree 3 and $\mu : V(G) \to \{t \mid t \text{ is a leaf of } T\}$ is a bijective function. For an edge e of T, the connected components of $T - e$ induce a bipartition (X, Y) of the set of leaves of T. The *width* of an edge e of a rank-decomposition (T, μ) is $\rho_G(\mu^{-1}(X))$. The *width* of (T, μ) is the maximum width over all edges of T. The *rank-width* of G, $\mathbf{rw}(G)$ in short, is the minimum width over all rank-decompositions of G. We denote by $\mathcal{R}_i$ the class of all graphs of rank-width at most i. A *rooted* rank-decomposition is obtained from a rank-decomposition by subdividing an arbitrarily chosen edge, and the newly created vertex is called the *root*.

Unlike clique-width, rank-width can be computed exactly by a fixed-parameter algorithm (which also outputs a corresponding rank-decomposition) [26].

Monadic Second-Order Logic. Counting Monadic Second-Order Logic (CMSO$_1$) is a basic tool to express properties of vertex sets in graphs. The syntax of CMSO$_1$ includes logical connectives $\wedge, \vee, \neg, \Leftrightarrow, \Rightarrow$, variables for vertices and vertex sets, quantifiers $\exists, \forall$ over these variables, and the relations $a \in A$ where a is a vertex variable and A is a vertex set variable; $\mathrm{adj}(a, b)$, where a and b are vertex variables and the interpretation is that a and b are adjacent; equality of variables representing vertices and sets of vertices; $\mathrm{Parity}(A)$, where A is a vertex set variable and the interpretation is that $|A|$ is even.

The CMSO$_1$ Optimization problem is defined as follows:

CMSO$_1$-OPT

Instance: A graph G, a CMSO$_1$ formula $\phi(A)$ with a free set variable A, and opt $\in \{\min, \max\}$.

Task: Find an interpretation of the set A in G such that G models $\phi(A)$ and A is of minimum/maximum (depending on opt) cardinality.

From the fixed-parameter tractability of computing rank-width [26], the equivalence of rank-width and clique-width [34] and Courcelle's Theorem for graphs of bounded clique-width [7] it follows that:

▶ **Fact 1** ([21]). CMSO$_1$-OPT *is* FPT *parameterized by* $\mathbf{rw}(G) + |\phi|$*, where* G *is the input graph and* ϕ *is the* CMSO$_1$ *formula.*

3 $\mathcal{H}$-Treewidth

The aim of $\mathcal{H}$-treewidth is to capture the treewidth of a modulator to the graph class $\mathcal{H}$. However, one cannot expect to obtain a parameter with reasonable algorithmic applications by simply measuring the treewidth of the graph *induced* by a modulator to $\mathcal{H}$ – instead, one needs to measure the treewidth of a so-called *torso*, which adds edges to track how the vertices in the modulator interact through $\mathcal{H}$. To substantiate this, we observe that Hamiltonian Cycle would become NP-hard even on graphs with a modulator that (1) induces an edgeless graph, and (2) is a modulator to an edgeless graph, and where (3) each connected component outside the modulator has boundedly many neighbors in the modulator [1].

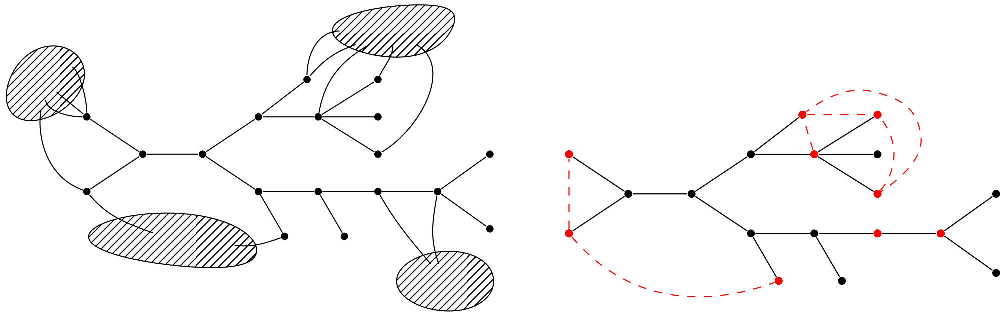


Figure 1 *Left:* A graph G with a tree as a modulator to $\mathcal{H}$ (the part in $\mathcal{H}$ is depicted hatched). *Right:* The corresponding $\mathcal{H}$-torso.

The notion of a *torso* has previously been algorithmically exploited in other settings [23, 25, 33], and its adaptation is our first step towards the definition of $\mathcal{H}$-treewidth (see also Figure 1).

▶ **Definition 1** ($\mathcal{H}$-Torso). *Let G be a graph and $X \subseteq V(G)$. For a graph class $\mathcal{H}$, $G \circ X$ is an* $\mathcal{H}$-torso *of G if each connected component C of $G[X]$ satisfies $C \in \mathcal{H}$.*

▶ **Definition 2** ($\mathcal{H}$-Treewidth). *The $\mathcal{H}$-treewidth of a graph G is the minimum treewidth of an $\mathcal{H}$-torso of G. We denote the $\mathcal{H}$-treewidth of G by $\mathbf{tw}_{\mathcal{H}}(G)$.*

Typically, we will want to consider a graph class $\mathcal{H}$ for which certain problems are polynomial-time tractable. Hence, we will assume w.l.o.g. that $(\emptyset, \emptyset) \in \mathcal{H}$. From the definition we easily observe that $\mathbf{tw}_{\mathcal{H}}(G) \leq \mathbf{tw}(G)$ for every G and $\mathcal{H}$.

3.1 Nice $\mathcal{H}$-Tree-Decompositions

Just like for tree decompositions, we can also define a canonical form of decompositions which has properties that are convenient when formulating dynamic programs using $\mathcal{H}$-treewidth. Intuitively, a *nice $\mathcal{H}$-tree decomposition* behaves like a nice tree decomposition on the torso graph (see points 1-3), with the exception that the neighborhoods of the collapsed parts must occur as special *boundary* leaves (see points 4-5).

▶ **Definition 3** (Nice $\mathcal{H}$-Tree-Decomposition). *A nice $\mathcal{H}$-tree decomposition of a graph G is a triple $(X, T, \{B_t \mid t \in V(T)\})$ where $X \subseteq V(G)$ such that $G \circ X$ is an $\mathcal{H}$-torso, $(T, \{B_t \mid t \in V(T)\})$ is a rooted tree decomposition of $G \circ X$, and:*

1. *Every node in T has at most two children.*
2. *If a node t has children $t_1 \neq t_2$, then $B_t = B_{t_1} = B_{t_2}$ and we call t a* join node*.*
3. *If a node t has exactly one child t', then either (a) there exists $x \in V(G) \setminus B_{t'}$ such that $B_t = B_{t'} \cup \{x\}$ and we call t an* introduce node*, or (b) there exists $x \in V(G) \setminus B_t$ such that $B_{t'} = B_t \cup \{x\}$ and we call t a* forget node*.*
4. *If a node t is a leaf, then (a) $|B_t| = 1$ and we call t a* simple leaf*, or (b) $B_t = N(C)$ for some connected component C of $G[X]$ and we call t a* boundary leaf*.*
5. *For each connected component C of $G[X]$ there is a unique leaf t with $B_t = N(C)$.*

An illustration of a nice $\mathcal{H}$-tree decomposition showcasing how it differs from a nice tree decomposition is provided in Figure 2; in line with standard terminology for treewidth, we call the sets B_t *bags*. The width of a nice $\mathcal{H}$-tree decomposition is simply the width of $(T, \{B_t \mid t \in V(T)\})$. Given a node t in a nice $\mathcal{H}$-tree decomposition T, we let Y_t be the set of all vertices

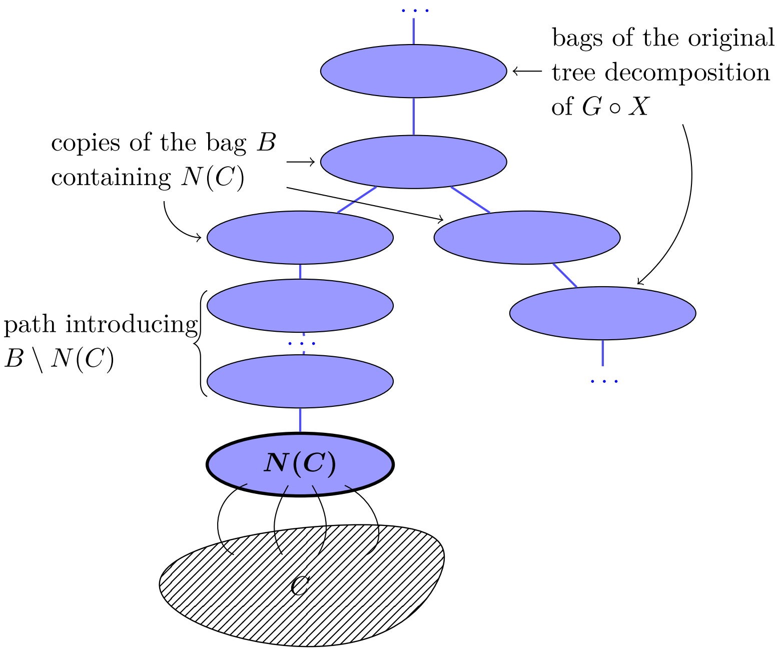


Figure 2 Part of a nice $\mathcal{H}$-tree-decomposition (blue) including a boundary leaf (bold) and a connected component C (hatched) of X.

contained in the bags of the subtree rooted at t, i.e., $Y_t = B_t \cup \bigcup_{p \text{ is separated from the root by } t} B_p$. It is possible to show that computing a nice $\mathcal{H}$-tree decomposition of bounded width can be reduced to finding an appropriate $\mathcal{H}$-torso (this is because a nice $\mathcal{H}$-tree decomposition can be obtained straightforwardly from a nice tree decomposition of the torso).

Hence, we can state the problem of computing a decomposition as follows:

$\mathcal{H}$-TREEWIDTH *Parameter: k*
Instance: A graph G, an integer k.
Task: Find an $\mathcal{H}$-torso U of G such that $\mathbf{tw}(U) \le k$, or correctly determine that no such $\mathcal{H}$-torso exists.

An Algorithmic Meta-Theorem. Before proceeding to the flagship application of $\mathcal{H}$-treewidth where $\mathcal{H}$ is the class of graphs of bounded rank-width, here we give a generic set of conditions that allow fixed-parameter algorithms for problems parameterized by $\mathcal{H}$-treewidth. Specifically, we consider graph problems that are *finite-state* [6] or have *finite integer index* [2, 4, 20]. Informally speaking, such problems only transfer a limited amount of information across a small separator in the input graph and hence can be solved "independently" on both sides of such a separator. Since these notions are only used in this section, we provide concise definitions below.

First of all, we will need the notion of boundaried graphs and gluing. A graph $\bar{G}$ is called *t-boundaried* if it contains t distinguished vertices identified as $b_1^G, \ldots, b_t^G$. The *gluing operation* $\oplus$ takes two t-boundaried graphs $\bar{G}$ and $\bar{H}$, creates their disjoint union, and then alters this disjoint union by identifying the boundaries of the two graphs (i.e. by setting $b_i^G = b_i^H$ for each $i \in [t]$).

Consider a decision problem $\mathcal{P}$ whose input is a graph. We say that two t-boundaried graphs $\bar{C}$ and $\bar{D}$ are *equivalent*, denoted by $\bar{C} \sim_{\mathcal{P},t} \bar{D}$, if for each t-boundaried graph $\bar{H}$ it holds that $\bar{C} \oplus \bar{H} \in \mathcal{P}$ if and only if $\bar{D} \oplus \bar{H} \in \mathcal{P}$. We say that $\mathcal{P}$ is *finite-state* (or *FS*, in brief) if, for each $t \in \mathbb{N}$, $\sim_{\mathcal{P},t}$ has a finite number of equivalence classes.

Next, consider a decision problem $\mathcal{Q}$ whose input is a graph and an integer. In this case we say that two t-boundaried graphs $\bar{C}$ and $\bar{D}$ are equivalent (denoted by $\bar{C} \sim_{\mathcal{Q},t} \bar{D}$) if there exists an *offset* $\delta(\bar{C}, \bar{D}) \in \mathbb{Z}$ such that for each t-boundaried graph $\bar{H}$ and each $q \in \mathbb{Z}$: $(\bar{C} \oplus \bar{H}, q) \in \mathcal{Q}$ if and only if $(\bar{D} \oplus \bar{H}, q + \delta(\bar{C}, \bar{D})) \in \mathcal{Q}$. We say that $\mathcal{Q}$ has *finite integer index* (or is *FII*, in brief) if, for each $t \in \mathbb{N}$, $\sim_{\mathcal{Q},t}$ has a bounded number of equivalence classes.

We note that a great number of natural graph problems are known to be FS or FII. For instance, all problems definable in Monadic Second Order logic are FS [2, Lemma 3.2], while examples of FII problems include Vertex Cover, Independent Set, Feedback Vertex Set, Dominating Set, to name a few [20]. We say that a FS or FII problem $\mathcal{P}$ is *efficiently extendable* on a graph class $\mathcal{H}$ if there is a fixed-parameter algorithm (parameterized by t) that takes as input a t-boundaried graph $\bar{G}$ such that the boundary is a modulator to $\mathcal{H}$ and outputs the equivalence class of $\bar{G}$ w.r.t. $\sim_{\mathcal{P},t}$.

▶ **Theorem 4.** *Let $\mathcal{P}$ be a FS or FII graph problem and $\mathcal{H}$ be a graph class such that (1) $\mathcal{P}$ is efficiently extendable on $\mathcal{H}$, (2) $\mathcal{P}$ is* FPT *parameterized by treewidth, and (3) $\mathcal{H}$-*treewidth *is* FPT*. Then $\mathcal{P}$ is* FPT *parameterized by $\mathcal{H}$-treewidth.*

Proof Sketch. We can solve $\mathcal{P}$ as follows. First of all, we use Point (3) to compute an $\mathcal{H}$-torso $G \circ X$ of treewidth k, where k is the $\mathcal{H}$-treewidth. Next, for each connected component C of $G[X]$, we use the fact that $C \in \mathcal{H}$ and Point (1) to compute the equivalence class of the boundaried graph $\bar{H} = \overline{G[C \cup N(C)]}$ where the boundary is $N(C)$. Note that since $\mathbf{tw}(G \circ X) \leq k$ and $N(C)$ forms a clique in $G \circ X$, $|N(C)| \leq k$ and hence this step takes only fixed-parameter time. Next, we use a brute-force enumeration argument to compute a bounded-size representative of the equivalence class of $\bar{H}$, and replace $\bar{H}$ with this representative. After doing this exhaustively, we obtain a graph G' of bounded treewidth, for which we can invoke Point (2). ◀

4 $\mathcal{R}_c$-Treewidth

This section focuses on the properties of $\mathcal{R}_c$-treewidth, a hierarchy of graph parameters that represent our flagship application of the generic notion of $\mathcal{H}$-treewidth.

Comparison to Known Parameters. It follows from the definition of $\mathcal{H}$-treewidth that $\mathcal{R}_c$-treewidth dominates treewidth (for every $c \in \mathbb{N}$). Similarly, it is obvious that $\mathcal{R}_c$-treewidth dominates the size of a modulator to $\mathcal{R}_c$ (also for every $c \in \mathbb{N}$). The following lemma shows that, for every fixed c, $\mathcal{R}_c$-treewidth is dominated by rankwidth.

▶ **Lemma 5.** *Let $c \in \mathbb{N}$. If $\mathbf{tw}_{\mathcal{R}_c}(G) = k$ then $\mathbf{rw}(G) \leq c + k + 1$.*

Proof. Let the $\mathcal{R}_c$-treewidth of G be witnessed by some nice $\mathcal{R}_c$-tree-decomposition $(X, T, \{B_t \mid t \in V(T)\})$ of width k.

We can obtain a rank-decomposition (T', μ) of G from $(X, T, \{B_t \mid t \in V(T))\}$ as follows:

For vertices v of $G \circ X$ such that there is no leaf node $t \in V(T)$ with $B_t = \{v\}$, let $t \in V(T)$ be a forget node with child t' such that $B_t \cup \{v\} = B_{t'}$. Turn t' into a join node by introducing t_1, t_2 with $B_{t_1} = B_{t_2} = B_{t'}$ as children of t', attaching the former child of t' to t_1 and a new leaf node t_v with $B_{t_v} = \{v\}$ below t_2. Note that this preserves the fact that for any $v \in V(G) \setminus X$, $T[\{u \in V(T) \mid v \in B_u\}]$ is a tree. Now we can choose for each $v \in V(G \circ X)$ some $\mu(v) \in V(T)$ such that $B_t = \{v\}$. This defines an injection from $V(G \circ X)$ to the leaves of T. However not every leaf of T is mapped to by μ. On one hand there are the boundary leaf nodes, below which we will attach subtrees to obtain a rank-decomposition of G. On the other hand there may be $v \in V(G \circ X)$ for which the choice of $\mu(v)$ was not unique, i.e. there is $t \neq \mu(v)$ with $B_t = \{v\}$. For all such v and t we delete all nodes on the root-t-path in T that do not lie on a path from the root to a vertex in $\{\mu(w) \mid w \in V(G \circ X)\} \cup \{t' \mid t' \text{ boundary leaf node}\}$. This turns μ into an injection from $V(G \circ X)$ to the leaves of T, that is surjective on the non-boundary leaf nodes.

Next, we extend (T, μ) to a rank-decomposition of G by proceeding in the following way for each connected component C of $G[X]$:
Let $t_C \in V(T)$ be the boundary leaf node with $B_t = N(C)$. Since $\mathbf{rw}(C) \leq c$, we find a rank-decomposition (T_C, μ_C) of C with width at most c. Attach T_C below t_C.
Let T' be the tree obtained by performing these modifications for all connected components C. Consider the rank-decomposition of G given by

$$\left(T', v \mapsto \begin{cases} \mu(v) & \text{if } v \in V(G \circ X) \\ \mu_C(v) & \text{if } v \in C \text{ for some } C \text{ as above} \end{cases}\right).$$

We show that its width is at most $c + k + 1$. Any edge e of T' is of one of the following types:

- e corresponds to an edge already contained in T: Then e induces a bipartition (X_G, Y_G) of $V(G)$. Fix $t \in V(T)$ to be the vertex in which e starts.
  Let $x \in X_G$ and $y \in Y_G$ be such that $xy \in E(G)$. Observe that e does not separate neighbors in X as these lie within the same connected component of $G[X]$ whose rank-decomposition is, by construction, attached completely within one of the two subtrees of T' separated by e. So, we consider $x \in V(G \circ X)$. If $y \in V(G \circ X)$ then x and y occur together in some bag of the original tree of the tree decomposition, and as remarked earlier this is still the case modified tree. This implies that at least one of $\{x, y\}$ must be present in B_t. If $y \notin V(G \circ X)$, by the construction of T', y corresponds to a leaf $t_y \in V(T')$ in a subtree attached to T rooted at $t_C \in V(T)$ with $B_{t_C} = N(C) \ni x$ and t_C and t_y are not separated by the removal of e. Also, x corresponds to a leaf $t_x \in V(T)$ which is in the subtree of $T' - e$ not containing t_y, i.e. the subtree not containing t_C. This means any subtree containing t_C and t_x also contains t, and since $(T, \{B_t \mid t \in V(T)\})$ is a tree-decomposition and $x \in B_{t_x} \cap B_{t_C}$ this means $x \in B_t$.
  In both cases at least one of $\{x, y\}$ is in B_t. Since this argument applies for every edge crossing the bipartition (X_G, Y_G), it follows that $\boldsymbol{A}_G[X_G, Y_G]$ may only contain "1" entries in rows and columns that correspond to the vertices in B_t. Since $|B_t| \leq k + 1$, it holds that $\boldsymbol{A}_G[X_G, Y_G]$ can be converted into a zero matrix by deleting at most $k + 1$ rows plus columns, which is a sufficient condition for $\boldsymbol{A}_G[X_G, Y_G]$ having rank at most $k + 1$.
- e corresponds to an edge in a rank-decomposition of some connected component C of $G[X]$: Then e induces a bipartition (X_G, Y_G) of $V(G)$ and a bipartition (X_C, Y_C), where $X_C = X_G \cap C$ and $Y_C = Y_G \cap C$, of C. Since vertices of C are only connected to vertices in $N(C)$ outside of C and $N(C) \subseteq B_t$ for some $t \in V(T)$, we have $\rho_G(X_G) \leq \rho_C(X_C) + |N(C)| \leq c + k + 1$.
- e corresponds to an edge connecting the rank-decomposition of some connected component C of $G[X]$ to T: Then the bipartition induced is $(C, V(G) \setminus C)$ and as $N(C) \subseteq B_t$ for some $t \in V(T)$ $\rho_G(C) \leq |N(C)| \leq k + 1$. ◀

Next, we compare $\mathcal{R}_c$-treewidth to Telle and Saether's *sm-width* [36].

▶ **Lemma 6.** *$\mathcal{R}_c$-treewidth and sm-width are incomparable.*

Computing $\mathcal{R}_c$-Treewidth. Our aim here is to determine the complexity of computing our parameters, i.e., finding a torso of small treewidth. Obtaining such a torso is a base prerequisite for our algorithms. We formalize the problem below.

$\mathcal{R}_c$-Treewidth *Parameter:* k
Instance: A graph G, an integer k.
Task: Find a $\mathcal{R}_c$-torso U of G such that $\mathbf{tw}(U) \leq k$, or correctly determine that no such $\mathcal{R}_c$-torso exists.

▶ **Lemma 7.** *$\mathcal{R}_c$-Treewidth is* FPT*.*

5 Algorithms Exploiting Modulators to $\mathcal{R}_c$

For a problem $\mathcal{P}$ to be FPT parameterized by $\mathcal{R}_c$-treewidth, $\mathcal{P}$ must necessarily be FPT parameterized by treewidth and also FPT parameterized by the size of a modulator to $\mathcal{R}_c$. However, it is important to note that the latter condition is not sufficient; indeed, one can easily invent artificial problems that are defined in a way which make them trivial in both of the cases outlined above, but become intractable (or even undecidable) once parameterized by $\mathcal{R}_c$-treewidth. That is, after all, why we need the notion of *efficient extendability* in Theorem 4.

Hence, in order to develop fixed-parameter algorithms for Chromatic Number, Hamiltonian Cycle and Max-Cut parameterized by $\mathcal{R}_c$-treewidth, we first need to show that they are not only FPT parameterized by the size of a modulator to $\mathcal{R}_c$, but they are also efficiently extendable. Such a result would be sufficient to employ Theorem 4 together with Lemma 7 in order to establish the desired fixed-parameter tractability results. That is also our general aim in this section, with one caveat: in order to give explicit and tight upper bounds on the parameter dependency of our algorithms, we provide algorithms that solve generalizations of Chromatic Number, Hamiltonian Cycle and Max-Cut parameterized by the size of a modulator to $\mathcal{R}_c$, whereas it will become apparent in the next section that these generalizations precisely correspond to the records required by the treewidth-based dynamic program that will be used in the torso. In other words, the efficient extendability of our problems on $\mathcal{R}_c$ is not proved directly but rather follows as an immediate consequence of our proofs in this section and the correctness of known treewidth-based algorithms.

Chromatic Number. In Chromatic Number, we are given a graph G and asked for the smallest number $\chi(G)$ such that the vertex set of G can be properly colored using $\chi(G)$ colors, i.e., the smallest number $\chi(G)$ such that $V(G)$ can be partitioned into $\chi(G)$ independent sets. Our aim in this section is to solve a variant of Chromatic Number on graphs with a k-vertex modulator X to $\mathcal{R}_c$ where X is precolored:

$\mathcal{R}_c$-Precoloring Extension *Parameter:* k
Instance: A graph G, a k-vertex modulator $X \subseteq V(G)$ to $\mathcal{R}_c$ and a coloring of X.
Task: Compute the smallest number of colors required to extend the coloring of X to a proper coloring of G.

▶ **Theorem 8.** *$\mathcal{R}_c$-Precoloring Extension can be solved in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$.*

Proof Sketch. Let G be a graph together with a k-vertex modulator X to $\mathcal{R}_c$ and a proper coloring of X by colors $[k]$; let col_X be the set of colors assigned to at least one vertex in X. For disjoint vertex sets S and Q of G, two vertices v and w in S are *twins* with respect to Q if $N(v) \cap Q = N(w) \cap Q$. A *twin class* of S with respect to Q is a maximal subset of S that consists of pairwise twins w.r.t. Q.

Our starting point is a rooted rank-decomposition (T, μ) of $G - X$ of width at most c, which may be computed in time $\mathcal{O}(n^3)$ [26]. On a high level, our algorithm will apply dynamic programming along (T, μ) where it will group colors together based on which twin classes they occur in (analogously as in the XP algorithm for Chromatic Number parameterized by clique-width, due to Kobler and Rotics [31]), but keep different (more detailed) records about the at most k colors used in X.

For each $t \in V(T)$, let S_t be the set of all vertices that are assigned to the descendants of t, and let $G_t := G[S_t \cup X]$. By our definition of (T, μ), recall that $\rho_{G-X}(S_t) \leq c$ and that there are at most $z = 2^c$ twin classes of S_t w.r.t. $V(G) \setminus (X \cup S_t)$. We will refer to these twin classes as $R^t_1, R^t_2, \ldots, R^t_z$.

We are now ready to formally define the dynamic programming table M_t that stores the information we require at a node t of T. For $b_1, b_2, \ldots, b_k \subseteq [z]$ and $\{\, d_Z \in [n] \mid Z \subseteq [z] \,\}$, we let $M_t(b_1, b_2, \ldots, b_k, \{\, d_Z \mid Z \subseteq [z] \,\}) = 1$ if there is a proper coloring of G_t such that (1) for every $i \in [k]$, the color i appears in twin classes in $\{R^t_j : j \in b_i\}$ and does not appear in other twin classes, and (2) for every $Z \subseteq [z]$, d_Z is the number of colors from $\{k+1, k+2, \ldots, n\}$ that appear in twin classes of $\{R^t_j : j \in Z\}$ and do not appear in other twin classes. On the other hand, if no such proper coloring exists then we let $M_t(b_1, b_2, \ldots, b_k, \{\, d_Z \mid Z \subseteq [z] \,\}) = 0$.

The table M_t will be filled in a leaf-to-root fashion. Observe that by definition of d_Z's, for distinct subsets Z_1, Z_2 of $[z]$, d_{Z_1} and d_{Z_2} count disjoint sets of colors. This provides an easy way to count the total number of colors used. Since all vertices in $G - X$ appear below the root node r, the minimum number of colors required for a proper coloring of G will be the minimum value of $\left|\mathrm{col}_X \cup \{\, i \in [k] \mid b_i \neq \emptyset \,\}\right| + \sum_{Z \subseteq [z]} d_Z$, over all tuples $(b_1, b_2, \ldots, b_k, \{\, d_Z \mid Z \subseteq [z] \,\})$ whose M_r value is 1. ◀

Hamiltonian Cycle. In Hamiltonian Cycle, we are given an n-vertex graph G and asked whether G contains a cycle of length n as a subgraph. Note that if we restrict G to some subset of vertices $Y \subseteq V(G)$, then what remains from a Hamiltonian Cycle in G is a set of paths that start and end in the neighborhood of $V(G) \setminus Y$. Hence, the aim of this section is to solve the following generalization of Hamiltonian Cycle:

$\mathcal{R}_c$-Disjoint Paths Cover	*Parameter: k*
Instance:	A graph G, a k-vertex modulator $X \subseteq V(G)$ to $\mathcal{R}_c$, and $m \leq k$ pairs $(s_1, t_1), \ldots, (s_m, t_m)$ of vertices from X with $s_i \neq t_i$ for all $i \in [m]$.
Task:	Decide whether there are internally vertex-disjoint paths $P_1, P_2, \ldots, P_m$ in G such that P_i is a path from s_i to t_i and every vertex in $G - X$ belongs to precisely one path in $P_1, P_2, \ldots, P_m$.

▶ **Theorem 9.** $\mathcal{R}_c$-Disjoint Paths Cover *can be solved in time* $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.

Proof Sketch. Let G be a graph, X be a k-vertex modulator to $\mathcal{R}_c$, and $(s_1, t_1), \ldots, (s_m, t_m)$ be m pairs of vertices from X. Our starting point is once again a rooted rank-decomposition (T, μ) of $G - X$ of width at most c, which may be computed in time $\mathcal{O}(n^3)$ [26]. We will obtain a fixed-parameter algorithm for checking the existence of such paths $P_1, \ldots, P_m$ in G by expanding the records used in Espelage, Gurski and Wanke's algorithm [15] for computing Hamiltonian Cycle parameterized by clique-width.

To follow partial solutions on each subgraph G_t, we consider certain generalizations of path-partitions of subgraphs of G. For a subgraph H of G, an *X-lenient path-partition* $\mathcal{P}$ of H is a collection of paths in H that are internally vertex-disjoint and share only endpoints in X such that $\bigcup_{P \in \mathcal{P}} V(P) = V(H)$. For convenience, we consider a path as an ordered sequence of vertices, and for a path $P = v_1 v_2 \cdots v_x$, we define $\ell(P) = v_1$ and $r(P) = v_x$.

We proceed by introducing our dynamic programming table. For each node t of T, we use the following tuples (D, SP) as indices of the table. Let $D = \{d_{b_1,b_2} \in \{0, 1, \ldots, n\} \mid (b_1, b_2) \in [z] \times [z]\}$. The integer d_{b_1,b_2} will represent the number of paths in an X-lenient path-partition of G_t that are fully contained in $G_t - X$ and whose endpoints are contained in $R^t_{b_1}$ and $R^t_{b_2}$. Let SP be a set such that

- for each $i \in \{1, \ldots, m\}$, $(i, 0, x)$, $(0, i, x)$, (i, i) with some $x \in [z]$ are the only possible tuples in SP,
- each integer in $\{1, \ldots, m\}$ appears at most once as an ℓ among all tuples $(\ell, 0, p)$ or (ℓ, ℓ) in SP, and similarly, each integer in $\{1, \ldots, m\}$ appears at most once as an r among all tuples $(0, r, p)$ or (r, r) in SP.

In short, the tuple $(i, 0, t)$ indicates the existence of a path starting in s_i and ending at a vertex in R^t_x. Similarly, $(0, i, t)$ indicates the existence of a path starting in t_i and ending in R^t_x. The tuple (i, i) indicates the existence of a path starting in s_i and ending in t_i. Note that there are at most $(n+1)^{z^2}$ possibilities for D. For an element of SP, there are $2mz + 1$ possible elements in SP, and thus there are at most 2^{2kz+1} possibilities for SP. This implies that the number of possible tuples (D, SP) is bounded by $(n+1)^{z^2} 2^{2kz+1}$.

We define a DP table M_t such that $M_t(D, SP) = 1$ if there is an X-lenient path-partition $\mathcal{P} = \mathcal{P}_1 \uplus \mathcal{P}_2$ of G_t such that

- $\mathcal{P}_1$ is the subset of $\mathcal{P}$ that consists of all paths fully contained in $G_t - X$,
- for every $d_{b_1,b_2} \in D$, there are exactly d_{b_1,b_2} distinct paths in $\mathcal{P}_1$ with endpoints in $R^t_{b_1}$ and $R^t_{b_2}$,
- for every (ℓ, r, p) or $(\ell, r) \in SP$, there is a unique path $P \in \mathcal{P}_2$ such that
  - if $\ell = i > 0$, then $\ell(P) = s_i$, and if $r = i > 0$, then $r(P) = t_i$,
  - if $\ell = 0$, then $\ell(P) \in R^t_p$, and if $r = 0$, then $r(P) \in R^t_p$,

In this case, we say that the X-lenient path-partition $\mathcal{P}$ is a partial solution with respect to (D, SP), and also (D, SP) is a characteristic of $\mathcal{P}$. We define $\mathcal{Q}_t$ as the set of all tuples (D, SP) where $M_t(D, SP) = 1$.

The table M_t is filled in a leaf-to-root fashion. Since all vertices in $G - X$ appear below the root node ro, to decide whether there is a desired X-lenient path-partition, it suffices to confirm that there are D and SP such that $M_{ro}(D, SP) = 1$, for every $d_{b_1,b_2} \in D$, $d_{b_1,b_2} = 0$, and for every $i \in \{1, 2, \ldots, m\}$, $(i, i) \in SP$.

The proof can be completed by describing a dynamic program to fill in the table M_t for each node $t \in V(T)$ in a leaf-to-root fashion. ◀

Max-Cut. The third problem we consider is Max-Cut, where we are given an integer ℓ together with an n-vertex graph G and asked whether $V(G)$ can be partitioned into sets V_1 and V_2 such that the number of edges with precisely one endpoint in V_1 (called the *cut size*) is at least ℓ.

$\mathcal{R}_c$-Max-Cut Extension *Parameter:* k

Instance: A graph G, a k-vertex modulator $X \subseteq V(G)$ to $\mathcal{R}_c$, $s \subseteq X$, and $\ell \in \mathbb{N}$.

Task: Is there a partition of $V(G)$ into sets V_1 and V_2 such that $X \cap V_1 = s$ and the number of edges between V_1 and V_2 is at least ℓ.

▶ **Theorem 10.** *$\mathcal{R}_c$-Max-Cut Extension can be solved in polynomial time.*

6 Algorithmic Applications of $\mathcal{R}_c$-Treewidth

In this section, we show that Chromatic Number, Hamiltonian Cycle and Max-Cut are FPT parameterized by $\mathcal{R}_c$-treewidth. As our starting point, recall that each of these problems admits a fixed-parameter algorithm when parameterized by treewidth which is based on leaf-to-root dynamic programming along the nodes of a nice tree decomposition. Notably, the algorithms are based on defining a certain *record* $\delta(P, Q)$ (for vertex sets P, Q) such that $\delta(B_t, Y_t)$ captures all the relevant information required to solve the problem on $G[Y_t]$ and to propagate this information from a node t to its parent. The algorithms compute these records on the leaves of the tree decomposition by brute force, and then dynamically update these records while traversing a nice tree decomposition towards the root; once the record $\delta(B_r, Y_r)$ is computed for the root r of the decomposition, the algorithm outputs the correct answer.

Our general strategy for solving these problems will be to replicate the records employed by the respective dynamic programming algorithm $\mathbb{A}$ used for treewidth, but *only for the nice $\mathcal{R}_c$-tree decomposition of the torso* of the input graph G. Recall that aside from the "standard" simple leaf nodes, nice $\mathcal{R}_c$-tree decompositions also contain boundary leaf nodes, which serve as separators between the torso and a connected component C with rank-width at most c. For $\mathbb{A}$ to work correctly with the desired runtime, we need to compute the record for each boundary leaf node using a subprocedure that exploits the bounded rank-width of C; in particular, we will see that this amounts to solving the problems defined in Section 5. Before proceeding to the individual problems, we provide a formalization and proof for the general ideas outlined above.

▶ **Lemma 11.** *Let $\mathcal{P}$ be a graph problem which can be solved via a fixed-parameter algorithm $\mathbb{A}$ parameterized by treewidth, where $\mathbb{A}$ runs in time $f(k') \cdot n'^a$ and operates by computing a certain record δ in a leaves-to-root fashion along a provided nice width-k' tree decomposition of the n'-vertex input graph.*

Let $\mathcal{Q}$ be obtained from $\mathcal{P}$ by receiving the following additional information in the input: (1) a nice $\mathcal{R}_c$-tree decomposition $(X, T, \{B_t \mid t \in V(T)\})$ of width k for the input n-vertex graph G, and (2) for each boundary leaf node t corresponding to the neighborhood of a connected component C of $G[X]$, the record $\delta(B_t, B_t \cup C)$.

Then, $\mathcal{Q}$ can be solved in time $f(k) \cdot n^a$.

Chromatic Number. Chromatic Number is W[1]-hard parameterized by rank-width [17] but can be solved in time $2^{\mathcal{O}(\mathbf{tw}(G) \cdot \log \mathbf{tw}(G))} \cdot n$ on n-vertex graphs when a minimum-width tree-decomposition is provided with the input [28]; moreover, it is known that this runtime is essentially tight [32].

It is well known that the chromatic number is at most $\mathbf{tw}(G) + 1$. One possible way of defining records in order to achieve a runtime of $2^{\mathcal{O}(\mathbf{tw}(G) \cdot \log \mathbf{tw}(G))} \cdot n$ is to track, for each proper coloring of vertices in a bag B_t, the minimum number of colors required to extend such a coloring to Y_t [28]. Formally, let S_t be the set of all colorings of B_t with colors $[\mathbf{tw}(G) + 1]$, and let $\alpha(B_t, Y_t) : S_t \to \mathbb{Z}$ be defined as follows:

- $\alpha(B_t, Y_t)(s) = -1$ if s is not a proper coloring of $G[B_t]$.
- $\alpha(B_t, Y_t)(s) = q$ if q is the minimum number of colors used by any proper coloring of $G[Y_t]$ which extends s.

Using Theorem 8, we can compute such $\alpha(B_t, Y_t)(s)$ for every proper coloring s of B_t. Hence, combining Lemma 11 and Theorem 8, we obtain:

▶ **Theorem 12.** Chromatic Number *can be solved in time* $2^{\mathcal{O}(k\log(k))} \cdot n^{\mathcal{O}(1)}$ *if a nice* $\mathcal{R}_c$*-tree decomposition of width* k *is provided on the input.*

Hamiltonian Cycle. Hamiltonian Cycle is W[1]-hard parameterized by rank-width [17] but can be solved in time $2^{\mathcal{O}(\mathbf{tw}(G)\cdot\log\mathbf{tw}(G))} \cdot n$ on n-vertex graphs when a minimum-width tree-decomposition is provided with the input via standard dynamic programming. This algorithm can be improved to run in time $2^{\mathcal{O}(\mathbf{tw}(G))} \cdot n$) by applying the advanced *rank-based approach* of Cygan, Kratsch and Nederlof [10] to prune the number of records. To simplify our exposition, here we focus on extending the standard dynamic programming algorithm which yields a slightly super-exponential runtime.

One possibility for defining the records for Hamiltonian Cycle is to track all possible ways one can cover Y_t by paths that start and end in B_t (intuitively, this corresponds to what remains of a hypothetical solution if we "cut off" everything above Y_t) [12]. Formally, let ${B_t}^\diamond$ be defined as follows:

- if $|B_t| > 2$, then ${B_t}^\diamond$ is the set of graphs with at most $|B_t|$ edges and degree at most 2 over vertex set B_t;
- if $|B_t| = 2$, then ${B_t}^\diamond$ contains three (multi)graphs over vertex set B_t: the edgeless graph, the graph with one edge, and the multigraph with two edges and no loops;
- if $|B_t| = 1$, then ${B_t}^\diamond$ contains an edgeless graph and a graph with a single loop, both over the single vertex in B_t;
- if $|B_t| = 0$, then ${B_t}^\diamond = \{\text{YES}, \text{NO}\}$.

We let $\beta(B_t, Y_t) : {B_t}^\diamond \to \{0, 1\}$, where for $Q \in {B_t}^\diamond$ we set $\beta(B_t, Y_t)(Q) = 1$ if and only if there exists a set P of paths in $G[Y_t]$ and a bijection that maps each $(v_1, \dots, v_\ell) \in P$ to an edge $(v_1, v_\ell) \in E(Q)$ such that each vertex $v \in G[Y_t \setminus B_t]$ is contained in precisely one path in P. In the special case where $B_t = \emptyset$, our records explicitly state whether $G[Y_t]$ contains a Hamiltonian cycle or not.

As before, we can now shift our attention to the problem of computing our records in boundary leaf nodes. We do so by looping over all of the at most k^{2k}-many graphs $Q \in {B_t}^\diamond$; for each such Q we check whether $G[Y_t] - B_t$ can be covered by internally vertex-disjoint paths connecting the pairs of vertices in B_t that form the endpoints of the edges in Q. Hence, we are left with the $\mathcal{R}_c$-Disjoint Paths Cover problem. From Theorem 9 and Lemma 11, we obtain:

▶ **Theorem 13.** Hamiltonian Cycle *can be solved in time* $2^{\mathcal{O}(k\log(k))} \cdot n^{\mathcal{O}(1)}$ *if a nice* $\mathcal{R}_c$*-tree decomposition of width* k *is provided on the input.*

Max-Cut. Max-Cut is another problem that is W[1]-hard parameterized by rank-width [19] but admits a simple fixed-parameter algorithm parameterized by treewidth – notably, it can be solved in time $2^{\mathcal{O}(\mathbf{tw}(G))} \cdot n$ on n-vertex graphs when a minimum-width tree-decomposition is provided with the input via standard dynamic programming [9, 12].

The simplest way of defining the records for Max-Cut is to keep track of all possible ways the bag B_t can be partitioned into V_1 and V_2, and for each entry in our table we keep track of the maximum number of crossing edges in Y_t compatible with that entry. Formally, let $\gamma(B_t, Y_t) : 2^{B_t} \to \mathbb{N}_0$, where for each $s \in 2^{B_t}$ it holds that $\gamma(B_t, Y_t)(s)$ is the maximum cut size that can be achieved in $G[Y_t]$ by any partition (V_1, V_2) satisfying $V_1 \cap B_t = s$. As before, from Theorem 10 and Lemma 11, we obtain:

▶ **Theorem 14.** Max-Cut *can be solved in time* $2^k \cdot n^{\mathcal{O}(1)}$, *if a nice* $\mathcal{R}_c$*-tree decomposition of width* k *is provided on the input.*

7 Concluding Remarks

While the technical contribution of this paper mainly focused on $\mathcal{R}_c$-treewidth, a parameter that allows us to lift fixed-parameter algorithms parameterized by treewidth to well-structured dense graph classes, it is equally viable to consider $\mathcal{H}$-treewidth for other choices of $\mathcal{H}$. Naturally, one should aim at graph classes where problems of interest become tractable, but it is also important to make sure that a (nice) $\mathcal{H}$-tree decomposition can be computed efficiently (i.e., one needs to obtain analogues to our Lemma 7). Examples of graph classes that may be explored in this context include split graphs, interval graphs, and more generally graphs of bounded mim-width [27].

References

1 Takanori Akiyama, Takao Nishizeki, and Nobuji Saito. $\mathcal{NP}$-completeness of the Hamiltonian cycle problem for bipartite graphs. *J. Inform. Process.*, 3:73–76, January 1980.

2 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, November 2016. doi:10.1145/2973749.

3 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Preprocessing for Treewidth: A Combinatorial Analysis through Kernelization. *SIAM J. Discrete Math.*, 27(4):2108–2142, 2013.

4 Hans L. Bodlaender and Babette van Antwerpen-de Fluiter. Reduction Algorithms for Graphs of Small Treewidth. *Inf. Comput.*, 167(2):86–119, 2001.

5 Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.

6 Bruno Courcelle. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Inf. Comput.*, 85(1):12–75, 1990.

7 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

8 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Texts in Computer Science. Springer, 2013.

10 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity Checking Via Bases of Perfect Matchings. *J. ACM*, 65(3):12:1–12:46, 2018.

11 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer Verlag, New York, 2nd edition, 2000.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

13 Eduard Eiben, Robert Ganian, and O-joung Kwon. A single-exponential fixed-parameter algorithm for distance-hereditary vertex deletion. *J. Comput. Syst. Sci.*, 97:121–146, 2018.

14 Eduard Eiben, Robert Ganian, and Stefan Szeider. Solving Problems on Graphs of High Rank-Width. *Algorithmica*, 80(2):742–771, 2018.

15 Wolfgang Espelage, Frank Gurski, and Egon Wanke. Deciding Clique-Width for Graphs of Bounded Tree-Width. *J. Graph Algorithms Appl.*, 7(2):141–180, 2003.

16 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph Layout Problems Parameterized by Vertex Cover. In *Proc. ISAAC 2008*, volume 5369 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2008.

17 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Clique-width: on the price of generality. In *Proc. SODA 2009*, pages 825–834. SIAM, 2009.

18 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of Clique-Width Parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010.

19 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost Optimal Lower Bounds for Problems Parameterized by Clique-Width. *SIAM J. Comput.*, 43(5):1541–1563, 2014.

20 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.

21 Robert Ganian and Petr Hlinený. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discrete Applied Mathematics*, 158(7):851–867, 2010.

22 Robert Ganian, Petr Hlinený, and Jan Obdrzálek. A unified approach to polynomial algorithms on graphs of bounded (bi-)rank-width. *Eur. J. Comb.*, 34(3):680–701, 2013.

23 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going Beyond Primal Treewidth for (M)ILP. In *Proc. AAAI 2017*, pages 815–821. AAAI Press, 2017.

24 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Backdoor Treewidth for SAT. In *Proc. SAT 2017*, volume 10491 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2017.

25 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Combining Treewidth and Backdoors for CSP. In *Proc. STACS 2017*, volume 66 of *LIPIcs*, pages 36:1–36:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

26 Petr Hliněný and Sang-il Oum. Finding Branch-Decompositions and Rank-Decompositions. *SIAM J. Comput.*, 38(3):1012–1032, June 2008. doi:10.1137/070685920.

27 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A Unified Polynomial-Time Algorithm for Feedback Vertex Set on Graphs of Bounded Mim-Width. In *Proc. STACS 2018*, volume 96 of *LIPIcs*, pages 42:1–42:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

28 Klaus Jansen and Petra Scheffler. Generalized Coloring for Tree-like Graphs. *Discrete Applied Mathematics*, 75(2):135–155, 1997.

29 Mamadou Moustapha Kanté, Eun Jung Kim, O-joung Kwon, and Christophe Paul. An FPT Algorithm and a Polynomial Kernel for Linear Rankwidth-1 Vertex Deletion. *Algorithmica*, 79(1):66–95, 2017. doi:10.1007/s00453-016-0230-z.

30 Eun Jung Kim and O-joung Kwon. A Polynomial Kernel for Distance-Hereditary Vertex Deletion. In *Proc. WADS 2017*, volume 10389 of *Lecture Notes in Computer Science*, pages 509–520. Springer, 2017.

31 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.

32 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly Superexponential Parameterized Problems. *SIAM J. Comput.*, 47(3):675–702, 2018.

33 Dániel Marx and Paul Wollan. Immersions in Highly Edge Connected Graphs. *SIAM J. Discrete Math.*, 28(1):503–520, 2014.

34 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

35 Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

36 Sigve Hortemo Sæther and Jan Arne Telle. Between Treewidth and Clique-Width. In *Proc. WG 2014*, pages 396–407, 2014.

The Power Word Problem

Markus Lohrey
Universität Siegen, Germany

Armin Weiß
Universität Stuttgart, Germany

Abstract

In this work we introduce a new succinct variant of the word problem in a finitely generated group G, which we call the power word problem: the input word may contain powers p^x, where p is a finite word over generators of G and x is a binary encoded integer. The power word problem is a restriction of the compressed word problem, where the input word is represented by a straight-line program (i.e., an algebraic circuit over G). The main result of the paper states that the power word problem for a finitely generated free group F is AC^0-Turing-reducible to the word problem for F. Moreover, the following hardness result is shown: For a wreath product $G \wr \mathbb{Z}$, where G is either free of rank at least two or finite non-solvable, the power word problem is complete for coNP. This contrasts with the situation where G is abelian: then the power word problem is shown to be in TC^0.

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

Keywords and phrases word problem, compressed word problem, free groups

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.43

Related Version A full version [27] of the paper is available on arXiv https://arxiv.org/abs/1904.08343.

Funding *Markus Lohrey*: Funded by DFG project LO 748/12-1.
Armin Weiß: Funded by DFG project DI 435/7-1.

Acknowledgements We thank Laurent Bartholdi for pointing out the result [5, Theorem 6.6] on the bound of the order of elements in the Grigorchuk group, which allowed us to establish Theorem 10.

1 Introduction

Algorithmic problems in group theory have a long tradition, going back to the work of Dehn from 1911 [9]. One of the fundamental group theoretic decision problems introduced by Dehn is the *word problem* for a finitely generated group G (with a fixed finite generating set Σ): does a given word $w \in \Sigma^*$ evaluate to the group identity? Novikov [34] and Boone [8] independently proved in the 1950's the existence of finitely presented groups with undecidable word problem. On the positive side, in many important classes of groups the word problem is decidable, and in many cases also the computational complexity is quite low. Famous examples are finitely generated linear groups, where the word problem belongs to deterministic logarithmic space (L for short) [22] and hyperbolic groups where the word problem can be solved in linear time [17] as well as in LOGCFL [23].

In recent years, also compressed versions of group theoretical decision problems, where input words are represented in a succinct form, have attracted attention. One such succinct representation are so-called straight-line programs, which are context-free grammars that produce exactly one word. The size of such a grammar can be much smaller than the word it produces. For instance, the word a^n can be produced by a straight-line program of size $\mathcal{O}(\log n)$. For the *compressed word problem* for the group G the input consists of a straight-line program that produces a word w over the generators of G and it is asked whether w evaluates to the identity element of G. This problem is a reformulation of the circuit evaluation problem for G. The compressed word problem naturally appears when one tries to solve the word problem in automorphism groups or semidirect products [25, Section 4.2]. For the following classes of groups, the compressed word problem is known to be solvable in polynomial time: finite groups (where the compressed word problem is either P-complete or in NC^2 [6]), finitely generated nilpotent groups [20] (where the complexity is even in NC^2), hyperbolic groups [18] (in particular, free groups), and virtually special groups (i.e, finite extensions of subgroups of right-angled Artin groups) [25]. The latter class covers for instance Coxeter groups, one-relator groups with torsion, fully residually free groups and fundamental groups of hyperbolic 3-manifolds. For finitely generated linear groups there is still a randomized polynomial time algorithm for the compressed word problem [26, 25]. Simple examples of groups where the compressed word problem is intractable are wreath products $G \wr \mathbb{Z}$ with G a non-abelian group: for every such group the compressed word problem is coNP-hard [25] (this includes for instance Thompson's group F); on the other hand, if, in addition, G is finite, then the (ordinary) word problem for $G \wr \mathbb{Z}$ is in NC^1 [37].

In this paper, we study a natural variant of the compressed word problem, called the *power word problem.* An input for the power word problem for the group G is a tuple $(p_1, x_1, p_2, x_2, \dots, p_n, x_n)$ where every p_i is a word over the group generators and every x_i is a binary encoded integer (such a tuple is called a *power word*); the question is whether $p_1^{x_1} p_2^{x_2} \cdots p_n^{x_n}$ evaluates to the group identity of G.

From a power word $(p_1, x_1, p_2, x_2, \dots, p_n, x_n)$ one can easily (e.g. by an AC^0-reduction) compute a straight-line program for the word $p_1^{x_1} p_2^{x_2} \cdots p_n^{x_n}$. In this sense, the power word problem is at most as difficult as the compressed word problem. On the other hand, both power words and straight-line programs achieve exponential compression in the best case; so the additional difficulty of the the compressed word problem does not come from a higher compression rate but rather because straight-line programs can generate more "complex" words.

Our main results for the power word problem are the following; in each case we compare our results with the corresponding results for the compressed word problem:

- The power word problem for every finitely generated nilpotent group is in $\mathsf{DLOGTIME}$-uniform TC^0 and hence has the same complexity as the word problem (or the problem of multiplying binary encoded integers). The proof is a straightforward adaption of a proof from [33]. There, the special case, where all words p_i in the input power word are single generators, was shown to be in $\mathsf{DLOGTIME}$-uniform TC^0. The compressed word problem for every finitely generated nilpotent group belongs to the class $\mathsf{DET} \subseteq \mathsf{NC}^2$ and is hard for the counting class $\mathsf{C_{=}L}$ in case of a torsion-free nilpotent group [20].
- The power word problem for a finitely generated group G is NC^1-many-one-reducible to the power word problem for any finite index subgroup of G. An analogous result holds for the compressed word problem as well [20].
- The power word problem for a finitely generated free group is AC^0-Turing-reducible to the word problem for F_2 (the free group of rank two) and therefore belongs to L. In contrast, it was shown in [24] that the compressed word problem for a finitely generated free group of rank at least two is P-complete.
- The power word problem for a wreath product $G \wr \mathbb{Z}$ with G finitely generated abelian belongs to $\mathsf{DLOGTIME}$-uniform TC^0. For the compressed word problem for $G \wr \mathbb{Z}$ with G finitely generated abelian only the existence of a randomized polynomial time algorithm for the complement is known [21].
- The power word problem for the wreath products $F_2 \wr \mathbb{Z}$ and every wreath product $G \wr \mathbb{Z}$, where G is finite and non-solvable, is coNP-complete. For these groups this sharpens the corresponding coNP-hardness result for the compressed word problem [25].

Table 1 Our results on the power word problem compared to previous results on the (compressed) word problem. Here WP stands for “word problem”.

class of groups	PowerWP	CompressedWP	WP
nilpotent groups	TC^0	DET, $\mathsf{C}_=\mathsf{L}$-hard [20]	TC^0 [35]
Grigorchuk group G	L[a)]	PSPACE	L [13]
non-abelian f.g. free	L[b)]	P-complete [24]	L [22]
$G \wr \mathbb{Z}$ for G f.g. abelian	TC^0	coRP [21]	TC^0 [30]
$G \wr \mathbb{Z}$ for G finite non-solvable	coNP-complete	PSPACE, coNP-hard [25]	NC^1 [37]
$F_2 \wr \mathbb{Z}$	coNP-complete	PSPACE, coNP-hard [25]	L[b)] [37]
finite extension of a f.g. group H	NC^1-many-one-reducible to PowerWP(H) (resp. CompressedWP(H) [20], resp. WP(H) [37])		

a) AC^0-many-one-reducible to the word problem of G.
b) AC^0-Turing-reducible to the word problem of F_2.

- The power word problem for the Grigorchuk group is uAC^0-many-one-reducible to the word problem. The word problem for the Grigorchuk group is in L [13], which implies that the compressed word problem is in PSPACE. However, there is no non-trivial lower-bound known for the compressed word problem for the Grigorchuk group.

Table 1 summarizes the above results. Due to space constraints we present only short proof skteches for our main theorems; proofs of all lemmas can be found in the full version [27].

Related work. Implicitly, (variants of) the power word problem have been studied before. In the commutative setting, Ge [14] has shown that one can verify in polynomial time an identity $\alpha_1^{x_1}\alpha_2^{x_2}\cdots\alpha_n^{x_n} = 1$, where the α_i are elements of an algebraic number field and the x_i are binary encoded integers.

Another problem related to the power word problem is the knapsack problem [12, 28, 31] for a finitely generated group G (with generating set Σ): for a given sequence of words $w, w_1, \ldots, w_n \in \Sigma^*$, the question is whether there exist $x_1, \ldots, x_n \in \mathbb{N}$ such that $w = w_1^{x_1}\cdots w_n^{x_n}$ holds in G. For many groups G one can show that if such $x_1, \ldots, x_n \in \mathbb{N}$ exist, then there exist such numbers of size $2^{\mathrm{poly}(N)}$, where $N = |w| + |w_1| + \cdots + |w_n|$ is the input length. This holds for instance for right-angled Artin groups (also known as graph groups). In this case, one nondeterministically guesses the binary encodings of numbers $x_1, \ldots, x_n$ and then verifies, using an algorithm for the power word problem, whether $w_1^{x_1}\cdots w_n^{x_n}w^{-1} = 1$ holds. In this way, it was shown in [28] that for every right-angled Artin group the knapsack problem belongs to NP (using the fact that the compressed word problem and hence the power word problem for a right-angled Artin group belongs to P).

In [16], Gurevich and Schupp present a polynomial time algorithm for a compressed form of the subgroup membership problem for a free group F, where group elements are represented in the form $a_1^{x_1}a_2^{x_2}\cdots a_n^{x_n}$ with binary encoded integers x_i. The a_i must be standard generators of the free group F. This is the same input representation as in [33] and is more restrictive then our setting, where we allow powers of the form w^x for w an arbitrary word over the group generators (on the other hand, Gurevich and Schupp consider the subgroup membership problem, which is more general than the word problem).

2 Preliminaries

Words. An *alphabet* is a (finite or infinite) set Σ; an element $a \in \Sigma$ is called a *letter*. The free monoid over Σ is denoted by Σ^*, its elements are called *words*. The multiplication of the monoid is concatenation of words. The identity element is the empty word 1. The length of a word w is denoted by $|w|$. If w, p, x, q are words such that $w = pxq$, then we call x a *factor* of w, p a *prefix* of w, and q a *suffix* of w. We write $v \leq_{\mathrm{pref}} w$ (resp. $v <_{\mathrm{pref}} w$) if v is a (strict) prefix of w and $v \leq_{\mathrm{suff}} w$ (resp. $v <_{\mathrm{suff}} w$) if v is a (strict) suffix of w.

String rewriting systems. Let Σ be an alphabet and $S \subseteq \Sigma^* \times \Sigma^*$ be a set of pairs, called a *string rewriting system*. We write $\ell \to r$ if $(\ell, r) \in S$. The corresponding *rewriting relation* $\xRightarrow[S]{}$ over Σ^* is defined by: $u \xRightarrow[S]{} v$ if and only if there exist $\ell \to r \in S$ and words $s, t \in \Sigma^*$ such that $u = s\ell t$ and $v = srt$. We also say that u can be rewritten to v in one step. We write $u \xRightarrow[S]{k} v$ if u can be rewritten to v in exactly k steps, i.e., if there are $u_0, \dots, u_k$ with $u = u_0$, $v = u_k$ and $u_i \xRightarrow[S]{} u_{i+1}$ for $0 \leq i \leq k-1$. We denote the transitive closure of $\xRightarrow[S]{}$ by $\xRightarrow[S]{+} = \bigcup_{k \geq 1} \xRightarrow[S]{k}$ and the reflexive and transitive closure by $\xRightarrow[S]{*} = \bigcup_{k \geq 0} \xRightarrow[S]{k}$. Moreover $\xLeftrightarrow[S]{*}$ is the reflexive, transitive, and symmetric closure of $\xRightarrow[S]{}$; it is the smallest congruence containing S. The set of *irreducible word* with respect to S is $\mathrm{IRR}(S) = \{w \in \Sigma^* \mid \text{there is no } v \text{ with } w \xRightarrow[S]{} v\}$.

Free groups. Let X be a set and $X^{-1} = \left\{ a^{-1} \;\middle|\; a \in X \right\}$ be a disjoint copy of X. We extend the mapping $a \mapsto a^{-1}$ to an involution without fixed points on $\Sigma = X \cup X^{-1}$ by $(a^{-1})^{-1} = a$ and finally to an involution without fixed points on Σ^* by $(a_1 a_2 \cdots a_n)^{-1} = a_n^{-1} \cdots a_2^{-1} a_1^{-1}$. For an integer $z < 0$ and $w \in \Sigma^*$ we write w^z for $(w^{-1})^{-z}$. The string rewriting system $S = \left\{ aa^{-1} \to 1 \;\middle|\; a \in \Sigma \right\}$ is strongly confluent and terminating meaning that for every word $w \in \Sigma^*$ there exists a unique word $\mathrm{red}(w) \in \mathrm{IRR}(S)$ with $w \xRightarrow[S]{*} \mathrm{red}(w)$ (for precise definitions see e.g. [7, 19]). Words from $\mathrm{IRR}(S)$ are called *freely reduced*. The system S defines the free group $F_X = \Sigma^*/S$ with basis X. More concretely, elements of F_X can be identified with freely reduced words, and the group product of $u, v \in \mathrm{IRR}(S)$ is defined by $\mathrm{red}(uv)$. With this definition $\mathrm{red} : \Sigma^* \to F_X$ becomes a monoid homomorphism that commutes with the involution $\cdot^{-1}$: $\mathrm{red}(w)^{-1} = \mathrm{red}(w^{-1})$ for all words $w \in \Sigma^*$. If $|X| = 2$, we write F_2 for F_X. It is known that for every countable set X, F_2 contains an isomorphic copy of F_X.

Finitely generated groups and the power word problem. A group G is called *finitely generated* if there exist a finite a finite set X and a surjective group homomorphism $h : F_X \to G$. In this situation, the set $\Sigma = X \cup X^{-1}$ is called a finite (symmetric) generating set for G. For words $u, v \in \Sigma^*$ we usually say that $u = v$ in G or $u =_G v$ in case $h(\mathrm{red}(u)) = h(\mathrm{red}(v))$. The *word problem* for the finitely generated group G, $\mathrm{WP}(G)$ for short, is defined as follows:

- input: a word $w \in \Sigma^*$.
- question: does $w =_G 1$ hold?

A *power word* (over Σ) is a tuple $(p_1, x_1, p_2, x_2, \dots, p_n, x_n)$ where $p_1, \dots, p_n \in \Sigma^*$ are words over the group generators (called the periods of the power word) and $x_1, \dots, x_n \in \mathbb{Z}$ are integers that are given in binary notation. Such a power word represents the word $p_1^{x_1} p_2^{x_2} \cdots p_n^{x_n}$. Quite often, we will identify the power word $(p_1, x_1, p_2, x_2, \dots, p_n, x_n)$ with the word $p_1^{x_1} p_2^{x_2} \cdots p_n^{x_n}$. Moreover, if $x_i = 1$, then we usually omit the exponent 1 in a power word. The *power word problem* for the finitely generated group G, PowerWP(G) for short, is defined as follows:

- input: a power word $(p_1, x_1, p_2, x_2, \dots, p_n, x_n)$.
- question: does $p_1^{x_1} p_2^{x_2} \cdots p_n^{x_n} =_G 1$ hold?

Due to the binary encoded exponents, a power word can be seen as a succinct description of an ordinary word. Hence, a priori, the power word problem for a group G could be computationally more difficult than the word problem. We will see examples of groups G, where PowerWP(G) is indeed more difficult than WP(G) (under standard assumptions from complexity theory), as well as examples of groups G, where PowerWP(G) and WP(G) are equally difficult.

Wreath products. Let G and H be groups. Consider the direct sum $K = \bigoplus_{h \in H} G_h$, where G_h is a copy of G. We view K as the set $G^{(H)}$ of all mappings $f \colon H \to G$ such that $\mathrm{supp}(f) := \{h \in H \mid f(h) \neq 1\}$ is finite, together with pointwise multiplication as the group operation. The set $\mathrm{supp}(f) \subseteq H$ is called the *support* of f. The group H has a natural left action on $G^{(H)}$ given by $hf(a) = f(h^{-1}a)$, where $f \in G^{(H)}$ and $h, a \in H$. The corresponding semidirect product $G^{(H)} \rtimes H$ is the (restricted) *wreath product* $G \wr H$. In other words:

- Elements of $G \wr H$ are pairs (f, h), where $h \in H$ and $f \in G^{(H)}$.
- The multiplication in $G \wr H$ is defined as follows: Let $(f_1, h_1), (f_2, h_2) \in G \wr H$. Then $(f_1, h_1)(f_2, h_2) = (f, h_1 h_2)$, where $f(a) = f_1(a) f_2(h_1^{-1} a)$.

Complexity. We assume that the reader is familiar with the complexity classes P, NP, and coNP and many-one reductions; see e.g. [2] for details. We use circuit complexity for classes below deterministic logspace (L for short).

A language $L \subseteq \{0,1\}^*$ is AC^0-Turing-reducible to $K \subseteq \{0,1\}^*$ if there is a family of constant-depth, polynomial-size Boolean circuits with oracle gates for K deciding L. More precisely, $L \subseteq \{0,1\}^*$ belongs to $\mathsf{AC}^0(K)$ if there exists a family $(C_n)_{n \geq 0}$ of circuits which, apart from the input gates $x_1, \dots, x_n$ are built up from *not*, *and*, *or*, and *oracle gates* for K (which output 1 if and only if their input is in K). All gates may have unbounded fan-in, but there is a polynomial bound on the number of gates and wires and a constant bound on the depth (length of a longest path from an input gate x_i to the output gate o). Finally, C_n accepts exactly the words from $L \cap \{0,1\}^n$, i.e., if each input gate x_i receives the input $a_i \in \{0,1\}$, then a distinguished output gate evaluates to 1 if and only if $a_1 a_2 \cdots a_n \in L$.

In the following, we only consider DLOGTIME-uniform $\mathsf{AC}^0(K)$ for which we write $\mathsf{uAC}^0(K)$. 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 the type of a given gate. For more details on these definitions we refer to [36]. If the languages K and L in the above definition of $\mathsf{uAC}^0(K)$ are defined over a non-binary alphabet Σ, then one first has to fix a binary encoding of words over Σ.

The class uTC^0 is defined as uAC^0(Majority) where Majority is the problem to determine whether the input contains more 1s than 0s. The class NC^1 is the class of languages accepted by Boolean circuits of bounded fan-in and logarithmic depth. When talking about hardness for uTC^0 or NC^1 we use uAC^0-Turing reductions unless stated otherwise. As a consequence of Barrington's theorem [3], we have $\mathsf{NC}^1 = \mathsf{uAC}^0(\mathrm{WP}(A_5))$ where A_5 is the alternating group over 5 elements [36, Corollary 4.54]. Moreover, the word problem for any finite group G is in NC^1. Robinson proved that the word problem for the free group F_2 is NC^1-hard [35], i.e., $\mathsf{NC}^1 \subseteq \mathsf{uAC}^0(\mathrm{WP}(F_2))$.

3 Results

In this section we state our (and prove the easy) results on the power word problem. Outlines of the proofs of Theorems 2, 8 and 9 can be found in Sections 4 and 5, respectively.

▶ **Theorem 1.** *If G is a finitely generated nilpotent group, then* $\textsc{PowerWP}(G)$ *is in* uTC^0.

Proof. In [33], the so-called word problem with binary exponents was shown to be in uTC^0 for finitely generated nilpotent groups. We can apply the same techniques as in [33]: we compute Mal'cev normal forms of all p_i [33, Theorem 5], then use the power polynomials from [33, Lemma 2] to compute Mal'cev normal forms with binary exponents of all $p_i^{x_i}$. Finally, we compute the Mal'cev normal form of $p_1^{x_1} \cdots p_n^{x_n}$ again using [33]. ◀

▶ **Theorem 2.** *The power word problem for a finitely generated free group is* AC^0*-Turing-reducible to the word problem for the free group* F_2.

Notice that if the free group has rank one, then the power word problem is in uTC^0 because iterated addition is in uTC^0.

▶ Remark 3. If the input is of the form $(p_1, x_1, p_2, x_2, \ldots, p_n, x_n)$ where all p_i are freely reduced, then the reduction in Theorem 2 is a uTC^0-many-one reduction.

▶ Remark 4. One can consider variants of the power word problem, where the exponents are not given in binary representation but in even more compact forms. *Power circuits* as defined in [32] are such a representation that allow non-elementary compression for some integers. The proof of Theorem 2 involves iterated addition and comparison of exponents. For power circuits iterated addition is in uAC^0 (just putting the power circuits next to each other), but comparison (even for equality) is P-complete [38]. Hence, the variant of the power word problem, where exponents are encoded with power circuits is P-complete for free groups.

▶ Remark 5. The proof of Theorem 2 can be easily generalized to free products. However, in order to have a simpler presentation we only state and prove the result for free groups and postpone the free product case to a future full version.

It is easy to see that the power word problem for every finite group belongs to NC^1. The following result generalizes this fact:

▶ **Theorem 6.** *Let G be finitely generated and let $H \leq G$ have finite index. Then* $\textsc{PowerWP}(G)$ *is* NC^1*-many-one-reducible to* $\textsc{PowerWP}(H)$.

Proof sketch. W.l.o.g. we can assume that H is a finitely generated normal subgroup and R is a finite set of representatives of $Q := G/H$ with $1 \in R$. As a first step we replace in the input power word every $p_i^{x_i}$ by $h_i^{y_i} p_i^{z_i}$ where $x_i = y_i \left|Q\right| + z_i$, $0 \leq z_i < \left|Q\right|$ and h_i is a word over the generators of H with $p_i^{\left|Q\right|} =_G h_i$. Moreover, we write $p_i^{z_i}$ as a word without exponents. Using the conjugate collection process from [35, Theorem 5.2], the result can be rewritten in the form hr where h is a power word in the subgroup H and $r \in R$. ◀

As an immediate consequence of Theorem 2, Theorem 6 and the NC^1-hardness of the word problem for F_2 [35, Theorem 6.3] we obtain:

▶ **Corollary 7.** *The power word problem for every finitely generated virtually free group is* AC^0*-Turing-reducible to the word problem for the free group* F_2.

▶ **Theorem 8.** *For every finitely generated abelian group G,* $\textsc{PowerWP}(G \wr \mathbb{Z})$ *is in* uTC^0.

▶ **Theorem 9.** *Let G be either a finite non-solvable group or a finitely generated free group of rank at least two. Then* $\textsc{PowerWP}(G \wr \mathbb{Z})$ *is* coNP*-complete.*

▶ **Theorem 10.** *The power word problem for the Grigorchuk group (as defined in [15] and also known as* first Grigorchuk group*) is* uAC^0*-many-one-reducible to its word problem.*

Theorem 10 applies only if the generating set contains a neutral letter. Otherwise, the reduction is in uTC^0. It is well-know that the word problem for the Grigorchuk group is in L (see e.g. [13]). Thus, also the power word problem is in L.

Proof sketch of Theorem 10. By [5, Theorem 6.6], every element of length N in the Grigorchuk group has order at most $CN^{3/2}$ for some constant C. Since the order of every element is a power of two, we can reduce all exponents modulo the smallest power of two $\geq CN^{3/2}$ where N is the length of the longest period p_i. After that the words are short and can be written without exponents. ◀

4 Proof of Theorem 2

The proof of Theorem 2 consists of two main steps: first we do some preprocessing leading to a particularly nice instance of the power word problem. While this preprocessing is simple from a theoretical point of view, it is where the main part of the workload is performed during the execution of the algorithm. Then, in the second step, all exponents are reduced to polynomial size. After this shortening process, the power word problem can be solved by the ordinary word problem. The most difficult part is to prove correctness of the shortening process. For this, we introduce a rewriting system over an extended alphabet of words with exponents. We outline the proof in a sequence of lemmas which all follow rather easily from the previous ones and we give some small hints how to prove the lemmas.

Preprocessing. We use the notations from the paragraph on free groups in Section 2. In particular, recall that $S = \left\{ aa^{-1} \to 1 \;\middle|\; a \in \Sigma \right\}$. Fix an arbitrary order on the input alphabet Σ. This gives us the lexicographic order on Σ^*, which is denoted by $\preceq$. Let $\Omega \subseteq \mathrm{IRR}(S) \subseteq \Sigma^*$ denote the set of words w such that

- w is non-empty,
- w is cyclically reduced (i.e, w cannot be written as aua^{-1} for $a \in \Sigma$),
- w is primitive (i.e, w cannot be written as u^n for $n \geq 2$),
- w is lexicographically minimal among all cyclic permutations of w and w^{-1} (i.e., $w \preceq uv$ for all $u, v \in \Sigma^*$ with $vu = w$ or $vu = w^{-1}$).

Notice that Ω consists of Lyndon words [29, Chapter 5.1] with the stronger requirement of being freely reduced, cyclically reduced and also minimal among the conjugacy class of the inverse. The first aim is to rewrite the input power word in the form

$$w = s_0 p_1^{x_1} s_1 \cdots p_n^{x_n} s_n \qquad \text{with } p_i \in \Omega \text{ and } s_i \in \mathrm{IRR}(S). \tag{1}$$

The reason for this lies in the following crucial lemma which essentially says that, if a long factor of $p_i^{x_i}$ cancels with some $p_j^{x_j}$, then already $p_i = p_j$. Thus, only the same p_i can cancel implying that we can make the exponents of the different p_i independently smaller.

▶ **Lemma 11.** *Let $p, q \in \Omega$, $x, y \in \mathbb{Z}$ and let v be a factor of p^x and w a factor of q^y. If $vw \xRightarrow[S]{*} 1$ and $|v| = |w| \geq |p| + |q| - 1$, then $p = q$.*

Proof. Since p and q are cyclically reduced, v and w are freely reduced, i.e., $v = w^{-1}$ as words. Thus, v has two periods $|p|$ and $|q|$. Since v is long enough, by the theorem of Fine and Wilf [10] it has also a period of $\gcd(|p|, |q|)$. This means that also p and q have period $\gcd(|p|, |q|)$ (since cyclic permutations of p and q are factors of v). Assuming $\gcd(|p|, |q|) < |p|$, would mean that p is a proper power contradicting the fact that p is primitive. Hence, $|p| = |q|$. Since $|v| \geq |p| + |q| - 1 = 2|p| - 1$, p is a factor of v, which itself is a factor of q^{-y}. Thus, p is a cyclic permutation of q or of q^{-1}. By the last condition on Ω, this implies $p = q$. ◀

▶ **Lemma 12.** *The following is in* $\mathsf{uAC}^0(\mathrm{WP}(F_2))$*: given a power word v, compute a power word w of the form* (1) *such that* $v =_{F_X} w$.

The proof of this lemma is straightforward using [39, Proposition 20] in order to compute freely reduced words. We call these steps the *preprocessing steps*. Henceforth, we will assume that the inputs for the power word problem are given in the form (1).

The symbolic reduction system. We define the infinite alphabet $\Delta = \Delta' \cup \Delta''$ with $\Delta' = \Omega \times (\mathbb{Z} \setminus \{0\})$ and $\Delta'' = \mathrm{IRR}(S) \setminus \{1\}$. We write p^x for $(p, x) \in \Delta'$. A word over Δ can be read as a word over Σ in the natural way. Formally, we can define a canonical projection $\pi : \Delta^* \to \Sigma^*$ that maps a symbol $a \in \Delta$ to the corresponding word over Σ, but most of the times we will not write π explicitly.

Whenever there is the risk of confusion, we write $|v|_\Sigma$ to denote the length of $v \in \Delta^*$ read over Σ (i.e., $|v|_\Sigma = |\pi(v)|$) whereas $|v|_\Delta$ is the length over Δ. Moreover, we denote the number of occurrences of letters from Δ' in w with $|w|_{\Delta'}$. Finally, for a symbol $s \in \Delta''$ define $\lambda(s) = |s|_\Sigma$ and for $p^x \in \Delta'$ set $\lambda(p^x) = |p|_\Sigma$. For $u = a_1 \cdots a_m \in \Delta^*$ with $a_i \in \Delta$ for $1 \leq i \leq m$ we define $\lambda(u) = \sum_{i=1}^m \lambda(a_i)$. Thus, $\lambda(u)$ is the number of letters from Σ required to write down u ignoring the binary exponents.

A word w as in (1), which has been preprocessed as in the previous section, can be viewed as word over Δ with $w \in ((\Delta'' \cup \{1\})\Delta')^*(\Delta'' \cup \{1\})$, $|w|_{\Delta'} = n$ and $|w|_\Delta \leq 2n+1$ (we only have $\leq$ because some s_i might be empty).

We define the infinite string rewriting system T over Δ^* by the following rewrite rules, where $p^x, p^y, q^y \in \Delta'$, $s, t \in \Delta''$, $r \in \Delta'' \cup \{1\}$, and $d, e \in \mathbb{Z}$. Here, p^0 is identified with the empty word. Note that the strings in the rewrite rules are over the alphabet Δ, whereas the strings in the if-conditions are over the alphabet Σ.

$$p^x p^y \to p^{x+y} \tag{2}$$

$$p^x q^y \to p^{x-d} r q^{y-e} \qquad \text{if } p \neq q, p^x q^y \xrightarrow[S]{+} p^{x-d} r q^{y-e} \in \mathrm{IRR}(S) \text{ for} \tag{3}$$

$$r = p'q' \text{ with } p' <_{\mathrm{pref}} p^{\mathrm{sign}(x)} \text{ and } q' <_{\mathrm{suff}} q^{\mathrm{sign}(y)}$$

$$st \to r \qquad \text{if } st \xrightarrow[S]{+} r \in \mathrm{IRR}(S) \tag{4}$$

$$p^x s \to p^{x-d} r \qquad \text{if } p^x s \xrightarrow[S]{+} p^{x-d} r \in \mathrm{IRR}(S) \text{ for} \tag{5}$$

$$r = p's' \text{ with } p' <_{\mathrm{pref}} p^{\mathrm{sign}(x)} \text{ and } s' <_{\mathrm{suff}} s$$

$$s p^x \to r p^{x-d} \qquad \text{if } s p^x \xrightarrow[S]{+} r p^{x-d} \in \mathrm{IRR}(S) \text{ for} \tag{6}$$

$$r = s'p' \text{ with } s' <_{\mathrm{pref}} s \text{ and } p' <_{\mathrm{suff}} p^{\mathrm{sign}(x)}$$

▶ **Lemma 13.** *The following length bounds hold in the above rules:*

- *in rule* (3)*:* $0 < |r|_\Sigma \leq |p|_\Sigma + |q|_\Sigma$, $|d| \leq |q|_\Sigma$, *and* $|e| \leq |p|_\Sigma$
- *in rules* (5) *and* (6)*:* $|d| \leq |s|_\Sigma$.

The inequalities $|d| \leq |q|_\Sigma$ and $|e| \leq |p|_\Sigma$ follow from Lemma 11. The other inequalities are obvious. The next lemma is also straightforward from the definition.

▶ **Lemma 14.** *For $u \in \Delta^*$ we have $u =_{F_X} 1$ if and only if $u \xRightarrow[T]{*} 1$.*

▶ **Lemma 15.** *Let $u \in \Delta^*$. If $u \xRightarrow[T]{*} v$, then $u \xRightarrow[T]{\leq k} v$ for $k = 2\,|u|_\Delta + 4\,|u|_{\Delta'} \leq 6\,|u|_\Delta$.*

Proof sketch. The proof is based on the fact that at most $2|u|_{\Delta'} - 3$ applications of rules of the form (3) can occur. These are the only length increasing rules. All other rules either decrease the number of non-reduced two-letter factors of u (this can happen at most $|u|_\Delta - 1$ times) or decrease the length of u (this can happen at most $|u|_\Delta + 2|u|_{\Delta'} - 3$ times). ◀

Consider a word $u \in \Delta^*$ and $p \in \Omega$. Let $\Delta_p = \{p^x \mid x \in \mathbb{Z} \setminus \{0\}\}$. We can write u uniquely as $u = u_0 p^{y_1} u_1 \cdots p^{y_m} u_m$ with $u_i \in (\Delta \setminus \Delta_p)^*$. We define $\eta_p^i(u) = \sum_{j=1}^{i} y_j$ and $\eta_p(u) = \eta_p^m(u)$. By Lemma 13 we know that all rules of T change $\eta_p(\cdot)$ by at most $\lambda(u)$. We can use this observation in order to show the next lemma by induction on k.

▶ **Lemma 16.** *Let $u \xRightarrow[T]{k} v$. Then for all $v' \leq_{pref} v$ with $v' \in \Delta^*$ there is some $u' \in \Delta^*$ with $u' \leq_{pref} u$ and $|\eta_p(u') - \eta_p(v')| \leq (k+1)^2\,\lambda(u)$.*

The shortened version of a word. Take a word $u \in \Delta^*$ and $p \in \Omega$ and write u as $u = u_0 p^{y_1} u_1 \cdots p^{y_m} u_m$ with $u_i \in (\Delta \setminus \Delta_p)^*$ (we are only interested in the case that p^x appears as a letter in u for some $x \in \mathbb{Z}$). Let $\mathcal{C}$ be a finite set of finite, non-empty, non-overlapping intervals of integers, i.e., we can write $\mathcal{C} = \{\, [\ell_j, r_j] \mid 1 \leq j \leq k \,\}$ for $k = |\mathcal{C}|$ and $\ell_j \leq r_j$ for all j. We can assume that the intervals are ordered increasingly, i.e., we have $r_j < \ell_{j+1}$. We set $d_j = r_j - \ell_j + 1 > 0$. We say that u is *compatible* with $\mathcal{C}$ if $\eta_p^i(u) \notin [\ell_j, r_j]$ for any i, j. If w is compatible with $\mathcal{C}$, we define the *shortened version* $\mathcal{S}_\mathcal{C}(u)$ of u: for $i \in \{1, \ldots, m\}$ we set

$$C_i = C_i(u) = \begin{cases} \{\, j \mid 1 \leq j \leq k, \eta_p^{i-1}(u) < \ell_j \leq r_j < \eta_p^i(u) \,\} & \text{if } y_i > 0 \\ \{\, j \mid 1 \leq j \leq k, \eta_p^{i}(u) < \ell_j \leq r_j < \eta_p^{i-1}(u) \,\} & \text{if } y_i < 0, \end{cases}$$

i.e., C_i collects all intervals between $\eta_p^{i-1}(u)$ and $\eta_p^i(u)$. Then $\mathcal{S}_\mathcal{C}(u)$ is defined by

$$\mathcal{S}_\mathcal{C}(u) = u_0 p^{z_1} u_1 \cdots p^{z_m} u_m \qquad \text{where}$$

$$z_i = y_i - \operatorname{sign}(y_i) \cdot \sum_{j \in C_i} d_j = \begin{cases} y_i - \sum_{j \in C_i} d_j & \text{if } y_i > 0, \\ y_i + \sum_{j \in C_i} d_j & \text{if } y_i < 0. \end{cases}$$

A straightforward computation yields the next lemma:

▶ **Lemma 17.** *For all i we have $z_i \neq 0$ and $\operatorname{sign}(z_i) = \operatorname{sign}(y_i)$. In particular, if $u \in \mathrm{IRR}(T)$, then also $\mathcal{S}_\mathcal{C}(u) \in \mathrm{IRR}(T)$.*

Furthermore, we define $\operatorname{dist}_p(u, \mathcal{C}) = \min\{\, |\eta_p^i(u) - x| \mid 0 \leq i \leq m, x \in [\ell, r] \in \mathcal{C} \,\}$. Note that $\operatorname{dist}_p(u, \mathcal{C}) > 0$ if and only if u is compatible with $\mathcal{C}$. Moreover, if $\operatorname{dist}_p(u, \mathcal{C}) = a$, $v = v_0 p^{z_1} v_1 \cdots p^{z_m} v_m$, and $|\eta_p^i(u) - \eta_p^i(v)| \leq b$ for all $i \leq m$, then $\operatorname{dist}_p(v, \mathcal{C}) \geq a - b$.

▶ **Lemma 18.** *If $\operatorname{dist}_p(u, \mathcal{C}) > (k+1)^2\,\lambda(u)$ and $u \xRightarrow[T]{k} v$, then $\mathcal{S}_\mathcal{C}(u) \xRightarrow[T]{k} \mathcal{S}_\mathcal{C}(v)$.*

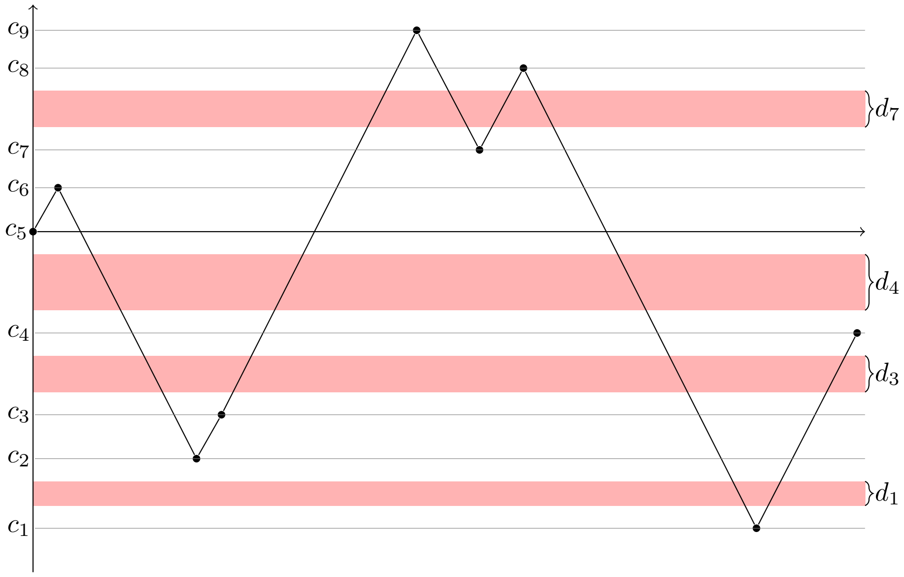


Figure 1 The red shaded parts represent the intervals from the set $\mathcal{C}_{u,p}^K$ in (7). The differences $c_3 - c_2$, $c_6 - c_5$, $c_7 - c_6$ and $c_9 - c_8$ are strictly smaller than $2K$.

Proof sketch. The first step for proving this lemma is to show that if $\operatorname{dist}_p(u, \mathcal{C}) > \lambda(u)$ and $u \underset{T}{\Longrightarrow} v$, then $S_{\mathcal{C}}(u) \underset{T}{\Longrightarrow} S_{\mathcal{C}}(v)$. To see this this, we distinguish between the rules applied: When applying one of the rules (3)–(6), we have $C_i(u) = C_i(v)$ for all i since the exponents are only changed slightly. Thus, the shortening process does the same on v as on u. When applying a rule (2), the exponents are added, which is compatible with the shortening process. Now we obtain the lemma by induction on k. In order to see that $\operatorname{dist}_p(u, \mathcal{C}) > \lambda(u)$ is satisfied in the inductive step, we use Lemma 16. ◀

We define a set of intervals which should be "cut out" from u as follows: We write $\{ c_1, \ldots, c_l \} = \{ \eta_p^i(u) \mid 0 \leq i \leq m \}$ with $c_1 < \cdots < c_l$ and we set

$$\mathcal{C}_{u,p}^K = \{ [c_j + K, c_{j+1} - K] \mid 1 \leq j \leq l-1, c_{j+1} - c_j \geq 2K \}. \tag{7}$$

Notice that $\operatorname{dist}_p(u, \mathcal{C}_{u,p}^K) = K$ (given that $\mathcal{C}_{u,p}^K \neq \emptyset$). The situation is shown in Figure 1.

▶ **Proposition 19.** *Let $p \in \Omega$, $u = u_0 p^{y_1} u_1 \cdots p^{y_m} u_m \in \Delta^*$ with $u_i \in (\Delta \setminus \Delta_p)^*$, and $K = (6\,|u|_\Delta + 1)^2\, \lambda(u) + 1$. Then $u =_{F_X} 1$ if and only if $S_{\mathcal{C}}(u) =_{F_X} 1$ for $\mathcal{C} = \mathcal{C}_{u,p}^K$.*

Proof. By Lemma 14 we have $u =_{F_X} 1$ if and only if $u \overset{*}{\underset{T}{\Longrightarrow}} 1$. Let $k = 6\,|u|_\Delta$. By Lemma 15, for all $u \overset{*}{\underset{T}{\Longrightarrow}} v$ we have $u \overset{\leq k}{\underset{T}{\Longrightarrow}} v$. By the choice of $\mathcal{C}$, we have $\operatorname{dist}_p(u, \mathcal{C}) > (k+1)^2\, \lambda(u)$. Hence, we can apply Lemma 18, which implies that $S_{\mathcal{C}}(u) \overset{*}{\underset{T}{\Longrightarrow}} S_{\mathcal{C}}(v)$ where v is a T-reduced (thus freely reduced) word for u. Clearly, if v is the empty word, then $S_{\mathcal{C}}(v)$ will be the empty word. On the other hand, if v is non-empty, then $S_{\mathcal{C}}(v)$ is non-empty and T-reduced by Lemma 17. Hence, we have $u =_{F_X} 1$ if and only if $S_{\mathcal{C}}(u) =_{F_X} 1$. ◀

▶ **Lemma 20.** *Let p, u, K, and $\mathcal{C}$ be as in Proposition 19 and $S_{\mathcal{C}}(u) = u_0 p^{z_1} u_1 \cdots p^{z_m} u_m$. Then $|z_i| \leq m \cdot (2 \cdot (6\,|u|_\Delta + 1)^2 \cdot \lambda(u) + 1)$ for all $1 \leq i \leq m$.*

Proof of Theorem 2. We start with the preprocessing as described in Lemma 12 leading to a word $w = s_0 p_1^{x_1} s_1 \cdots p_n^{x_n} s_n$ with $p_i \in \Omega$ and $s_i \in \mathrm{IRR}(S)$ as in (1). After that we apply the shortening procedure for all $p \in \{\, p_i \mid 1 \leq i \leq n \,\}$. This can be done in parallel for all p, as the outcome of the shortening only depends on the p-exponents. By Lemma 20 this leads to a word $\hat{w}$ of polynomial length. Finally, we can test whether $\hat{w} =_{F_X} 1$ using one oracle gate for $\mathrm{WP}(F_2)$ (recall that F_2 contains a copy of F_X). The computations for shortening only involve iterated addition (and comparisons of integers), which is in uTC^0 and, thus, can be solved in uAC^0 with oracle gates for $\mathrm{WP}(F_2)$. ◀

5 The power word problem in wreath products

The goal of this section is to prove Theorems 8 and 9. We first fix some notation. Let G be a finitely generated group with the finite symmetric generating set Σ. For $\mathbb{Z}$ we fix the generator a. Hence $\Sigma \cup \{a, a^{-1}\}$ is a symmetric generating set for the wreath product $G \wr \mathbb{Z}$. For a word $w = v_0 a^{e_1} v_1 \cdots a^{e_n} v_n$ with $e_i \in \{-1, 1\}$ and $v_i \in \Sigma^*$ let $\sigma(w) = e_1 + \cdots + e_n$. With $I(w)$ we denote the interval $[b, c] \subseteq \mathbb{Z}$, where b (resp., c) is the minimal (resp., maximal) integer of the form $e_1 + \cdots + e_i$ for $0 \leq i \leq n$. Note that if w represents $(f, d) \in G \wr \mathbb{Z}$, then $d = \sigma(w)$, $\mathrm{supp}(f) \subseteq I(w)$ and $0, d \in I(w)$.

Periodic words over groups. We recall a construction from [12]. With G^+ we denote the set of all tuples $(g_0, \ldots, g_{q-1})$ over G of arbitrary length $q \geq 1$. With G^ω we denote the set of all mappings $f : \mathbb{N} \to G$. Elements of G^ω can be seen as infinite sequences (or words) over the set G. We define the binary operation $\otimes$ on G^ω by pointwise multiplication: $(f \otimes g)(n) = f(n)g(n)$. The identity element is the mapping id with $\mathrm{id}(n) = 1$ for all $n \in \mathbb{N}$. For $f_1, f_2, \ldots, f_n \in G^\omega$ we write $\bigotimes_{i=1}^n f_i$ for $f_1 \otimes f_2 \otimes \cdots \otimes f_n$. If G is abelian, we write $\sum_{i=1}^n f_i$ for $\bigotimes_{i=1}^n f_i$. A function $f \in G^\omega$ is periodic with period $q \geq 1$ if $f(k) = f(k+q)$ for all $k \geq 0$. In this case, f can specified by the tuple $(f(0), \ldots, f(q-1))$. Vice versa, a tuple $u = (g_0, \ldots, g_{q-1}) \in G^+$ defines the periodic function $f_u \in G^\omega$ with $f_u(n \cdot q + r) = g_r$ for $n \geq 0$ and $0 \leq r < q$. One can view this mapping as the sequence u^ω obtained by taking infinitely many repetitions of u. Let G^ρ be the set of all periodic functions from G^ω. If f_1 is periodic with period q_1 and f_2 is periodic with period q_2, then $f_1 \otimes f_2$ is periodic with period $q_1 q_2$ (in fact, $\mathrm{lcm}(q_1, q_2)$). Hence, G^ρ forms a countable subgroup of G^ω. Note that G^ρ is not finitely generated: The subgroup generated by elements $f_i \in G^\rho$ with period q_i $(1 \leq i \leq n)$ contains only functions with period $\mathrm{lcm}(q_1, \ldots, q_n)$. For $n \geq 0$ we define the subgroup G^ρ_n of all $f \in G^\rho$ with $f(k) = 1$ for all $0 \leq k \leq n-1$. We consider the uniform membership problem for subgroups G^ρ_n, $\textsc{Membership}(G^\rho_*)$ for short:

- input: tuples $u_1, \ldots, u_n \in G^+$ (elements of G are represented by finite words over Σ) and a binary encoded number m.
- question: does $\bigotimes_{i=1}^n f_{u_i}$ belong to G^ρ_m?

The following result was shown in [12]:

▶ **Theorem 21.** *For every finitely generated abelian group G, $\textsc{Membership}(G^\rho_*)$ is in uTC^0.*

▶ **Lemma 22.** *Let $w \in (\Sigma \cup \{a, a^{-1}\})^*$ with $\sigma(w) \neq 0$, $n \geq 1$, and $I(w^n) = [b, c]$. Moreover, let $s = c - b + 1$ be the size of the interval $I(w)$ and let $(g, n \cdot \sigma(w)) \in G \wr \mathbb{Z}$ be the group element represented by w^n. Then g is periodic on the interval $[b + s, c - s]$ with period $|\sigma(w)|$.*

▶ **Example 23.** Let us consider the wreath product $\mathbb{Z} \wr \mathbb{Z}$ and let the left copy of $\mathbb{Z}$ in the wreath product be generated by b. Consider the word $w = ba^{-1}babab^3ab^3ab^5a^{-1}b$ and let $n = 8$. We have $\sigma(w) = 2$ and $I(w) = [-1, 3]$. Moreover, w represents the group element

$(f, 2)$ with $f(-1) = 1$, $f(0) = 2$, $f(1) = 3$, $f(2) = 4$, and $f(3) = 5$. Let us now consider the word w^8. The following diagram shows how to obtain the corresponding element of $\mathbb{Z} \wr \mathbb{Z}$:

-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	2	3	4	5														
		1	2	3	4	5												
				1	2	3	4	5										
						1	2	3	4	5								
								1	2	3	4	5						
										1	2	3	4	5				
												1	2	3	4	5		
														1	2	3	4	5
1	2	4	6	9	6	9	6	9	6	9	6	9	6	9	6	8	4	5

We have $I(w^8) = [-1, 17]$ and $\sigma(w^8) = 8\sigma(w) = 16$. If $(g, 16)$ is the group element represented by w^8, then the function g is periodic on the interval $[2, 14]$ (which includes the interval $[-1 + s, 17 - s]$, where $s = |I(w)| = 5$) with period 2.

Proofs of Theorem 8 and 9. A conjunctive truth-table reduction is a Turing reduction where the output is the conjunction over the outputs of all oracle gates.

▶ **Proposition 24.** *For every finitely generated group G,* PowerWP$(G \wr \mathbb{Z})$ *is conjunctive truth-table* uTC^0*-reducible to* Membership(G_*^ρ) *and* PowerWP(G).

Proof sketch. Let $w = u_1^{x_1} u_2^{x_2} \cdots u_k^{x_k}$ be the input power word and let $(f, d) \in G \wr \mathbb{Z}$ be the element represented by w. We can check in uTC^0 whether $d = 0$. The difficult part is to check whether f is the zero-mapping. For this we compute an interval I (of exponential size) that contains the support of f. We then partition I into two sets C and $I \setminus C$. The set C has polynomial size and we can check whether f is the zero-mapping on C using polynomially many oracle calls to PowerWP(G). The complement $I \setminus C$ can be written as a union of polynomially many intervals. The crucial property of C is that on each of these intervals f can be written as a sum of periodic sequences; for this we use Lemma 22. Using oracle calls to Membership(G_*^ρ) allows us to check whether f is the zero mapping on $I \setminus C$. ◀

Since for a finitely generated abelian group G, one can solve PowerWP(G) in uTC^0, Theorem 8 is a consequence of Proposition 24 and Theorem 21.

We split the proof of Theorem 9 into three propositions: one for the upper bound and two for the lower bounds. It is straightforward to show that if the word problem for the finitely generated group G belongs to coNP, then also Membership(G_*^ρ) belongs to coNP. Since coNP is closed under conjunctive truth-table uTC^0-reducibility, Proposition 24 yields:

▶ **Proposition 25.** *Let G be a finitely generated group such that* PowerWP(G) *belongs to* coNP*. Then also* PowerWP$(G \wr \mathbb{Z})$ *belongs to* coNP.

▶ **Proposition 26.** *If G is a finite, non-solvable group,* PowerWP$(G \wr \mathbb{Z})$ *is* coNP*-hard.*

Proof sketch. Barrington [4] proved the following result: Let C be a fan-in two boolean circuit of depth d with n input gates $x_1, \ldots, x_n$. From C one can compute a sequence of triples (a so-called G-program) $P_C = (k_1, g_1, h_1)(k_2, g_2, h_2) \cdots (k_\ell, g_\ell, h_\ell) \in ([1, n] \times G \times G)^*$ of length $\ell \leq (4|G|)^d$ such that for every input valuation $v : \{x_1, \ldots, x_n\} \to \{0, 1\}$ the following two statements are equivalent:

(a) C evaluates to 0 under the input valuation v.

(b) $c_1 c_2 \cdots c_\ell = 1$ in G, where $c_i = g_i$ if $v(x_{k_i}) = 0$ and $c_i = h_i$ if $v(x_{k_i}) = 1$.

This G-program is constructed as a sequence of iterated commutators, based on the observation that $[g,h]=1$ if and only if $g=1$ or $h=1$ (given some reasonable assumptions on g and h). Every formula C in conjunctive normal form can be written as a circuit of depth $\mathcal{O}(\log|C|)$. Hence the G-program P_C has length polynomial in $|C|$. From [4] it is easy to see that on input of the formula C, the G-program P_C can be computed in logspace.

Let $P_C=(k_1,g_1,h_1)\cdots(k_\ell,g_\ell,h_\ell)$ and $x_1,\dots,x_n$ be the variables in C. We compute in logspace the n first primes $p_1,\dots,p_n$ and $M=\prod_{i=1}^n p_i$ (the latter in binary notation). Let a denote the generator of $\mathbb{Z}$ in the wreath product $G\wr\mathbb{Z}$. We now compute for every $1\le i\le\ell$ the power word $w_i=(h_i(ag_i)^{p_{k_i}-1}a)^{M/p_{k_i}}a^{-M}$ and set $w_C=w_1w_2\cdots w_\ell$. The group element of $G\wr\mathbb{Z}$ represented by w_C is of the form $(f,0)$.

We claim that $w_C=1$ in $G\wr\mathbb{Z}$ if and only if C is unsatisfiable: For a number $z\in[0,M-1]$ we define the valuation $v_z:\{x_1,\dots,x_n\}\to\{0,1\}$ by $v_z(x_i)=1$ if $z\equiv 0 \mod p_i$ and $v_z(x_i)=0$ otherwise. By the Chinese remainder theorem, for every valuation $v:\{x_1,\dots,x_n\}\to\{0,1\}$ there exists $z\in[0,M-1]$ with $v=v_z$. Based on the above statements (a) and (b), the final step of the proof checks that $f(z)=1$ if and only if C evaluates to 0 under v_z. ◀

▶ **Proposition 27.** *Let F be a finitely generated free group of rank at least two. Then* PowerWP$(F\wr\mathbb{Z})$ *is* coNP*-hard.*

The proof is almost the same as for Proposition 26. The difference is that we mimic Robinson's proof that the word problem for F_2 is NC^1-hard [35] instead of Barrington's result.

6 Further Research

We conjecture that the method of Section 4 can be generalized to right-angled Artin groups (RAAGs – also known as graph groups) and hyperbolic groups, and hence that the power word problem for a RAAG (resp., hyperbolic group) G is AC^0-Turing-reducible to the word problem for G. One may also try to prove transfer results for the power word problem with respect to group theoretical constructions, e.g., graph products, HNN extensions and amalgamated products over finite subgroups. For finitely generated linear groups, the power word problem leads to the problem of computing matrix powers with binary encoded exponents. The complexity of this problem is open; variants of this problem have been studied in [1, 11].

Another open question is what happens if we allow nested exponents. We conjecture that in the free group for any nesting depth bounded by a constant the problem is still in $\mathsf{uAC}^0(\mathrm{WP}(F_2))$. However, for unbounded nesting depth it is not clear what happens: we only know that it is in P since it is a special case of the compressed word problem; but it still could be in $\mathsf{uAC}^0(\mathrm{WP}(F_2))$ or it could be P-complete or somewhere in between.

References

1 Eric Allender, Nikhil Balaji, and Samir Datta. Low-Depth Uniform Threshold Circuits and the Bit-Complexity of Straight Line Programs. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science, MFCS 2014, Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 13–24. Springer-Verlag, 2014. doi:10.1007/978-3-662-44465-8_2.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

3 David A. Mix Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 1–5. ACM, 1986. doi:10.1145/12130.12131.

4 David A. Mix Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989. doi:10.1016/0022-0000(89)90037-8.

5 Laurent Bartholdi, Rostislav I. Grigorchuk, and Zoran Šuniḱ. Branch groups. In *Handbook of algebra, Vol. 3*, pages 989–1112. Elsevier/North-Holland, Amsterdam, 2003. doi:10.1016/S1570-7954(03)80078-5.

6 Martin Beaudry, Pierre McKenzie, Pierre Péladeau, and Denis Thérien. Finite Monoids: From Word to Circuit Evaluation. *SIAM Journal on Computing*, 26(1):138–152, 1997.

7 Ron Book and Friedrich Otto. *String-Rewriting Systems*. Springer-Verlag, 1993.

8 William W. Boone. The word problem. *Annals of Mathematics*, 70(2):207–265, 1959.

9 Max Dehn. Ueber unendliche diskontinuierliche Gruppen. *Mathematische Annalen*, 71:116–144, 1911.

10 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16:109–114, 1965.

11 Esther Galby, Joël Ouaknine, and James Worrell. On Matrix Powering in Low Dimensions. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *LIPIcs*, pages 329–340. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.329.

12 Moses Ganardi, Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack Problems for Wreath Products. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, volume 96 of *LIPIcs*, pages 32:1–32:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. URL: http://www.dagstuhl.de/dagpub/978-3-95977-062-0.

13 Max Garzon and Yechezkel Zalcstein. The complexity of Grigorchuk groups with application to cryptography. *Theoretical Computer Science*, 88(1):83–98, 1991.

14 Guoqiang Ge. Testing Equalities of Multiplicative Representations in Polynomial Time (Extended Abstract). In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science, FOCS 1993*, pages 422–426. IEEE Computer Society, 1993.

15 Rostislaw I. Grigorchuk. Burnside's problem on periodic groups. *Functional Analysis and Its Applications*, 14:41–43, 1980.

16 Yuri Gurevich and Paul Schupp. Membership problem for the modular group. *SIAM Journal on Computing*, 37:425–459, 2007.

17 Derek Holt. Word-hyperbolic groups have real-time word problem. *International Journal of Algebra and Computation*, 10:221–227, 200.

18 Derek Holt, Markus Lohrey, and Saul Schleimer. Compressed Decision Problems in Hyperbolic Groups. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPIcs*, pages 37:1–37:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.37.

19 Matthias Jantzen. *Confluent String Rewriting*, volume 14 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.

20 Daniel König and Markus Lohrey. Evaluation of Circuits Over Nilpotent and Polycyclic Groups. *Algorithmica*, 80(5):1459–1492, 2018. doi:10.1007/s00453-017-0343-z.

21 Daniel König and Markus Lohrey. Parallel identity testing for skew circuits with big powers and applications. *International Journal of Algebra and Computation*, 28(6):979–1004, 2018. doi:10.1142/S0218196718500431.

22 Richard J. Lipton and Yechezkel Zalcstein. Word Problems Solvable in Logspace. *Journal of the ACM*, 24:522–526, 1977.

23 Markus Lohrey. Decidability and complexity in automatic monoids. *International Journal of Foundations of Computer Science*, 16(4):707–722, 2005.

24 Markus Lohrey. Word Problems and Membership Problems on Compressed Words. *SIAM Journal on Computing*, 35(5):1210–1240, 2006. doi:10.1137/S0097539704445950.

25 Markus Lohrey. *The Compressed Word Problem for Groups*. Springer Briefs in Mathematics. Springer-Verlag, 2014. doi:10.1007/978-1-4939-0748-9.

26 Markus Lohrey and Saul Schleimer. Efficient computation in groups via compression. In *Proceedings of the 2nd International Symposium on Computer Science in Russia, CSR 2007*, volume 4649 of *Lecture Notes in Computer Science*, pages 249–258. Springer-Verlag, 2007.

27 Markus Lohrey and Armin Weiß. The power word problem. *CoRR*, abs/1904.08343, 2019. URL: `https://arxiv.org/abs/1904.08343`.

28 Markus Lohrey and Georg Zetzsche. Knapsack in Graph Groups. *Theory of Computing Systems*, 62(1):192–246, 2018. `doi:10.1007/s00224-017-9808-3`.

29 M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and Its Applications*. Addison-Wesley, 1983. Reprinted by *Cambridge University Press*, 1997.

30 Alexei Miasnikov, Svetla Vassileva, and Armin Weiß. The Conjugacy Problem in Free Solvable Groups and Wreath Products of Abelian Groups is in TC^0. *Theory of Computing Systems*, 63(4):809–832, 2018. `doi:10.1007/s00224-018-9849-2`.

31 Alexei Myasnikov, Andrey Nikolaev, and Alexander Ushakov. Knapsack Problems in Groups. *Mathematics of Computation*, 84(292):987–1016, 2015.

32 Alexei Myasnikov, Alexander Ushakov, and Won Dong-Wook. Power Circuits, exponential Algebra, and Time Complexity. *International Journal of Algebra and Computation*, 22(6):3–53, 2012.

33 Alexei Myasnikov and Armin Weiß. TC^0 circuits for algorithmic problems in nilpotent groups. In *Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017*, volume 83 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.MFCS.2017.23`.

34 Pyotr S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *Trudy Mat. Inst. Steklov*, pages 1–143, 1955. In Russian.

35 David Robinson. *Parallel Algorithms for Group Word Problems*. PhD thesis, University of California, San Diego, 1993.

36 Heribert Vollmer. *Introduction to Circuit Complexity*. Springer-Verlag, 1999.

37 Stephan Waack. The parallel complexity of some constructions in combinatorial group theory. *Journal of Information Processing and Cybernetics*, 26(5-6):265–281, 1990.

38 Armin Weiß. *On the Complexity of Conjugacy in Amalgamated Products and HNN Extensions*. Dissertation, Institut für Formale Methoden der Informatik, Universität Stuttgart, 2015.

39 Armin Weiß. A Logspace Solution to the Word and Conjugacy problem of Generalized Baumslag-Solitar Groups. In *Algebra and Computer Science*, volume 677 of *Contemporary Mathematics*, pages 185–212. American Mathematical Society, 2016.

Upper Bounds on the Length of Minimal Solutions to Certain Quadratic Word Equations

Joel D. Day[1]
Loughborough University, UK
Kiel University, Germany
J.Day@lboro.ac.uk

Florin Manea
Kiel University, Germany
flm@informatik.uni-kiel.de

Dirk Nowotka
Kiel University, Germany
dn@informatik.uni-kiel.de

Abstract

It is a long standing conjecture that the problem of deciding whether a quadratic word equation has a solution is in NP. It has also been conjectured that the length of a minimal solution to a quadratic equation is at most exponential in the length of the equation, with the latter conjecture implying the former. We show that both conjectures hold for some natural subclasses of quadratic equations, namely the classes of regular-reversed, k-ordered, and variable-sparse quadratic equations. We also discuss a connection of our techniques to the topic of unavoidable patterns, and the possibility of exploiting this connection to produce further similar results.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words; Theory of computation → Problems, reductions and completeness

Keywords and phrases Quadratic Word Equations, Length Upper Bounds, NP, Unavoidable Patterns

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.44

Funding Florin Manea's work was supported by the DFG grant MA 5725/2-1.

1 Introduction

A *word equation* is an equation $\alpha = \beta$ in which the two sides, α and β, are words consisting of letters from a *terminal alphabet* $\Sigma = \{\mathtt{a}, \mathtt{b}, \dots\}$ and *variables* from a set $X = \{x, y, z, \dots\}$. It has a solution if the variables may be substituted for words over Σ in such a way that the two sides become identical. For example, the equation $\mathtt{ab}x\mathtt{b}y = x\mathtt{b}yx\mathtt{b}$ has a solution where x is substituted by $\mathtt{a}$, and y is substituted by $\mathtt{ab}$. Usually such a substitution is represented by a morphism $h : (X \cup \Sigma)^* \to \Sigma^*$ which preserves the terminal symbols (i.e. $h(\mathtt{a}) = \mathtt{a}$ for all $\mathtt{a} \in \Sigma$). Situated in the intersection between computer science and algebra, word equations are an important tool for describing structural relations between words, and as such are of interest in a variety of areas, ranging from combinatorial group and monoid theory [20, 19, 7], to unification [25, 12, 14], database theory [11, 10], model checking, verification and security, where there has been much interest recently in developing so-called string solvers capable of dealing with word equations such as HAMPI [17], CVC4 [3], Stranger [28], ABC [2], Norn [1], S3P [27] and Z3str3 [4]. Of course, not all equations have solutions (consider the trivial example $\mathtt{a}x = \mathtt{b}x$, or less trivial ones such as $\mathtt{abc}x = x\mathtt{cba}$), and the problem of deciding whether a given equation has a solution – the satisfiability problem – has been at the

[1] Corresponding author

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 44; pp. 44:1–44:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

centre of research on word equations since their inception. It is not difficult to see that the satisfiability problem contains an inherent degree of computational complexity. In particular, there are several simple reductions from NP-complete problems such as the membership problem for pattern languages [9, 8] and linear arithmetic. On the other hand, bounding the complexity from above has proven considerably more challenging. After considerable effort, Makanin [21] famously showed that the satisfiability problem for word equations is algorithmically decidable. This result was later improved by Plandowski [23] who gave an algorithm which requires only polynomial space, and more recently this has been refined further to linear space by Jeż [13, 15] using the elegant method of recompression. Nevertheless, determining the precise complexity, and in particular whether the problem is contained in NP, remains one of the outstanding open problems in the area.

One method for obtaining upper bounds on the complexity is to consider the lengths of minimal solutions – those for which no shorter solution exists to the same equation. Since it may easily be checked in polynomial time (in the length of the substitution) whether a given substitution satisfies a given word equation: simply apply the substitution to each side and compare the resulting words, we have a clear relation between the lengths of minimal solutions and the complexity of the satisfiability problem. If the minimal solutions are guaranteed to be short enough (e.g. polynomial in the length of the equation), we have a non-deterministic polynomial time algorithm which simply guesses a solution and then checks it. In fact, Plandowski and Rytter [24] showed that minimal solutions may be compressed substantially in such a way that their compressed versions may still be checked efficiently, so a similar approach works for equations whose minimal solutions are at most exponentially long in the length of the equation. Unfortunately, the best known upper bound on the length of minimal solutions to word equations in general is double exponential. It is worth pointing out that it is not difficult to construct examples of equations for which the lengths of minimal solutions are (single) exponential in the length of the equation.

In the absence of matching upper and lower bounds on the complexity for the whole class of word equations, it makes sense to first consider subclasses. For example, equations with at most two distinct variables (which may each occur multiple times) can be solved in polynomial time (see, e.g. [6]), and thus do not exhibit the same intractability as more general classes, while another class which is generally well understood is the class of equations in which variables may only occur on one side (i.e. either $\alpha \in \Sigma^*$ or $\beta \in \Sigma^*$). In this case, the satisfiability problem is simply the (NP-complete) problem of pattern matching with variables, for which the computational complexity has been studied extensively. Aside from these examples, however, there seem to be surprisingly few classes of equations for which the satisfiability problem is known to be contained in NP, especially amongst those with corresponding hardness results.

Quadratic word equations are word equations in which each variable may occur at most twice – although the number of variables is unrestricted. There are many reasons that the quadratic equations form a particularly interesting subclass of equations. On one hand, NP-hardness remains (see [8]) but also even for the very restricted case in which the variables must occur in exactly the same order on both sides and only the terminal symbols may vary (see [5]). On the other hand, unlike the general case, there is a straightforward proof that the satisfiability problem for quadratic word equations is decidable, using so-called Nielsen transformations (see [18]). Moreover, while examples of equations with three occurrences of each variable are known for which the minimal solutions are exponentially long in the length of the equation, no such examples are known for quadratic equations. However, while these results seem to indicate that quadratic equations may not be as complex as the general case, understanding quadratic equations, and in particular determining whether the satisfiability of quadratic equations is in NP, has proven exceptionally difficult and as with the general case, remains a long-standing open problem.

Our Contribution. In the current paper, we further develop a method for showing upper bounds on the length of solutions to quadratic word equations first introduced in [5], and use it to obtain such bounds for several subclasses of quadratic equations. The method extends the existing technique of *filling the positions* (see [16, 24]), and relies on arranging the individual positions of a solution, as referenced by their origin (e.g. the third letter in the second occurrence of the variable x) into chains, which may be represented as words (chain-words). This chains-representation of solutions is discussed in detail in Section 3.

We show firstly that quadratic equations with a high concentration of terminal symbols and variables occurring only once – with few variables occurring the maximal two times – have minimal solutions at length at most $n2^{2^{O(V^4)}}$ where V is the number of variables occurring twice and n is the length of the equation. Using the previously mentioned algorithm of Plandowski and Rytter, it follows that for the class of equations for which $V < \log(n)$, which we shall refer to as *variable-sparse* quadratic equations, the satisfiability problem is in NP. Moreover, by observing that variables occurring only once do not have a dramatic impact on the length of minimal solutions (Proposition 2), we also obtain that if V is bounded by a constant, the satisfiability problem may be solved in polynomial time.

As a straightforward consequence, we are also able to show that equations which may be obtained by concatenating many "small" quadratic equations (over disjoint, constant-size sets of variables) also have short solutions, thus may be solved in non-deterministic polynomial time. Such equations may be arbitrarily disordered at a local level – we do not restrict the structure of each individual equation before concatenating – but possess a global order in which the sets of variables must occur from left to right in each side of the equation. Since these equations, which we shall call k-ordered equations, generalise the regular-ordered equations considered in [5], we also get the corresponding NP-hardness lower bound in this case. Thus, the remaining "interesting", cases must possess some global disorder among the variables.

Our main result (Theorem 21) establishes upper bounds of $n2^{3n}$ on the length of minimal solutions to a subclass of quadratic equations for which this necessary separation of the variables is maximal, namely the class of regular-reversed equations, in which the variables occur in the opposite order on the LHS as on the RHS. We also show (Proposition 14) that this bound is close to the best that we may expect using our approach. The proof of Theorem 21, although technical, revolves around two simple operations on the chains-representation of a solution. The first is a compression of certain subchains, which produces a new, shorter solution to an equation of the same length. In a sense, this kind of compression generalises the Nielsen transformations mentioned previously. The second operation, given by Lemma 22 and taking advantage of the structure induced by the carefully chosen compression, is a simple deletion of a variable from the equation (and solution) such that the property of being chain-square-free is preserved.

The final part to our contribution focusses on potential forward steps in the general case, and as such begins to explore a connection to the topic of avoidability of patterns within the field of combinatorics on words. Our main result in this direction is a characterisation of possible chain words in the case of regular equations (equations in which each variable occurs at most once on each side) which is considerably simpler than the more abstract definition given in Section 3. This leaves open a particular interesting problem to determine whether there exist "long" non-repetitive (square-free) words satisfying this characterisation, where a negative answer, for an appropriate definition of "long", would, due to results presented in Section 3, yield that the satisfiability problem for this large subclass of quadratic word equations is in NP. We also discuss, with Lemma 27 as an example, how looking at various structures other than direct repetitions might also be sufficient to obtain the same result.

The rest of the paper is organised as follows. In Section 2, we establish the basic notations and definitions we will need, along with some preliminary results. In Section 3, we shall introduce the main framework used to establish our results. This section recaps the main construction (the chains representation of solutions) and lemma (Lemma 11) originally presented in [5] which form the basis of the framework, and mentions some new additional useful results such as Lemma 12 and Proposition 14. Our main results concerning upper bounds on the length of minimal solutions to equations are presented in Sections 4 and 5. Finally, in Section 6, we present our characterisation of chain-words and discuss the connection to the topic of avoidability of patterns and the resulting possibilities for exploiting Lemma 11 in Section 3 further.

2 Preliminaries

Let Σ be an alphabet. We denote by Σ^* the set of all words over Σ. The empty word is denoted ε. A word u is a *prefix* of a word w if there exists v such that $w = uv$. Similarly, u is a *suffix* of w if there exists v such that $w = vu$, and u is a *factor* of w if there exist v, v' such that $w = vuv'$. A *square* is a word ww for some $w \in \Sigma^* \backslash \{\varepsilon\}$. The length of a word w is denoted $|w|$, and the number of occurrences of a letter $\mathtt{a}$ in a word w is denoted $|w|_{\mathtt{a}}$. The i^{th} letter of w, as counted from the left, is denoted $w[i]$. The *reversal* of a word w is the word $w^R = w[|w|]w[|w|-1]\ldots w[2]w[1]$. For two alphabets Σ_1, Σ_2, a morphism $h : {\Sigma_1}^* \to {\Sigma_2}^*$ is a mapping such that, for all $u, v \in {\Sigma_1}^*$, $h(u)h(v) = h(uv)$. Thus, a morphism is uniquely defined by its image on each letter in Σ_1. In the rest of the paper, we shall distinguish between two alphabets: an (infinite) set X of *variables*, and a *terminal alphabet* Σ. We shall generally assume that the terminal alphabet contains at least two letters, but otherwise its cardinality will not be important for our purposes. For a word $\alpha \in (X \cup \Sigma)^*$, we shall denote the set $\{x \in X \mid |\alpha|_x \geq 1\}$ by $\mathrm{var}(\alpha)$ and the set $\{\mathtt{a} \in \Sigma \mid |\alpha|_{\mathtt{a}} \geq 1\}$ by $\mathrm{alph}(\alpha)$. We shall say a morphism $h : (X \cup \Sigma)^* \to \Sigma^*$ is a *substitution* if $h(\mathtt{a}) = \mathtt{a}$ for all $\mathtt{a} \in \Sigma$.

A *word equation* is a tuple (α, β) (usually written as $\alpha = \beta$) such that $\alpha, \beta \in (X \cup \Sigma)^*$. α and β are called the left hand side (LHS) and right hand side (RHS) respectively. Solutions are substitutions $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ such that $h(\alpha) = h(\beta)$. The length of a word equation $\alpha = \beta$ is $|\alpha\beta|$. The length of a solution h to the equation is $|h(\alpha)|$. A solution is *minimal* if no shorter solution exists. The *Satisfiability Problem* is the decision problem of determining whether, for a given word equation, there exists a solution. The following result of Plandowski and Rytter establishes a relationship between the length of minimal solutions (when they exist) and the computational complexity of the satisfiability problem.

▶ **Theorem 1** ([24]). *Suppose that for a given class of word equations, there exists a polynomial P such that any equation in the class which has a solution, has one whose length is at most $2^{P(n)}$ where n is the length of the equation. Then the satisfiability problem for that class is in* NP.

A word equation $\alpha = \beta$ is *quadratic* if $|\alpha|_x + |\beta|_x \leq 2$ for every $x \in X$. It is *regular* if $|\alpha|_x \leq 1$ and $|\beta|_x \leq 1$ for all $x \in X$. It is regular-ordered if it is regular, and the variables occur in the same order in both sides of the equation (i.e. there do not exist $x, y \in X$ and $\alpha', \beta' \in (X \cup \Sigma)^*$ such that $x\alpha'y$ is a factor of α and $y\beta'x$ is a factor of β). It was shown in [5] that the satisfiability for regular-ordered word equations is NP-hard.

Finally, we remark that, when considering asymptotic bounds on the length of minimal solutions to quadratic equations, it is not necessary to consider equations in which a variable occurs only once. Hence we shall, unless otherwise stated, consider classes of equations in which the variables occur exactly twice. All of our results providing upper bounds on the length of minimal solutions (and thus containment in NP) can easily be adapted for classes permitting variables which occur only once.

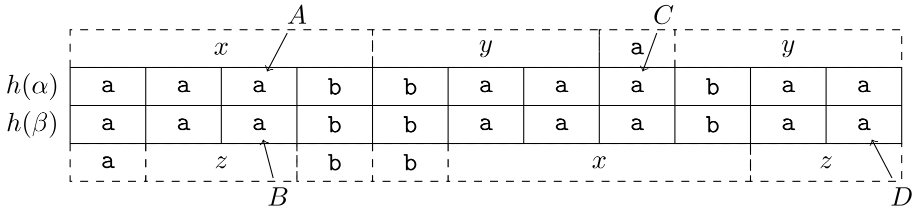
▶ **Proposition 2.** *Let $\alpha = \beta$ be a quadratic word equation and let $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ be a minimal solution. Let X_1 be the set of variables occurring exactly once in $\alpha\beta$ and let X_2 be the set of variables occurring twice. Let $g : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to (\mathrm{var}(\alpha\beta) \cup \Sigma)^*$ be the morphism such that $g(x) = h(x)$ if $x \in X_1$ and $g(x) = x$ otherwise. Then the equation $g(\alpha) = g(\beta)$ has length less than $2|\alpha\beta|$, and has a minimal solution $h' : (X_2 \cup \Sigma)^* \to \Sigma^*$ such that $|h'(g(\alpha))| = |h(\alpha)|$.*

3 A Method for Upper Bounds

Despite Theorem 1, showing inclusion in NP often remains a challenge, particularly among those classes for which the corresponding NP-hardness lower bound exists. In the present section, we outline a framework, first presented in [5] for reasoning about the (non)-minimality of solutions which we shall rely on in the proofs of most of our results. The main idea centers around the simple principle that a solution is not minimal if we can remove some parts of it to obtain another, shorter solution. While this is in itself a trivial statement, the consequences of our approach will lead, as we shall see in Section 6, to an entirely non-trivial method of reasoning generally about the lengths of minimal solutions to quadratic equations and a surprising link to the topic of avoidability of patterns, a central theme within combinatorics on words.

Positions in a Solution. Our approach extends the method of "filling the positions", used to determine whether a word equation has a solution when the lengths of the substitutions for the variables are given explicitly (see [16, 24] for an overview). In this respect, we must first specify precisely what we mean by a position. For our reasoning, it will be convenient to be able to reference parts of the substitution/solution both purely from their location in the full image, and relative to the part of the equation – an occurrence of a variable or terminal symbol – from which they originate. Accordingly, for an equation E given by $\alpha = \beta$ and a substitution $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ such that $|h(\alpha)| = |h(\beta)|$, we define the set of *absolute positions* of h w.r.t. E as $\mathcal{AP}_E^h = \{i \mid 1 \leq i \leq |h(\alpha)|\}$, and the set of *relative positions* (or just positions) of h w.r.t. E as $\mathcal{RP}_E^h = \{(x, i, d) \mid x \in \mathrm{var}(\alpha\beta) \cup \mathrm{alph}(\alpha\beta), 1 \leq i \leq |\alpha\beta|_x, \text{ and } 1 \leq d \leq |h(x)|\}$.

Intuitively, the relative position (x, i, d) corresponds to "the d^{th} letter in the image of the i^{th} occurrence of x", where occurrences are counted from left to right in $\alpha\beta$. As such, we have exactly two relative positions corresponding to each absolute position (one for the LHS and one for the RHS). Let $f_E^h : \mathcal{RP}_E^h \to \mathcal{AP}_E^h$ be the (non-injective) function mapping a relative position to the corresponding absolute position. More formally, for a relative position $r = (x, i, d)$, let $f_E^h(r)$ be the unique integer j such that $j - d = |h(\gamma)| \mod |h(\alpha)|$ where γ is the prefix of $\alpha\beta$ up to, but not including, the i^{th} occurrence of x (i.e. the longest prefix of $\alpha\beta$ such that $|\gamma|_x < i$). We shall say that a relative position $(x, i, d) \in \mathcal{RP}_E^h$ is a *terminal position* if $x \in \Sigma$ (in which case it is guaranteed that $d = 1$), and otherwise we shall say that the position is *non-terminal* and *belongs* to the variable x.



$h(\alpha)$	a	a	a	b	b	a	a	a	b	a	a
$h(\beta)$	a	a	a	b	b	a	a	a	b	a	a

Figure 1 The solution h given by $h(x) = \texttt{aaab}$, $h(y) = \texttt{baa}$ and $h(z) = \texttt{aa}$ to the equation E given by $xy\texttt{a}y = \texttt{a}z\texttt{bb}xz$. Each individual rectangle is a position in $\mathcal{RP}_E^h$. For example, A is the position $(x, 1, 3)$, while D is the position $(z, 2, 2)$, and C is the (terminal) position $(\texttt{a}, 1, 1)$. The positions A and B are neighbours. B and D are siblings, meaning that A is the successor of D.

Neighbours and Siblings. We shall now introduce the two relations on the set of relative positions used in the method of filling the positions. Firstly, we shall say that two relative positions r_1, r_2 are *siblings* if there exist x, d, i, i' with $i \neq i'$ such that $r_1 = (x, i, d)$ and $r_2 = (x, i', d)$. Secondly, we shall say that two relative positions r_1 and r_2 are *neighbours* if $f_E^h(r_1) = f_E^h(r_2)$ and $r_1 \neq r_2$. The following remarks are immediate:

▶ Remark 3. Each relative position $r = (x, i, d)$ has exactly one neighbour. It has exactly $|\alpha\beta|_x - 1$ siblings. In particular, if $\alpha = \beta$ is quadratic, then either r is a terminal position or r has at most one sibling.

While we will not give a full description of the method of filling the positions in the present paper, it essentially consists of constructing, for a given equation E and solution h, the equivalence relation $\mathcal{R}_E^h$ obtained by the reflexive and transitive closure of the union of the neighbour and sibling relations. The following remarks are standard facts regarding the method of filling the positions and can easily be verified.

▶ Remark 4. A substitution $h : (\text{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ is a solution to the equation E given by $\alpha = \beta$ if and only if $|h(\alpha)| = |h(\beta)|$ and, for any two positions $r_1 = (x, i, d), r_2 = (x', i', d')$ such that $(r_1, r_2) \in \mathcal{R}_E^h$, we have that $h(x)[d] = h(x')[d']$.

▶ Remark 5. If $h : (\text{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ is a solution to the equation E given by $\alpha = \beta$, and there exists an equivalence class of the relation $\mathcal{R}_E^h$ which does not contain a terminal position, then the positions belonging to this class can be substituted by any word without affecting the fact that h is a solution. In particular, they can be substituted by the empty word (i.e. removed altogether), so the solution is not minimal.

The Chains Representation and Chain-words. Suppose now that we have a quadratic equation E given by $\alpha = \beta$ and a solution $h : (\text{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ to E. From now on, we shall only consider quadratic equations. Then by Remark 3, we can organise the relative positions into sequences, or *chains* using the sibling and neighbour relations as follows.

▶ **Definition 6** (Chains Representation of a Solution). *A sequence $r_1, r_2, \ldots, r_n$ of positions from $\mathcal{RP}_E^h$ is a chain-sequence of h with respect to E if*

- *$r_1 = (x_1, i_1, d_1)$ where either x_1 is a terminal symbol from Σ or $|\alpha\beta|_{x_1} = 1$,*
- *r_2 is the neighbour of r_1,*
- *$r_n = (x_n, i_n, d_n)$ where either x_n is a terminal symbol from Σ or $|\alpha\beta|_{x_n} = 1$,*
- *for all j, $2 < j \leq n$, r_j is the neighbour of the sibling of r_{j-1}.*

The set of all chain-sequences is the chains-representation *of h with respect to E. For a position r in a chain, the next position r' in the chain, if it exists, is called the* successor *of r, while r is the* predecessor *of r'. For the sake of brevity, we shall usually refer to chain-sequences simply as "chains".*

▶ Remark 7. Throughout the rest of the paper, we shall only consider solutions for which all equivalence classes of the relation $\mathcal{R}_E^h$ contain a terminal position (otherwise, we can just erase the appropriate parts to obtain a shorter solution, see Remark 5). Under this assumption, all positions $r \in \mathcal{RP}_E^h$ which are not terminal and have a sibling r' will occur (exactly once) in exactly one chain, while terminal positions and positions without a sibling will occur twice (once at the start of a chain-sequence, and once at the end). Thus the length of the solution will be at most half the sum of the lengths of all chains in the chains representation.

▶ Remark 8. For every chain $r_1, r_2, \ldots, r_n$ in the chains representation, there will also exist a dual chain $r_n, \overline{r_{n-1}}, \ldots, \overline{r_2}, r_1$ where, for $2 \leq i \leq n-1$, $\overline{r_i}$ denotes the sibling of r_i. Similarly, for every subchain $r_i, r_{i+1}, \ldots, r_j$ such that r_i and r_j are not terminal positions, there exists a dual subchain $\overline{r_j}, \overline{r_{j-1}}, \ldots, \overline{r_i}$, obtained by reversing and swapping each position for its neighbour. Each equivalence class of the relation $\mathcal{R}_E^h$ corresponds exactly to the positions contained within one chain and its dual.

▶ Remark 9. In the particular case of regular equations, the sibling of a position will always occur on the opposite side of the equation. Since the same is always true by definition for the neighbour of a position, it follows that the successor in the chains representation of a position will belong to the same side. In particular, every position (x, i, d) in the chain which belongs to a variable, with the exception of the first, will have the same value i.

▶ **Definition 10** (Chain-Words and Similarity). *Let Γ_E^h be the set $\{(x, i) \mid x \in \mathrm{var}(\alpha\beta)$ and $i \in \{1, 2\}\}$, and let $\rho : \mathcal{RP}_E^h \to \Gamma_E^h$ be the projection of $\mathcal{RP}_E^h$ onto the first two elements (so that $\rho((x, i, d)) = (x, i)$). For each chain-sequence $C = r_1, r_2, \ldots, r_n$ of h with respect to E, the chain-word induced by C is the word $w = \rho(r_2)\rho(r_3)\ldots\rho(r_{n-1})$ (over the alphabet Γ_E^h). The set of all chain-words induced by chain-sequences of h w.r.t. E is denoted $\Delta_{h,E}$. We shall say that two (sub)chains are* similar *if they induce the same chain-words.*

The following lemma is the main motivation for extending the framework of filling the positions to take into account the additional order expressed in the chains representation. It shall provide the basis for our reasoning later that long solutions to (certain) equations cannot be minimal due to the fact that their chains must contain some repetitive structure.

▶ **Lemma 11** ([5]). *Let E be a quadratic word equation given by $\alpha = \beta$ and let $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ be a solution to E. If there exists a chain-word $w \in \Delta_{h,E}$ such that w contains a square, then h is not minimal.*

If, for a solution $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ to an equation $\alpha = \beta$, there does not exist a chain-word which contains a square, we shall say that the chains-representation is *square-free*, or that the solution h is *chain-square-free*. The simplest examples of solutions which are not chain-square-free are when two occurrences of the same variable overlap (i.e. the parts of the solution word corresponding to the occurrences of these variables intersect), leading to a square of length one in a chain-word. Generalising this slightly to squares of length two, the following lemma gives another simple example of when a solution is not chain-square-free, which is used in the proofs of our later results.

▶ **Lemma 12.** *Let $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ be a solution to a quadratic equation E given by $\alpha = \beta$. Let $i, i', j, j' \in \{1, 2\}$ with $i \neq i'$ and $j \neq j'$, and let $x, y \in \mathrm{var}(\alpha\beta)$. If $f_E^h((x, i, 1)) \leq f_E^h((y, j, 1)) \leq f_E^h((x, i, |h(x)|)) \leq f_E^h((y, j, |h(y)|))$ and $f_E^h((x, i', 1)) \leq f_E^h((y, j', 1)) \leq f_E^h((x, i', |h(x)|)) \leq f_E^h((y, j', |h(y)|))$, then h is not chain-square-free.*

The following gives a full example of a solution to a quadratic equation along with its chains representation. The solution is not chains-square-free, and thus a shorter one exists.

▶ **Example 13.** Consider the equation E given by $xy\mathtt{a}y = \mathtt{a}z\mathtt{bb}xz$ over variables $x, y, z \in X$ and terminal symbols $\mathtt{a}, \mathtt{b} \in \Sigma$. Consider the solution $h : ((x, y, z) \cup \Sigma)^* \to \Sigma^*$ such that $h(x) = \mathtt{aaab}$, $h(y) = \mathtt{baa}$ and $h(z) = \mathtt{aa}$. Then the set of relative positions is:

$$\begin{aligned}\mathcal{RP}_E^h = \{&(x,1,1),(x,1,2),(x,1,3),(x,1,4),(x,2,1),(x,2,2),(x,2,3),(x,2,4),\\&(y,1,1),y(1,2),(y,1,3),(y,2,1),(y,2,2),(y,2,3),\\&(z,1,1),(z,1,2),(z,2,1),(z,2,2),(\mathtt{a},1,1),(\mathtt{a},2,1),(\mathtt{b},1,1),(\mathtt{b},2,1)\},\end{aligned}$$

the chains-representation of h consists of the four chains:

$$\begin{aligned}C_1 &= (\mathtt{a},1,1),(x,2,3),(z,1,2),(y,2,3),(x,2,2),(z,1,1),(y,2,2),(x,2,1),(\mathtt{a},2,1)\\C_2 &= (\mathtt{b},1,1),(x,1,4),(y,2,1),(\mathtt{b},2,1)\\C_3 &= (\mathtt{a},2,1),(x,1,1),(y,1,2),(z,2,1),(x,1,2),(y,1,3),(z,2,2),(x,1,3),(\mathtt{a},1,1)\\C_4 &= (\mathtt{b},2,1),(y,1,1),(x,2,4),(\mathtt{b},1,1)\end{aligned}$$

where C_1 and C_3 are dual, as are C_2 and C_4. The set of chain-words is given as follows, where for ease of reading, the elements $(x,1)$,$(x,2)$, $(y,1)$, $(y,2)$, $(z,1)$ and $(z,2)$ of Γ_E^h are denoted A, B, C, D, E and F respectively.

$$\Delta_{h,E} = \{BEDBEDB, AD, ACFACFA, CB\}.$$

The fact that one of the chain-words contains a square (e.g. $BEDBED$ in the first one) means that, due to Lemma 11, a shorter solution exists. In this case, we have the solution h' given by $h'(x) = \mathtt{aab}$, $h'(y) = \mathtt{ba}$ and $h'(z) = \mathtt{a}$.

Finally, we point out that this general approach of analysing the chain-words for squares can, at best, give exponential upper bounds on the length of minimal solutions to quadratic word equations, and additionally that the property of being chain-square-free, while necessary, is not sufficient for being minimal.

▶ **Proposition 14.** *Let $n \in \mathbb{N}$. Then the equation $x_n x_{n-1} \ldots x_1 \mathtt{a} = \mathtt{a} x_1 \ldots x_{n-1} x_n$ has a solution $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ which is not minimal but is chain-square-free such that $|h(x_n x_{n-1} \ldots x_1 \mathtt{a})| = 2^n$.*

4 Variable-Sparse and k-Ordered Quadratic Equations

We begin in the current section by considering quadratic equations without many repeating variables – those equations $\alpha = \beta$ for which the set $\{x \mid |\alpha\beta|_x = 2\}$ is “small”. Let the variable-sparse equations be defined as follows.

▶ **Definition 15.** *A word equation $\alpha = \beta$ is* variable-sparse *if $|\{x \mid |\alpha\beta|_x \geq 2\}| \leq \log(|\alpha\beta|)$.*

While there are practical reasons to care about variable-sparse equations – it seems reasonable to expect that equations encountered in practice may often have this form – we can also take advantage of the insights gained by considering this class to settle the complexity of another, more general class, namely the k-ordered equations, which complements, and thus motivates, the class of regular-reversed equations considered in the next section.

As the next proposition shows, the lengths of possible chains in the chains-representation of a minimal solution is bounded by a double-exponential function in number of repeating variables, and consequently so is the length of minimal solutions. The double-exponential bound is derived from the upper bound on lengths of minimal solutions to the whole class of word equations (see [22]), and thus we do not expect it to be tight. However, it is sufficient to show that in the case of variable-sparse quadratic equations, minimal solutions are at most single exponential in the length of the equation, and thus that the satisfiability problem for this class is contained in NP.

▶ **Proposition 16.** *Let $\alpha = \beta$ be a quadratic equation and let $V = |\{x \mid |\alpha\beta|_x = 2\}|$. If there exists a solution to $\alpha = \beta$, then there exists a solution $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ to $\alpha = \beta$ such that each chain in the chains representation of h w.r.t. $\alpha = \beta$ has length at most $2^{2^{O(V^4)}}$. Consequently, if h is minimal, then $|h(\alpha)| \leq |\alpha\beta|2^{2^{O(V^4)}}$. Moreover, it follows that the satisfiability problem for the class of variable-sparse quadratic equations is in NP.*

If we limit the number of variables further, bounded by a constant instead of $\log(|\alpha\beta|)$, then the bound on the length of minimal solutions becomes polynomial in the length of the equation, and we are able to infer the following.

▶ **Proposition 17.** *Let $V \in \mathbb{N}$ be a constant. Then the satisfiability problem for the class of quadratic equations with at most V variables can be solved in polynomial time.*

The k-ordered equations generalise the regular-ordered equations considered in [5], and essentially rely on the same idea of an order in which the variables must occur from left to right in both sides of the equation. However, this order is only enforced on small subsets of variables as a whole, and does not dictate the local order in which variables in a single subset may occur. For example, $x_1x_2x_3x_4x_5x_6 = x_3x_2x_1x_6x_4x_5$ is 3-ordered, but not 2-ordered.

▶ **Definition 18.** *Let $k \in \mathbb{N}$ and let $\alpha = \beta$ be a quadratic word equation. Then $\alpha = \beta$ is* k-ordered *if there exist pairwise disjoint sets of variables $X_1, X_2, \ldots, X_\ell$ with $|X_i| \leq k$ for $1 \leq i \leq \ell$, and $\alpha_i, \beta_i \in (X_i \cup \Sigma)^*$ for $1 \leq i \leq \ell$ such that $\alpha = \alpha_1\alpha_2 \ldots \alpha_\ell$ and $\beta = \beta_1\beta_2 \ldots \beta_\ell$.*

Length bounds for minimal solutions to k-ordered equations can be derived in a relatively straightforward manner from Proposition 17: after removing variables occurring only once (see Proposition 2), a basic length argument can be used to (non-deterministically) divide a k-ordered equation into linearly many individual equations over pairwise disjoint sets X_i of at most k variables X_i. Since these equations do not share any variables, solutions to the original equation can be obtained by simply combining solutions to the individual equations, which if they exist, due to Proposition 16, may be chosen to be short.

▶ **Proposition 19.** *Let $k \in \mathbb{N}$ be a constant and let $\alpha = \beta$ be a k-ordered quadratic word equation. Let $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ be a minimal solution to $\alpha = \beta$. Then $|h(\alpha)| \leq |\alpha\beta|2^{2^{O(k^4)}}$. Thus, the satisfiability problem for k-ordered quadratic equations is in NP.*

5 Regular-Reversed Quadratic Equations

Proposition 19 settles the complexity of the satisfiability problem for a large class of quadratic equations, in which the variables occur according to some global order on both sides of the equation. In the present section, we present our main result, concerning a class of equations which are, in a sense, opposite to k-ordered equations, namely the regular-reversed equations, in which the orders in which variables occur on each side are reversed. More formally, we define regular-reversed equations as follows:

▶ **Definition 20.** *Let $\pi : (X \cup \Sigma)^* \to X^*$ be the morphism such that $\pi(x) = x$ if $x \in X$ and $\pi(x) = \varepsilon$ otherwise. A quadratic word equation $\alpha = \beta$ is* regular-reversed *if $\pi(\alpha) = \pi(\beta)^R$.*

Since the equations described in Proposition 14 are in fact regular-reversed, we cannot hope to achieve polynomial bounds on the length of minimal solutions by relying on Lemma 11; the proposition tells us that in fact, exponentially long chain-square-free solutions exist. Nevertheless, we are able to exploit Lemma 11 to obtain exponential upper bounds, which due to Theorem 1, is sufficient to show that the satisfiability problem is NP for this class.

▶ **Theorem 21.** *Let $\alpha = \beta$ be a regular-reversed equation with n distinct variables. Let $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ be a chain-square-free solution to $\alpha = \beta$. Then $|h(\alpha)| \leq 2^{3n}|\alpha\beta|$. Consequently, the satisfiability of regular-reversed equations is in* NP*.*

The proof in full is rather technical and too long to include in the main exposition. However, in order to give a flavour of the reasoning, we include the following crucial lemma. Essentially, the lemma tells us that if a regular-reversed equation has a chain-square-free solution fulfilling certain combinatorial conditions, then we may erase a variable from both the equation and solution, to obtain a shorter equation over fewer variables and a new, shorter solution which is at least half as long as the original, but which remains chain-square-free. This reduction in the number of variables allows us to apply a straightforward induction, which, due to the linear reduction in the size of the solution in each step, yields exponential bounds on the length of chain-square-free (and hence also on minimal) solutions. The majority of the remaining effort in the proof is focused on reaching a point at which these combinatorial conditions are met. It is a straightforward observation that if the conditions are met, then the solution is not minimal, highlighting the usefulness of using chain-square-free solutions in place of minimal ones.

▶ **Lemma 22.** *Let $\alpha = \beta$ be a regular-reversed equation given by $u_0x_1u_1x_2 \dots x_nu_n = v_nx_nv_{n-1}x_{n-1} \dots v_1x_1v_0$ where $u_i, v_i \in \Sigma^*$ for $0 \leq i \leq n$, and $x_i \in X$ for $1 \leq i \leq n$. Suppose there exists a chain-square-free solution $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ such that for some $1 \leq p < q \leq n$:*

(1) *$h(u_0x_1 \dots u_{p-1}x_p) = h(v_nx_n \dots v_p)$ and $h(u_px_{p+1} \dots u_{n-1}x_nu_n) = h(x_pv_{p-1} \dots x_1v_0)$, and*

(2) *$|h(u_px_{p+1} \dots x_q)| < |h(x_p)|$ and $|h(x_qv_{q-1} \dots v_p)| \leq |h(x_p)|$, and*

(3) *$|h(u_px_{p+1} \dots x_qu_q)| + |h(v_qx_qv_{q-1} \dots v_p)| \geq |h(x_p)|$.*

Then the substitution $g : ((\mathrm{var}(\alpha\beta)\backslash\{x_p\}) \cup \Sigma)^* \to \Sigma^*$ given by $g(x) = h(x)$ for all $x \in \mathrm{var}(\alpha\beta)\backslash\{x_p\}$ is a chain-square-free solution to the equation $\alpha' = \beta'$ obtained by erasing x_p from α and β, and moreover $|g(\alpha')| \geq \frac{|h(\alpha)|}{2}$.

Proof. To see that g is a solution to $\alpha' = \beta'$, let $w = h(u_0x_1 \dots u_{p-1})$ and let $w' = h(v_{p-1} \dots x_1v_0)$. Note that $g(v_nx_n \dots v_p) = wh(x_p)$ and $g(u_px_{p+1} \dots u_{n-1}x_nu_n) = h(x_p)w'$. It follows that $g(\alpha') = wh(x_p)w' = g(\beta')$, while $h(\alpha) = wh(x_p)h(x_p)w' = h(\beta)$. It follows immediately that $|h(x_p)| \leq \frac{|h(\alpha)|}{2}$, and thus that $|g(\alpha')| = |h(\alpha)| - |h(x_p)| \geq \frac{|h(\alpha)|}{2}$. It remains to see that g is chain-square-free, for which we must understand the chains representation of g compared to that of h. In particular, we shall show (via Claims 23 and 24) that the chain(-words) of g are obtained by simply removing positions belonging to the variable x_p from the chain(-words) of h. Before we prove this statement, we observe some basic facts about the sibling and neighbour relations for each solution. Note firstly that $\mathcal{RP}^g_{\alpha'=\beta'} = \mathcal{RP}^h_{\alpha=\beta}\backslash\{(x_p, i, d) \mid i \in \{1,2\}, d \in [1..|h(x_p)|]\}$. Moreover, note that two positions $r_1, r_2 \in \mathcal{RP}^g_{\alpha'=\beta'}$ are siblings in the chains representation of g if and only if they are siblings in the chains representation of h. Finally, for each $r \in \mathcal{RP}^g_{\alpha'=\beta'}$, we have $f^g_{\alpha'=\beta'}(r) = f^h_{\alpha=\beta}(r) - \mu$ where $\mu = 0$ if $f^h_{\alpha=\beta}(r) \leq |wh(x_p)|$ and $\mu = |h(x_p)|$ otherwise. Consequently, for any position $r \in \mathcal{RP}^g_{\alpha'=\beta'}$ such that $f^h_{\alpha=\beta}(r) \leq |w|$ or $f^h_{\alpha=\beta}(r) > |h(\alpha)| - |w'|$, the neighbour of r in the chains representation of g is the same as the neighbour of r in the chains representation of h.

The following two claims describe the successor relation (and hence the chains) for g w.r.t. $\alpha' = \beta'$ in terms of the successor relation for h w.r.t. $\alpha = \beta$.

▷ Claim 23. Let r, r' be a subchain of some chain in the chains representation of h w.r.t. $\alpha = \beta$. Suppose that neither r nor r' belongs to x_p. Then r, r' is also a subchain of some chain in the chains representation of g w.r.t. $\alpha' = \beta'$.

Proof (Claim 23). Let $\overline{r} = r$ if r belongs to a terminal symbol, and let $\overline{r}$ be the sibling of r otherwise. Note that since r does not belong to x_p, the definition of $\overline{r}$ is the same for both chains-representations. Moreover, note that since no variables occur only once in the equations $\alpha = \beta$ and $\alpha' = \beta'$, the successor of r is the neighbour of $\overline{r}$ in both chains representations. If $f^h_{\alpha=\beta}(\overline{r}) \leq |w|$ or $f^h_{\alpha=\beta}(\overline{r}) > |h(\alpha)| - |w'|$, then as previously mentioned, the neighbour of $\overline{r}$, and hence successor of r, is the same in both chains representations and the claim follows. If instead $|w| < f^h_{\alpha=\beta}(\overline{r}) \leq |h(\alpha)| - |w'|$, then either $\overline{r}$ belongs to x_p, or the neighbour of $\overline{r}$ in the chains representation of h w.r.t $\alpha = \beta$ belongs to x_p. If $\overline{r}$ belongs to x_p, then r belongs to x_p which is a contradiction. Similarly, since the neighbour of $\overline{r}$ is the successor of r, namely r', if it belongs to x_p, then we again get a contradiction, so neither case is possible under the assumptions of the claim. ◁

▷ Claim 24. Let r, r', r'' be a subchain of some chain in the chains representation of h w.r.t. $\alpha = \beta$ such that r' belongs to x_p. Then neither r nor r' belongs to x_p, and moreover, r, r'' is also a subchain of some chain in the chains representation of g w.r.t. $\alpha' = \beta'$.

Proof (Claim 24). The fact that r and r'' do not belong to x_p follows immediately from the fact that h is chain-square-free along with Remark 9. As before, let $\overline{r} = r$ if r belongs to a terminal symbol and let $\overline{r}$ be the sibling of r otherwise. Again, note that since r does not belong to x_p, the definition of $\overline{r}$ is the same for both chains-representations. Let $\overline{r'}$ be the sibling of r' in the chains representation of h w.r.t $\alpha = \beta$ (recall that x_p occurs twice in $\alpha = \beta$, so the sibling exists). Then r' is the neighbour of $\overline{r}$ and r'' is the neighbour of $\overline{r'}$ in the chains representation of h w.r.t $\alpha = \beta$. We shall proceed by distinguishing two cases based on r'. Suppose firstly that $r' = (x_p, 2, d)$ for some $d \in [1..|h(x_p)|]$ (so r' belongs to the occurrence of x_p on the RHS). Then $|wh(x_p)| < f^h_{\alpha=\beta}(r') = f^h_{\alpha=\beta}(\overline{r}) \leq |h(\alpha)| - |w'|$ and $f^h_{\alpha=\beta}(\overline{r'}) + |h(x_p)| = f^h_{\alpha=\beta}(r')$. Hence $f^g_{\alpha'=\beta'}(\overline{r}) = f^h_{\alpha=\beta}(\overline{r}) - |h(x_p)|$, and

$$f^h_{\alpha=\beta}(r'') = f^h_{\alpha=\beta}(\overline{r'}) = f^h_{\alpha=\beta}(r') - |h(x_p)| = f^h_{\alpha=\beta}(\overline{r}) - |h(x_p)| \leq |h(\alpha)| - |w'| - |h(x_p)|.$$

Since $|h(\alpha)| - |w'| - |h(x_p)| = |wh(x_p)|$, this implies that $f^g_{\alpha'=\beta'}(r'') = f^h_{\alpha=\beta}(r'')$. Thus:

$$f^g_{\alpha'=\beta'}(\overline{r}) = f^h_{\alpha=\beta}(\overline{r}) - |h(x_p)| = f^h_{\alpha=\beta}(r') - |h(x_p)| = f^h_{\alpha=\beta}(\overline{r'}) = f^h_{\alpha=\beta}(r'') = f^g_{\alpha'=\beta'}(r'').$$

Symmetrically, if instead $r' = (x_p, 1, d)$ for some $d \in [1..|h(x_p)|]$ (so r' belongs to the occurrence of x_p on the LHS), then in the same manner, we can derive:

$$f^g_{\alpha'=\beta'}(\overline{r}) = f^h_{\alpha=\beta}(\overline{r}) = f^h_{\alpha=\beta}(r') = f^h_{\alpha=\beta}(\overline{r'}) - |h(x_p)| = f^h_{\alpha=\beta}(r'') - |h(x_p)| = f^g_{\alpha'=\beta'}(r'').$$

In both cases we get that $f^g_{\alpha'=\beta'}(\overline{r}) = f^g_{\alpha'=\beta'}(r'')$, so r'' and $\overline{r}$ are neighbours in the chains representation of g w.r.t $\alpha' = \beta'$ and hence r'' is the successor of r in the chains representation of g w.r.t $\alpha' = \beta'$ and the statement of the claim follows. ◁

It follows easily from Claims 23 and 24 that each chain in the chains representation of g is obtained by removing all positions belonging to x_p from some chain in the chains-representation of h. It remains to check that the act of removing the positions belonging to x_p does not introduce any squares in the resulting chain-words.

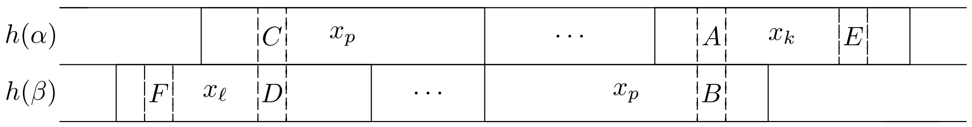


Figure 2 By Condition (1), the two occurrences of x_p are adjacent. Consequently, if (x_k, i', d_1) and (x_p, i, d_2) are neighbours, and (x_ℓ, i, d_3) and (x_p, i', d_2) are neighbours, then it cannot be the case that (x_k, i', d_4) and (x_ℓ, i, d_5) are neighbours. The case $i = 2$ is shown. The case $i = 1$ is symmetric and may be obtained by swapping the locations of x_k and x_ℓ. The positions (x_k, i', d_1), (x_p, i, d_2), (x_p, i', d_2), (x_ℓ, i, d_3), (x_k, i', d_4) and (x_ℓ, i, d_5) are marked as A, B, C, D, E and F respectively.

Suppose for contradiction that a new square is introduced. That is, there exists a subchain C in the chains representation of h for which the induced chain-word is not a square, but for which the chain-word induced by the new subchain $\hat{C}$ obtained by removing all positions belonging to x_p from C is a square. There are two ways in which this may happen. The first is that the original subchain C has the form $C', (x_p, i, d_1), C''$ where C' and C'' are subchains of length at least one and are similar (thus inducing a square in the chain-word once the central (x_p, i, d) is removed). Note that in this case there might be further positions belonging to x_p in C' and C'', but the act of removing them will not alter their similarity so we do not need to keep track of them explicitly.

By Condition (3), every position belonging to x_p is either the successor, or predecessor of a position which is either terminal or belongs to a variable x_j with $p < j \leq q$. Since the subchains C' and C'' do not contain terminal positions (this would contradict the assumption that they are similar), they must both either start with a position belonging to some x_j with $p < j \leq q$, or both end with a position to belonging to some x_j with $p < j \leq q$. It follows from Condition (2) that the neighbour of any position belonging to x_j must belong to x_p. Thus the successor and predecessor of a position belonging to x_j must also belong to x_p. However, this implies that the original subchain C occurs directly before, or after a position (x_p, i, d_2) (by Remark 9, it will have the same index i as the position (x_p, i, d_1)). However, this results in a subchain of the form $(x_p, i, d_2), C', (x_p, i, d_1), C''$ or of the form $C', (x_p, i, d_1), C'', (x_p, i, d_2)$, which in either case induces chain-word containing a square, a contradiction to the fact that h is chain-square-free.

The second possibility is that C may be divided into two further subchains C' and C'' (so $C = C', C''$), where C' and C'' are not similar, but become similar once positions belonging to x_p are removed. This implies the existence of (not necessarily consecutive) subchains $(x_k, i, d_1), (x_p, i, d_2), (x_\ell, i, d_3)$ and $(x_k, i, d_4), (x_\ell, i, d_5)$ of C (one occurring in C' and the other in C''). Clearly, since h is chain-square-free, we must have that $k \neq \ell \neq p$. Let $i' = i + 1 \mod 2$. Then (x_k, i', d_1) and (x_p, i, d_2) are neighbours, (x_p, i', d_2) and (x_ℓ, i, d_3) are neighbours and (x_k, i', d_4) and (x_ℓ, i, d_5) are neighbours. However this is only possible if the two occurrences of x_p in the solution $h(x)$ are not adjacent (see Fig. 2), or more precisely, it implies that $f^h_{\alpha=\beta}((x_p, 1, |h(x_p)|)) < f^h_{\alpha=\beta}((x_p, 2, 1)) - 1$. This clearly contradicts Condition (1). In all cases we get a contradiction, so g must be chain-square-free as claimed. ◀

6 Avoiding Squares and other Patterns

Lemma 11 invites an obvious question: do there exist long solutions for which the chain-words (which must then also be long) are square-free? This question is particularly interesting as a negative answer for the appropriate meaning of long would be sufficient to show that the satisfiability of quadratic word equations is in NP. For regular equations with two variables, the chain-words will be over an alphabet of size two, meaning we get an immediate answer: a quick exhaustive search reveals that any word over two letters of length at least 4 contains a square. Thus, for a regular equation with two variables, the chain-words of a minimal solution can have length at most 3, and thus, any minimal solution must have length at most $3n$ where n is the number of terminal symbols in the equation.

Unfortunately, a famous result of Thue [26] reveals that there exist infinitely long words over three letters which do not contain squares, meaning such a simple proof will not work for equations with more variables. This does not mean, however, that Lemma 11 is of no use in more the more general case. It is easily established that not all words may occur as chain-words of some solution to an equation (note, e.g. that the number of possible different factors of length two in a chain-word is $2n - 1$ where n is the number of variables, while in general there are n^2 such factors). The next theorem gives a characterisation of when a word w is a chain-word of some solution to a regular word equation.

▶ **Theorem 25.** *Let w be a word and let Γ be the alphabet of letters occurring in w. There exists a regular word equation E with solution h such that w is a chain-word in $\Delta_{h,E}$ if and only if, there exist letters $\$, \# \notin \Gamma$ and linear orders $<_1$, $<_2$ on the sets $\Gamma \cup \{\#\}$ and $\Gamma \cup \{\$\}$ respectively such that for every $u \in \Gamma^*$ and $A, B, C, D \in \Gamma \cup \{\$, \#\}$ with $A \neq B$ and $C \neq D$, if AuC and BuD are both factors of $\#w\$$, then either that $A <_2 B$ and $C <_1 D$ or that $B <_2 A$ and $D <_1 C$.*

▶ **Corollary 26.** *Let E be a regular word equation and let h be a solution to E. Let $w \in \Delta_{h,E}$. Let A, B, C, D be letters from w such that $A \neq B$ and $C \neq D$ Then for any word u, at least one of AuC, BuC, AuD, BuD is not a factor of w.*

While this characterisation appears not to reveal immediately whether "long" square-free chain-words exist, we can make use of it to derive further conditions which may be more useful. As an example. we show in the following lemma that chain-words avoiding squares must also avoid other types of pattern (which are not necessarily avoidable in general).

▶ **Lemma 27.** *Let E be a quadratic word equation given by $\alpha = \beta$ and let $h : (\mathrm{var}(\alpha\beta) \cup \Sigma)^* \to \Sigma^*$ be a solution to E. If there exists a chain-word $w \in \Delta_{E,h}$ which contains factor of the form $x_1x_2x_3x_4x_2x_1x_3$ such that x_3 is not a prefix of x_1 or x_2, then there exists a (possibly distinct from w) chain-word $w' \in \Delta_{E,h}$ which contains a square.*

Unlike squares, it follows from the famous Zimin algorithm [18] that all words which are "long enough" will encounter a factor of the form $x_1x_2x_3x_4x_2x_1x_3$. In other words, $x_1x_2x_3x_4x_2x_1x_3$ is an *unavoidable pattern*. Unfortunately, however, this does not guarantee the additional condition that x_3 is not a prefix of x_1 or x_2, so Lemma 27 does not immediately provide a bound on the length of chain-square-free solutions. Nevertheless, a quick exhaustive search again reveals that any word of length at least 8 over three letters contains such a factor of the form $x_1x_2x_3x_4x_2x_1x_3$ or its reversal where x_1, x_2 and x_3 are all distinct letters (and so satisfying the prefix/suffix conditions given in Lemma 27). Moreover, it is not difficult to adapt the proof of Lemma 27 to produce other "forbidden factors" which must be avoided in chain-words of chain-square-free solutions. Thus we expect it to be a promising direction to try to obtain upper bounds on the lengths of (subclasses of) quadratic word equations through the lens of unavoidable patterns: do there exist combinations of patterns which are unavoidable, at least when considered over some over-approximation of all possible chain-words which ultimately guarantee the existence of squares in the chain-words of the same solutions?

References

1 P. A. Abdulla, M. F. Atig, Y. Chen, L. Holík, A. Rezine, P. Rümmer, and J. Stenman. Norn: An SMT Solver for String Constraints. In *Proc. CAV 2015*, volume 9206 of *LNCS*, pages 462–469, 2015.

2 A. Aydin, L. Bang, and T. Bultan. Automata-Based Model Counting for String Constraints. In *Proc. CAV 2015*, volume 9206 of *LNCS*, pages 255–272, 2015.

3 C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proc. CAV 2011*, volume 6806 of *LNCS*, pages 171–177, 2011.

4 M. Berzish, V. Ganesh, and Y. Zheng. Z3str3: A string solver with theory-aware heuristics. In *Proc. FMCAD 2017*, pages 55–59. IEEE, 2017.

5 J. D. Day, F. Manea, and D. Nowotka. The Hardness of Solving Simple Word Equations. In *Proc. MFCS 2017*, volume 83 of *LIPIcs*, pages 18:1–18:14, 2017.

6 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.

7 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.

8 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.

9 A. Ehrenfeucht and G. Rozenberg. Finding a Homomorphism Between Two Words is NP-Complete. *Information Processing Letters*, 9:86–88, 1979.

10 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.

11 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.

12 J. Jaffar. Minimal and Complete Word Unification. *Journal of the ACM*, 37(1):47–85, 1990.

13 A. Jeż. Recompression: a simple and powerful technique for word equations. In *Proc. STACS 2013*, volume 20 of *LIPIcs*, pages 233–244, 2013.

14 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.

15 A. Jeż. Word Equations in Nondeterministic Linear Space. In *Proc. ICALP 2017*, volume 80 of *LIPIcs*, pages 95:1–95:13, 2017.

16 J. Karhumäki, F. Mignosi, and W. Plandowski. The expressibility of languages and relations by word equations. *Journal of the ACM (JACM)*, 47(3):483–505, 2000.

17 A. Kiezun, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst. HAMPI: a solver for string constraints. In *Proc. ISSTA 2009*, pages 105–116. ACM, 2009.

18 M. Lothaire. *Combinatorics on Words.* Addison-Wesley, 1983.

19 R. C. Lyndon. Equations in free groups. *Transactions of the American Mathematical Society*, 96:445–457, 1960.

20 R. C. Lyndon and P. E. Schupp. *Combinatorial Group Theory.* Springer, 1977.

21 G. S. Makanin. The problem of solvability of equations in a free semigroup. *Sbornik: Mathematics*, 32(2):129–198, 1977.

22 W. Plandowski. Satisfiability of Word Equations with Constants is in NEXPTIME. In *Proc. STOC 1999*, pages 721–725. ACM, 1999.

23 W. Plandowski. Satisfiability of word equations with constants is in PSPACE. In *Proc. FOCS 1999*, pages 495–500. IEEE, 1999.

24 W. Plandowski and W. Rytter. Application of Lempel-Ziv Encodings to the Solution of Words Equations. In *Proc. ICALP 1998*, volume 1443 of *LNCS*, pages 731–742, 1998.

25 K. U. Schulz. Word Unification and Transformation of Generalized Equations. *Journal of Automated Reasoning*, 11:149–184, 1995.

26 A. Thue. Über unendliche Zeichenreihen. *Kra. Vidensk. Selsk. Skrifter. I Mat. Nat. Kl.*, 7, 1906.

27 M. Trinh, D. Chu, and J. Jaffar. Progressive Reasoning over Recursively-Defined Strings. In *Proc. CAV 2016*, volume 9779 of *LNCS*, pages 218–240, 2016.

28 F. Yu, M. Alkhalaf, and T. Bultan. STRANGER: An Automata-based String Analysis Tool for PHP. In *Proc. TACAS 2010*, volume 6015 of *LNCS*, 2010.

The Power of the Weisfeiler-Leman Algorithm to Decompose Graphs

Sandra Kiefer
RWTH Aachen University, Aachen, Germany
kiefer@cs.rwth-aachen.de

Daniel Neuen
RWTH Aachen University, Aachen, Germany
neuen@cs.rwth-aachen.de

Abstract

The Weisfeiler-Leman procedure is a widely-used approach for graph isomorphism testing that works by iteratively computing an isomorphism-invariant coloring of vertex tuples. Meanwhile, a fundamental tool in structural graph theory, which is often exploited in approaches to tackle the graph isomorphism problem, is the decomposition into bi- and triconnected components.

We prove that the 2-dimensional Weisfeiler-Leman algorithm implicitly computes the decomposition of a graph into its triconnected components. Thus, the dimension of the algorithm needed to distinguish two given graphs is at most the dimension required to distinguish the corresponding decompositions into 3-connected components (assuming dimension at least 2).

Our result implies that for $k \geq 2$, the k-dimensional algorithm distinguishes k-separators, i.e., k-tuples of vertices that separate the graph, from other vertex k-tuples. As a byproduct, we also obtain insights about the connectivity of constituent graphs of association schemes.

In an application of the results, we show the new upper bound of k on the Weisfeiler-Leman dimension of graphs of treewidth at most k. Using a construction by Cai, Fürer, and Immerman, we also provide a new lower bound that is asymptotically tight up to a factor of 2.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Theory of computation → Finite Model Theory; Theory of computation → Graph algorithms analysis

Keywords and phrases Weisfeiler-Leman, separators, first-order logic, counting quantifiers

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.45

Related Version http://arxiv.org/abs/1908.05268

1 Introduction

Originally introduced in [37], the Weisfeiler-Leman (WL) algorithm has become a – if not the – fundamental subroutine in the context of isomorphism testing for graphs. It is used in theoretical as well as in practical approaches to tackle the graph isomorphism problem (see e.g. [5, 19, 31, 32, 35]), among them also Babai's recent quasipolynomial-time isomorphism test [4]. For every $k \geq 1$, there is a k-dimensional version of the algorithm which colors the vertex k-tuples of the input graph and iteratively refines the coloring in an isomorphism-invariant manner.

There are various characterizations of the algorithm, which link it to other areas in theoretical computer science (see also *Further Related Work*). For example, very recent results in the context of machine learning show that the 1-dimensional version of the algorithm is as expressive as graph neural networks with respect to distinguishing graphs [34]. Following Grohe [16], an indicator to investigate the expressive power of the algorithm is the so-called *WL dimension* of a graph, defined as the minimal dimension of the WL algorithm required in order to distinguish the graph from every other non-isomorphic graph.

There is no fixed dimension of the algorithm that decides graph isomorphism in general, as was proved by Cai, Fürer, and Immerman [9]. Still, when focusing on particular graph classes, often a bounded dimension of the algorithm suffices to identify every graph in the

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 45; pp. 45:1–45:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

class. This proves that for the considered class, graph isomorphism is solvable in polynomial time, since the k-dimensional algorithm can be implemented in time $\mathcal{O}(n^{k+1}\log n)$ [27]. For example, it suffices to apply the 3-dimensional WL algorithm to identify every planar graph [29]. Also, the WL dimension of graphs of treewidth at most k is bounded by $k+2$ [18]. More generally, by a celebrated result by Grohe, for all graph classes with an excluded minor, the WL dimension is bounded [15]. Very recent work provides explicit upper bounds on the WL dimension, which are linear in the rank width [19] and in the Euler genus [17], respectively, of the graph.

Regarding combinatorial techniques, to handle graphs with complex structures, the decomposition into connected, biconnected, and triconnected components provides a fundamental tool from structural graph theory. The decomposition can be computed in linear time (see e.g. [25, 36]). Hopcroft and Tarjan used the decomposition of a graph into its triconnected components to obtain an algorithm that decides isomorphism for planar graphs in quasi-linear time [22, 23, 24], which was improved to linear time by Hopcroft and Wong [26].

Also, in [29], to prove the bound on the WL dimension for the class of planar graphs, the challenge of distinguishing two arbitrary planar graphs is reduced to the case of two arc-colored triconnected planar graphs, by exploiting the fact that the 3-dimensional WL algorithm is able to implicitly compute the decomposition of a graph into its triconnected components. Similarly, the bound on the WL dimension for graphs parameterized by their Euler genus from [17] relies on an isomorphism-invariant decomposition of the graphs into their triconnected components.

Our Contribution. We show that for $k \geq 2$, the k-dimensional WL algorithm implicitly computes the decomposition into the triconnected components of a given graph. More specifically, we prove that already the 2-dimensional WL algorithm distinguishes separating pairs, i.e., pairs of vertices that separate the given graph, from other vertex pairs. This improves on a result from [29], where an analogous statement was proved for the 3-dimensional WL algorithm. Using the decomposition techniques discussed there, we conclude that for the k-dimensional WL algorithm with $k \geq 2$, to identify a graph, it suffices to determine vertex orbits on all arc-colored 3-connected components of it. Since it is easy to see that $k = 1$ does not suffice to distinguish vertices contained in 2-separators from others, our upper bound of 2 is tight.

The expressive power of the k-dimensional algorithm corresponds to definability in the logic C^{k+1}, the extension of the $(k+1)$-variable fragment of first-order logic by counting quantifiers [9, 27]. Exploiting this correspondence, our results imply that for every $n \in \mathbb{N}$, there is a formula $\varphi_n(x_1, x_2) \in \mathsf{C}^3$ (first-order logic with counting quantifiers over three variables) such that for an n-vertex graph G, it holds that $G \models \varphi_n(v, w)$ if and only if $\{v, w\}$ is a 2-separator in G. With only three variables at our disposal, it is not possible to take the route of [29] by comparing certain numbers of walks between different pairs of vertices. Instead, the formulas obtained from our proof are essentially a disjunction over all n-vertex graphs and subformulas for two distinct graphs may look completely different, exploiting specific structural properties of the graphs. While this makes the proof rather involved, it also stresses the power of the 2-dimensional WL algorithm and equivalently, the expressive power of the logic C^3. We show that for all $n, s \in \mathbb{N}$, there is a formula $\varphi_{n,s}(x_1, x_2, x_3) \in \mathsf{C}^3$ such that for an n-vertex graph G, it holds that $G \models \varphi_{n,s}(u, v, w)$ if and only if $s = |C|$, where C is the vertex set of the connected component containing u after removing v and w from the graph G.

Our result can also be viewed in a combinatorial setting. In 1985, Brouwer and Mesner [8] proved that the vertex connectivity of a strongly regular graph equals its degree and that in fact, the only minimal disconnecting vertex sets are neighborhoods. Later, Brouwer conjectured this to be true for any constituent graph of an association scheme (i.e., any graph consisting in a single color class of the association scheme) [6]. While some progress has been made on certain special cases [13], most prominently distance-regular graphs [7], the general question is still open. Our results imply that any connected constituent graph of an association scheme is either a cycle or 3-connected. Such a statement was previously only known for symmetric association schemes [30], which are far more restricted than the general ones.

A natural use case of these results is to determine or to improve upper bounds on the WL dimension of certain graph classes. As a first application in this direction, we obtain a new upper bound of k on the WL dimension for graphs of treewidth at most k. Based on [10], we also provide a new lower bound for this graph class, thus delimiting the value of the WL dimension of graphs of treewidth bounded by k to the interval $\left[\lceil\frac{k}{2}\rceil - 3, k\right]$.

Due to space restrictions, some of the proofs and proof details have been omitted. For a full version of this paper, we refer the reader to [28].

Further Related Work. Apart from its correspondence to counting logics, the WL algorithm has further surprising links to other areas. For example, the algorithm has a close connection to Sherali-Adams relaxations of particular linear programs [3, 20] and captures the same information as certain homomorphism counts [11]. It can also be characterized via winning strategies in so-called pebble games [21], which are a particular family of Ehrenfeucht-Fraissë games.

As mentioned above, the 1-dimensional WL algorithm essentially corresponds to graph neural networks. In order to make them more powerful, the authors of [34] propose an extension of graph neural networks based on the k-dimensional WL algorithm (see also [33]).

Towards understanding the expressive power of the algorithm, in a related direction of research, it has been studied which graph properties the WL algorithm can detect, which may become particularly relevant in the graph-learning framework. In this context, Fürer [14] as well as Arvind et al. [2] obtained results concerning the ability of the algorithm to detect and count certain subgraphs.

2 Preliminaries

2.1 Graphs

A *graph* is a pair $G = \big(V(G), E(G)\big)$ of a *vertex set* $V(G)$ and an *edge set* $E(G) \subseteq \big\{\{u, v\} \mid u, v \in V(G)\big\}$. To give explicit reference to G, we also write $V(G)$ for V and $E(G)$ for E. All graphs considered in this paper are finite, simple (i.e., they contain no loops or multiple edges), and undirected. For $v, w \in V$, we also write vw as a shorthand for $\{v, w\}$. The *neighborhood* of v is denoted by $N(v)$, and the *closed neighborhood* of v is $N[v] \coloneqq N(v) \cup \{v\}$. The *degree* of v, denoted by $\deg(v)$, is the number of edges incident with v. For $X \subseteq V(G)$ we define $N(X) \coloneqq \big(\bigcup_{v \in X} N(v)\big) \setminus X$.

A *walk of length* k from v to w is a sequence of vertices $v = u_0, u_1, \dots, u_k = w$ such that $u_{i-1}u_i \in E$ for all $i \in \{1, \dots, k\}$. A *path of length* k from v to w is a walk of length k from v to w for which all occurring vertices are pairwise distinct. We refer to the *distance* between two vertices $v, w \in V(G)$ by $\operatorname{dist}(v, w)$. For a set $A \subseteq V(G)$, we denote by $G[A]$ the *induced subgraph* of G on vertex set A. Also, we denote by $G - A$ the subgraph induced by the complement of A, that is, the graph $G - A := G[V(G) \setminus A]$. A set $S \subseteq V(G)$ is a *separator of* G if $G - S$ has more connected components than G. A *k-separator* of G is a separator of G of size k. A vertex $v \in V(G)$ is a *cut vertex* if $\{v\}$ is a separator of G. The graph G is *k-connected* if it is connected and has no $(k-1)$-separator.

An *isomorphism* from G to another graph H is a bijection $\varphi\colon V(G) \to V(H)$ that respects the edge relation, that is, for all $v, w \in V(G)$, it holds that $vw \in E(G)$ if and only if $\varphi(v)\varphi(w) \in E(H)$. Two graphs G and H are *isomorphic* ($G \cong H$) if there is an isomorphism from G to H. We write $\varphi\colon G \cong H$ to denote that φ is an isomorphism from G to H.

A *vertex-colored graph* is a tuple (G, χ), where G is a graph and $\chi\colon V(G) \to \mathcal{C}$ is a mapping into some set $\mathcal{C}$ of colors. Similarly, an *arc-colored graph* is a tuple (G, χ), where G is a graph and $\chi\colon \{(v,v) \mid v \in V(G)\} \cup \{(u,v) \mid \{u,v\} \in E(G)\} \to \mathcal{C}$ is a mapping into some color set $\mathcal{C}$. Typically, $\mathcal{C}$ is chosen to be an initial segment $[n] := \{1, \dots, n\}$ of the natural numbers. Isomorphisms between vertex- and arc-colored graphs have to respect the colors of the vertices and arcs.

For details on the *treewidth* of a graph, we refer the reader to [12].

2.2 The Weisfeiler-Leman Algorithm

Let $\chi_1, \chi_2\colon V^k \to \mathcal{C}$ be colorings of the k-tuples of vertices of G, where $\mathcal{C}$ is some finite set of colors. We say χ_2 *refines* χ_1 if for all $\bar{v}, \bar{w} \in V^k$ we have $\big(\chi_2(\bar{v}) = \chi_2(\bar{w}) \;\Rightarrow\; \chi_1(\bar{v}) = \chi_1(\bar{w})\big)$. The k-dimensional WL algorithm is a procedure that, given a graph G and a coloring χ of its k-tuples of vertices, computes an isomorphism-invariant refinement of the coloring.

We describe the mechanisms of the algorithm in the following. For an integer $k > 1$ and a vertex-colored graph (G, χ), we let $\chi^0_{G,k}\colon V^k \to \mathcal{C}$ be the coloring where each k-tuple is colored with the isomorphism type of its underlying ordered colored subgraph. More formally, $\chi^0_{G,k}(v_1, \dots, v_k) = \chi^0_{G,k}(w_1, \dots, w_k)$ if and only if for all $i \in [k]$ it holds that $\chi(v_i) = \chi(w_i)$ and for all $i, j \in [k]$, it holds that $v_i = v_j \Leftrightarrow w_i = w_j$ and $v_iv_j \in E(G) \Leftrightarrow w_iw_j \in E(G)$. If G is arc-colored, the arc colors must be respected accordingly.

We then recursively define the coloring $\chi^i_{G,k}$ obtained after i rounds of the algorithm. Let $\chi^{i+1}_{G,k}(v_1, \dots, v_k) := (\chi^i_{G,k}(v_1, \dots, v_k); \mathcal{M})$, where $\mathcal{M}$ is a multiset defined as

$$\{\!\!\{ \big(\chi^i_{G,k}(\bar{v}[w/1]), \chi^i_{G,k}(\bar{v}[w/2]), \dots, \chi^i_{G,k}(\bar{v}[w/k])\big) \,\big|\, w \in V \}\!\!\}$$

where $\bar{v}[w/i] := (v_1, \dots, v_{i-1}, w, v_{i+1}, \dots, v_k)$.

For the 1-dimensional algorithm (i.e. $k = 1$), the definition is similar, but we iterate only over the neighbors of v_1, that is, the multiset $\mathcal{M}$ equals $\{\!\!\{\chi^i_{G,1}(w) \mid w \in N(v_1)\}\!\!\}$.

By definition, every coloring $\chi^{i+1}_{G,k}$ induces a refinement of the partition of the k-tuples of vertices of the graph G with coloring $\chi^i_{G,k}$. Thus, there is a minimal i such that the partition of the vertex k-tuples induced by $\chi^{i+1}_{G,k}$ is not strictly finer than the one induced by $\chi^i_{G,k}$. For this value of i, we call the coloring $\chi^i_{G,k}$ the *stable* coloring of G and denote it by $\chi_{G,k}$.

The original WL algorithm is its 2-dimensional variant [37]. Since that version is the central algorithm of this paper, we omit the index 2 and write χ_G instead of $\chi_{G,2}$.

For $k \in \mathbb{N}$, the *k-dimensional WL algorithm* takes as input a (vertex- or arc-)colored graph (G, χ) and returns the coloring $\chi_{G,k}$. The procedure can be implemented in time $O(n^{k+1} \log n)$ [27]. For two graphs G and H, we say that the k-dimensional WL algorithm *distinguishes* G and H if there is a color c such that the sets $\{\bar{v} \mid \bar{v} \in \big(V(G)\big)^k, \chi_{G,k}(\bar{v}) = c\}$ and $\{\bar{w} \mid \bar{w} \in \big(V(H)\big)^k, \chi_{H,k}(\bar{w}) = c\}$ have different cardinalities. We write $G \simeq_k H$ if the k-dimensional WL algorithm does not distinguish G and H. The algorithm *identifies* G if it distinguishes G from every non-isomorphic graph H.

Pebble Games. For further analysis, it is often cumbersome to work with the WL algorithm directly and more convenient to use the following characterization via pebble games, which is known to capture the same information. Let $k \in \mathbb{N}$. For graphs G and H on the same number of vertices and with vertex colorings χ and χ', respectively, we define the *bijective k-pebble game* $\mathrm{BP}_k(G,H)$ as follows:

- The game has two players called *Spoiler* and *Duplicator.*
- The game proceeds in rounds, each of which is associated with a pair of positions $(\bar{v},\bar{w})$ with $\bar{v} \in \big(V(G)\big)^\ell$ and $\bar{w} \in \big(V(H)\big)^\ell$, where $0 \leq \ell \leq k$.
- The initial position of the game is a pair of vertex tuples of equal length ℓ with $0 \leq \ell \leq k$. If not specified otherwise, the initial position is the pair $\big((),()\big)$ of empty tuples.
- Each round consists of the following steps. Suppose the current position of the game is $(\bar{v},\bar{w}) = ((v_1,\dots,v_\ell),(w_1,\dots,w_\ell))$. First, Spoiler chooses whether to remove a pair of pebbles or to play a new pair of pebbles. The first option is only possible if $\ell > 0$, and the latter option is only possible if $\ell < k$.

  If Spoiler wishes to remove a pair of pebbles, he picks some $i \in [\ell]$ and the game moves to position $(\bar{v} \setminus i, \bar{w} \setminus i)$ where $\bar{v} \setminus i := (v_1,\dots,v_{i-1},v_{i+1},\dots,v_\ell)$, and the tuple $(\bar{w} \setminus i)$ is defined in the analogous way. Otherwise, the following steps are performed.

  (D) Duplicator picks a bijection $f\colon V(G) \to V(H)$.

  (S) Spoiler chooses $v \in V(G)$. The new position is then $\big((v_1,\dots,v_\ell,v),(w_1,\dots, w_\ell,f(v))\big)$.

  If for the current position $\big((v_1,\dots,v_\ell),(w_1,\dots,w_\ell)\big)$, the induced ordered subgraphs of G and H are not isomorphic, Spoiler wins the play. More precisely, Spoiler wins if there is an $i \in [\ell]$ such that $\chi(v_i) \neq \chi'(w_i)$, or there are $i,j \in [\ell]$ such that $v_i = v_j \not\Leftrightarrow w_i = w_j$ or $v_iv_j \in E(G) \not\Leftrightarrow w_iw_j \in E(H)$. If there is no position of the play such that Spoiler wins, then Duplicator wins.

We say that Spoiler (and Duplicator, respectively) *wins the bijective k-pebble game* $\mathrm{BP}_k(G,H)$ if Spoiler (and Duplicator, respectively) has a winning strategy for the game.

The following theorem describes the correspondence between the Weisfeiler-Leman algorithm and the introduced pebble games.[1]

▶ **Theorem 1** (see e.g. [9]). *Let G and H be two graphs. Then $G \simeq_k H$ if and only if Duplicator wins the game* $\mathrm{BP}_{k+1}(G,H)$.

Association Schemes. Let V be a set. An *association scheme* on V is an ordered partition $(R_0,\dots,R_d)$ of V^2 such that

1. $R_0 = \{(v,v) \mid v \in V\}$, and
2. for every $i \in [d]$, there is a $j \in [d]$ such that the set $R_i^\intercal := \{(w,v) \mid (v,w) \in R_i\}$ equals R_j, and
3. for all i,j,k, there are numbers $p_{i,j}^k$ such that for all $(v,w) \in R_k$,

$$p_{i,j}^k = \big|\{x \in V \mid (v,x) \in R_i \text{ and } (w,x) \in R_j\}\big|.$$

An association scheme is *symmetric* if $R_i^\intercal = R_i$ for all $i \in [d]$. With each R_i, we associate a directed *constituent graph $G(R_i)$ of the association scheme*, defined as $G(R_i) := (V, R_i \cup R_i^\intercal)$.

[1] The pebble games in [9] are defined slightly differently. Still, a player has a winning strategy in the game described there if and only if they have one in our game and thus, Theorem 1 holds for both versions of the game.

Every association scheme induces a coloring on V^2, in which every (v, w) is colored with the relation it is contained in. This coloring is stable in the sense that it is not refined by the 2-dimensional WL algorithm (when V is interpreted as the vertex set of a complete directed graph). That is, for all $j \in [d]$ and all $(v_1, v_2), (w_1, w_2) \in R_j$, it holds that $\chi_{G(R_i)}(v_1, v_2) = \chi_{G(R_i)}(w_1, w_2)$. Conversely, every colored graph G with $\chi_G(v, v) = \chi_G(w, w)$ for all $v, w \in V(G)$ induces an association scheme in which the relations R_i are the color classes of the coloring $\chi_G = \chi_{G,2}$.

3 One Color

Our first goal is to prove that the 2-dimensional WL algorithm distinguishes vertex pairs that are separators in a graph from other pairs of vertices. We start with an analysis of the graphs in which all vertices are assigned the same color by the WL algorithm. In particular, this includes all constituent graphs of association schemes.

A main tool for the analysis are distance patterns of vertices. For a graph G and a vertex $v \in V(G)$, let $D(v) \coloneqq \{\{\mathrm{dist}(v, w) \mid w \in V(G)\}\}$. Note that for vertices $u, v \in V(G)$ it holds that $\chi_G(u, u) \neq \chi_G(v, v)$ whenever $D(u) \neq D(v)$ since the 2-dimensional WL algorithm detects distances between vertex pairs.“

▶ **Lemma 2.** *Let G be a graph and $uv \in E(G)$. Suppose that $D(u) = D(v)$. Then*

$$\begin{aligned} &\{\{\mathrm{dist}(u, w) \mid w \in V(G) \colon \mathrm{dist}(u, w) < \mathrm{dist}(v, w)\}\} \\ = &\{\{\mathrm{dist}(v, w) \mid w \in V(G) \colon \mathrm{dist}(v, w) < \mathrm{dist}(u, w)\}\}. \end{aligned}$$

Throughout the remainder of this section, if not explicitly stated otherwise, we make the following assumption.

▶ **Assumption 3.** G is a connected graph on n vertices with the following properties:

1. $\chi_G(u, u) = \chi_G(v, v)$ for all $u, v \in V(G)$, and
2. G has a 2-separator $\{w_1, w_2\}$.

In the rest of this section, we analyze the structure of G and ultimately prove that G must be a cycle. In particular, this completely characterizes constituent graphs of association schemes that are connected, but not 3-connected.

Note that Assumption 3 implies that G is regular, i.e., $\deg(u) = \deg(v)$ for all $u, v \in V(G)$.

▶ **Lemma 4.** *G is 2-connected, i.e., G does not contain any cut vertex.*

This is a consequence of Condition 1 in Assumption 3, since the 2-dimensional WL algorithm distinguishes cut vertices from other vertices (see [29, Corollary 7]). Note that the lemma implies that each of w_1 and w_2 has at least one neighbor in each of the connected components of $G - w_1w_2$.

▶ **Lemma 5.** *Let C be the vertex set of a connected component of $G - w_1w_2$ such that $|C| < \frac{n}{2}$ and let $v \in C$. Then there is no vertex $u \in N(v)$ such that $\mathrm{dist}(u, w_1) < \mathrm{dist}(v, w_1)$ and $\mathrm{dist}(u, w_2) < \mathrm{dist}(v, w_2)$.*

Proof. Suppose towards a contradiction that such a vertex $u \in N(v)$ exists. For all $w \in V(G)$, we have $|\mathrm{dist}(v, w) - \mathrm{dist}(u, w)| \leq 1$, since $uv \in E(G)$. Furthermore, it holds that $\sum_{w \in V(G)} \big(\mathrm{dist}(v, w) - \mathrm{dist}(u, w)\big) = 0$ because $D(u) = D(v)$ due to Condition 1 in Assumption 3. But $\mathrm{dist}(v, w) > \mathrm{dist}(u, w)$ for all $w \in V(G) \setminus C$, and $|V(G) \setminus C| > \frac{n}{2}$. This is a contradiction. ◀

▶ **Lemma 6.** *Let $d := \mathrm{dist}(w_1, w_2)$ and let C be the vertex set of a connected component of $G - w_1w_2$ such that $|C| \leq \frac{n-2}{2}$. Then for all $v \in C \cup \{w_1, w_2\}$ and all $i \in \{1, 2\}$, it holds that $\mathrm{dist}(v, w_i) \leq d$.*

Proof. By symmetry, it suffices to prove $\mathrm{dist}(v, w_2) \leq d$. The statement is proved by induction on $\ell := \mathrm{dist}(v, w_1)$. For $\ell = 0$, it holds that $v = w_1$ and $\mathrm{dist}(w_1, w_2) = d$. So suppose the statement holds for all $u \in C \cup \{w_1, w_2\}$ with $\mathrm{dist}(u, w_1) \leq \ell$. Obviously, the statement is true if $v = w_1$ or $v = w_2$. So pick $v \in C$ with $\mathrm{dist}(v, w_1) = \ell + 1$. Let $u \in N(v)$ such that $\mathrm{dist}(u, w_1) \leq \ell$. Then $\mathrm{dist}(v, w_2) \leq \mathrm{dist}(u, w_2) \leq d$ by Lemma 5 and the induction hypothesis. ◀

▶ **Lemma 7.** *$w_1w_2 \notin E(G)$.*

Proof. Suppose towards a contradiction that $w_1w_2 \in E(G)$. Let C be the vertex set of a connected component of $G - w_1w_2$ such that $|C| \leq \frac{n-2}{2}$. By Lemma 6, we conclude that $C \subseteq N(w_1) \cap N(w_2)$. Let $v \in C$. Since G is 2-connected, the vertex w_1 must have at least one neighbor in $V(G) \setminus C$, in addition to being adjacent to C and to w_2. Thus, $\deg(w_1) \geq |C| + 2 > |C| - 1 + |\{w_1, w_2\}| \geq \deg(v)$, which contradicts G being a regular graph. ◀

▶ **Lemma 8.** *Suppose that $N(w_1) \cap N(w_2) \neq \emptyset$. Then G is a cycle.*

Proof. By Lemma 7, it holds that $w_1w_2 \notin E(G)$. Furthermore, by the assumption of the lemma, we have $\mathrm{dist}(w_1, w_2) = 2$. Let C be the vertex set of a connected component of $G - w_1w_2$ such that $|C| \leq \frac{n-2}{2}$. Also let $C' := V(G) \setminus (C \cup \{w_1, w_2\})$. For $i, j \geq 1$ let

$$C_{i,j} := \{v \in C \mid \mathrm{dist}(v, w_1) = i \text{ and } \mathrm{dist}(v, w_2) = j\}.$$

By Lemma 6, we conclude that $C_{i,j} = \emptyset$ unless $(i, j) \in \{(1, 1), (1, 2), (2, 1), (2, 2)\}$.

Suppose there exists $v \in C_{1,2}$. We have $D(v) = D(w_1)$ and, by Lemma 4, also $N(w_1) \cap C' \neq \emptyset$. Thus, there is some vertex $u' \neq w_1$ such that $\mathrm{dist}(w_1, u') < \mathrm{dist}(v, u')$ and therefore, by Lemma 2, there is also a vertex $u \neq v$ such that $\mathrm{dist}(v, u) < \mathrm{dist}(w_1, u)$. For every such vertex u, it holds that $\mathrm{dist}(w_1, u) \leq 2$ and thus, $u \in N(v)$. Therefore, by Lemma 2, for every vertex $v' \neq w_1$ with $\mathrm{dist}(w_1, v') < \mathrm{dist}(v, v')$, it holds that $v' \in N(w_1)$. This implies that there is no $v' \in C'$ such that $\mathrm{dist}(w_1, v') = 2$ since such a vertex would satisfy $3 = \mathrm{dist}(v, v') > \mathrm{dist}(w_1, v')$.

Because w_2 is not a cut vertex (cf. Lemma 4), from every $v' \in C'$, there is a path to w_1 that does not contain w_2. However, this is only possible if there is no vertex $v' \in C'$ such that $\mathrm{dist}(w_1, v') > 1$, in other words: $C' \subseteq N(w_1)$. Since G is regular and $|N(w_1) \setminus C'| \geq 1$, it follows that $\deg(v) \geq |C'| + 1$. But $N(v) \subseteq (C \cup \{w_1\}) \setminus \{v\}$, which implies $\deg(v) \leq |C|$. The combination of both inequalities yields $|C'| + 1 \leq |C|$, which implies $n = 2 + |C| + |C'| \leq 1 + 2|C| \leq n - 1$, a contradiction. So $C_{1,2} = \emptyset$ and by symmetry, it also holds that $C_{2,1} = \emptyset$. But then $C_{2,2} = \emptyset$ by Lemma 5.

So $C = C_{1,1}$, which means that $C \subseteq N(w_1) \cap N(w_2)$. In particular, $\deg(w_1) \geq |C| + 1$ since $|N(w_1) \cap C'| \geq 1$. Since G is regular, this implies that $\deg(v) \geq |C| + 1$ for every $v \in C$, which is only possible if $N[v] = C \cup \{w_1, w_2\}$. Because $C \neq \emptyset$, this means that there is a vertex $v \in V(G)$ such that $G[N[v]]$ contains only one non-edge. Now by Condition 1 in Assumption 3, this also has to hold for w_1, and hence, since no vertex in $C \cap N(w_1)$ is adjacent to any vertex in $C' \cap N(w_1)$, it must hold that $\deg(w_1) = 2$. Therefore, by regularity, all vertices in G have degree 2 and thus, being connected, G is a cycle. ◀

▶ **Lemma 9.** *G is a cycle.*

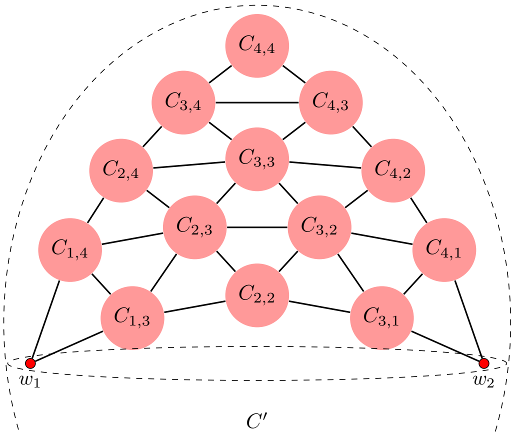


Figure 1 Visualization of the sets $C_{i,j}$ for $d = 4$ in the proof of Lemma 9. An arc between two sets indicates that there may be edges connecting vertices from the two sets.

Proof Sketch. We only present a sketch here. It suffices to prove the lemma for the case that G is a graph with a maximum edge set that satisfies Assumption 3. Indeed, if such a graph G is a cycle, then G has n edges, and the lemma trivially holds for every graph with less edges since every connected regular graph has at least n edges.

Let $d \coloneqq \text{dist}(w_1, w_2)$. By Lemmas 7 and 8, we can assume that $d \geq 3$. Let C be the vertex set of a connected component of $G - w_1w_2$ such that $|C| \leq \frac{n-2}{2}$. Also let $C' \coloneqq V(G) \setminus (C \cup \{w_1, w_2\})$. For $i, j \geq 1$ let

$$C_{i,j} \coloneqq \{v \in C \mid \text{dist}(v, w_1) = i \text{ and } \text{dist}(v, w_2) = j\}.$$

By Lemma 6, we conclude that $C_{i,j} = \emptyset$ unless $i, j \leq d$. Furthermore, by the definition of d, we have that $C_{i,j} = \emptyset$ unless $i + j \geq d$. This situation is also visualized in Figure 1.

Using similar ideas as in the proof of Lemma 8, the first step is to show that $C_{d,d} = C_{d-1,d} = C_{d,d-1} = \emptyset$. The main part of the proof is based on the following claim, which exploits the edge maximality of the graph G.

▷ Claim 10. Let $u, v \in V(G)$ such that $\text{dist}(u, v) < d$. Then there is a unique shortest path from u to v.

Indeed, if there were a pair of vertices $u, v \in V(G)$ of distance $\ell < d$ with two shortest paths, then u and v would be "on the same side" of the separator w_1w_2. Hence, since the 2-dimensional WL algorithm is capable of detecting such pairs, adding all of them to the edge set of G would result in a graph with a larger number of edges that still fulfills Assumption 3. Using similar arguments, we also obtain the following related insight.

▷ Claim 11. Let $u, v \in V(G)$ such that $\ell \coloneqq \text{dist}(u, v) < d$. Furthermore, suppose there is a walk $u = u_0, \ldots, u_{\ell+1} = v$ of length $\ell + 1$ from u to v. Then there is some $i \in [\ell]$ such that $u_{i-1}u_{i+1} \in E(G)$.

These two claims drastically restrict the structure of the graph G and allow us to prove that G is a cycle. Intuitively speaking, the claims imply that, when looking towards the connected component $G[C]$ from any of the w_i, the graph has a tree-like shape, i.e., the initial segments of paths up to length $d - 1$ starting in w_i form a tree rooted in w_i. However, this only works out if G is a cycle. More formally, for $k' \in \{d, \dots, 2d\}$, let $\mathcal{C}_{k'} := \bigcup_{i,j:i+j=k'} C_{i,j}$ and let $k \in \{d, \dots, 2d\}$ be the maximal number such that $\mathcal{C}_k \neq \emptyset$. Then $k \leq 2d-2$ since we already know that $C_{d,d} = C_{d-1,d} = C_{d,d-1} = \emptyset$.

Consider a set $C_{i,k-i}$ for some i with $k-d+1 \leq i \leq d-1$ (i.e., none of the extremal sets). Then, for all $v \in C_{i,k-i}$, it holds that $\text{dist}(v, w_1) < d$ and $\text{dist}(v, w_2) < d$, which allows us to apply Claims 10 and 11 for shortest paths from v to w_1 and w_2. With this, we show that if $C_{i,k-i} \neq \emptyset$, then there is a vertex of degree 2, which implies that G is a cycle, since it is regular. In the other case, $C_{j,k-j} = \emptyset$ for all j with $k-d+1 \leq j \leq d-1$. We then consider the extremal sets and derive a contradiction from there. ◀

Reformulating the previous lemma, we obtain the following theorem.

▶ **Theorem 12.** *Let G be a graph such that $\chi_G(u,u) = \chi_G(v,v)$ for all $u, v \in V(G)$. Then (exactly) one of the following holds:*

1. *G is not connected, or*
2. *G is 3-connected, or*
3. *G is a cycle of length $\ell \geq 4$.*

Note that the complete graphs on two and three vertices are 3-connected. The theorem also implies that a connected constituent graph of an association scheme is either 3-connected or a cycle (for other work on the connectivity of relations in association schemes, see e.g. [6, 7, 8, 13]). It thus provides a generalization of Kodalen's and Martin's result in [30], where they proved the theorem in case the graph stems from a symmetric association scheme.

4 Two Colors

Recall that our overall goal is to prove that the 2-dimensional WL algorithm assigns special colors to 2-separators in a graph. We will use Lemma 9 to prove this in case the tuples (u,u) and (v,v) of a 2-separator $\{u,v\}$ obtain the same color under the 2-dimensional WL algorithm. To treat the much more difficult case that u and v obtain distinct colors, we intend to generalize the results of the previous section to two vertex colors. Maybe somewhat surprisingly, we obtain a similar statement to Lemma 9. However, now we require the input graphs to be 2-connected (instead of only being connected). This is a necessary condition, since for example the star graphs $K_{1,n}$ for $n \geq 2$ are neither 3-connected nor cycles but still have only two vertex colors under the 2-dimensional WL algorithm.

The route to proving the statement is similar to the one described in Section 3. Still, two colors allowing for more complexity in the graph structure, the statements and lemmas become more involved and additional cases need to be considered.

▶ **Lemma 13.** *Let G be a graph and suppose there are $c_1 \neq c_2$ such that $\{\chi_G(v,v) \mid v \in V(G)\} = \{c_1, c_2\}$ for some $c_1 \neq c_2 \in \mathbb{N}$. Let $U := \{u \in V(G) \mid \chi_G(u,u) = c_1\}$ and $V := \{v \in V(G) \mid \chi_G(v,v) = c_2\}$. Also let $U_1, \dots, U_k$ be the vertex sets of the connected components of $G[U]$ and let $V_1, \dots, V_\ell$ be the vertex sets of the connected components of $G[V]$. Let G' be the graph with $V(G') = \{U_1, \dots, U_k, V_1, \dots, V_\ell\}$ and $U_iV_j \in E(G')$ if and only if there are $u \in U_i$, $v \in V_j$ such that $uv \in E(G)$.*

Then $\chi_{G'}(U_i, U_i) = \chi_{G'}(U_j, U_j)$ for all $i, j \in [k]$ and $\chi_{G'}(V_i, V_i) = \chi_{G'}(V_j, V_j)$ for all $i, j \in [\ell]$.

▶ **Theorem 14.** *Let G be a 2-connected graph with the following properties:*

1. *G has a 2-separator w_1w_2, and*
2. *for every $v \in V(G)$, there is an $i \in \{1, 2\}$ such that $\chi_G(v,v) = \chi_G(w_i, w_i)$.*

Then G is a cycle.

Proof Idea. By Lemma 9, we can assume without loss of generality that $\chi_G(w_1, w_1) \neq \chi_G(w_2, w_2)$. The statement is proved by induction on the graph size n. For $n \leq 4$, a simple case analysis among the possible graphs G yields the statement.

So let $n \geq 5$. Again, it suffices to prove the statement for the case that G is an n-vertex graph with a maximum edge set that satisfies the requirements of the theorem. Let $U := \{u \in V(G) \mid \chi_G(u, u) = \chi_G(w_1, w_1)\}$ and $V := \{v \in V(G) \mid \chi_G(v, v) = \chi_G(w_2, w_2)\}$. Let $U_1, \dots, U_k$ be the vertex sets of the connected components of $G[U]$ and let $V_1, \dots, V_\ell$ be the vertex sets of the connected components of $G[V]$. Without loss of generality, assume that $w_1 \in U_1$ and $w_2 \in V_1$. Let C be the vertex set of a connected component of $G - w_1w_2$ such that $|C| \leq \frac{n-2}{2}$. Also let $C' := V(G) \setminus (C \cup \{w_1, w_2\})$.

Consider the graph G' defined in the same way as in Lemma 13. First suppose that G' has fewer vertices than the original graph G. If the induction hypothesis is applicable to the graph G', it follows that G' is a cycle and from there, it is not difficult to prove that G also must be a cycle. However, it may happen that G' is 3-connected in which case the induction hypothesis cannot be applied. But this can only happen in very specific cases, which we then treat separately.

If $|V(G')| = |V(G)|$, then there are no edges connecting vertices of the same color with respect to the 2-dimensional WL algorithm and thus, G is bipartite. In this case, we can proceed very similarly to the proof of Lemma 9. ◀

5 Detecting Decompositions with Weisfeiler and Leman

Let S be a set of colors. We say a path $u_0, \dots, u_\ell$ *avoids* S if $\chi_G(u_i, u_i) \notin S$ for every $i \in [\ell - 1]$. Note that we impose no restriction on the colors of the endpoints of the path. It is easy to see that, given two vertices $u, v \in V(G)$, the 2-dimensional WL algorithm is aware of whether there is a path from u to v that avoids S.

▶ **Lemma 15.** *Let G be a graph and let $X := \{\chi_G(v, v) \mid v \in V(G)\}$. Furthermore, let $S \subseteq X$ and define $G[[S]] = (V, E)$, where $V = \{v \in V(G) \mid \chi_G(v, v) \in S\}$ and*

$$E = \{uv \mid \text{there is a path from } u \text{ to } v \text{ in } G \text{ that avoids } S\}.$$

Then $\chi_{G[[S]]}(u, u) = \chi_{G[[S]]}(v, v)$ for all $u, v \in V$ with $\chi_G(u, u) = \chi_G(v, v)$.

▶ **Theorem 16.** *Let G and H be 2-connected graphs and let $w_1, w_2 \in V(G)$ such that w_1w_2 forms a 2-separator in G. Also let $v_1, v_2 \in V(H)$ and suppose $\chi_G(w_1, w_2) = \chi_H(v_1, v_2)$. Then v_1v_2 forms a 2-separator in H.*

Proof. Let $S := \{\chi_G(w_1, w_1), \chi_G(w_2, w_2)\}$ and let $G' := G[[S]]$. Clearly, the graph G' is connected. We argue that G' is 2-connected. Suppose towards a contradiction that there is a separating vertex w in G'. Let C and C' be the vertex sets of two connected components of $G' - w$ and let $v \in C$ and $v' \in C'$. We show that w separates v from v' in G. Towards a contradiction, suppose there is a path P from v to v' in G that does not pass w. Then there is a corresponding path P' in G', which simply skips all inner vertices of P not contained in S. In particular, P' connects v and v', but avoids w. This contradicts w being a cut vertex in G'. Hence, G' is 2-connected.

First suppose that $|V(G')| = 2$. Let $A := \{v \in V(H) \mid \chi_H(v, v) \in S\}$. Then $|A| = 2$ and thus $A = \{v_1, v_2\}$. Moreover, $H - A$ is disconnected, since the 2-dimensional WL algorithm detects that $G - w_1w_2$ is disconnected. Hence, v_1v_2 forms a 2-separator in H.

Now assume $|V(G')| \geq 3$ and suppose there is a vertex set C of a connected component of $G - w_1w_2$ such that $V(G') \subseteq C \cup \{w_1, w_2\}$. Let C' be the vertex set of a second connected component of $G - w_1w_2$ and let $v \in C'$. Then w_1 and w_2 are the only vertices with color in S that can be reached from v via a path that avoids S. Hence, using the expressive power of the 2-dimensional WL algorithm, it is not hard to see that there must also be some $u \in V(H)$ such that v_1 and v_2 are the only vertices with color in S that can be reached from u via a path that avoids S. Since $|V(G')| \geq 3$, there is some $u' \in V(H)$ such that $v_1 \neq u' \neq v_2$ and $\chi_H(u', u') \in S$ because in order not to be distinguished, unions of color classes with color in S must have the same cardinality in both graphs. But then v_1v_2 separates u from u' in H and thus, v_1v_2 forms a 2-separator in H.

In the other case, w_1w_2 forms a 2-separator in G' and hence, G' is a cycle by Lemma 15 and Theorem 14. Note that $|V(G')| \geq 4$ and $w_1w_2 \notin E(G')$. It follows that $H[[S]]$ is also a cycle, since otherwise, the 2-dimensional WL algorithm would distinguish the graphs. Also, $|V(H[[S]])| \geq 4$ and $v_1v_2 \notin E(H[[S]])$. So v_1v_2 forms a 2-separator in $H[[S]]$ and thus, it also forms a 2-separator in H. ◀

▶ **Corollary 17.** *Suppose $k \geq 2$. Let G and H be connected graphs. Suppose $\{w_1, \dots, w_k\} \subseteq V(G)$ is a k-separator in G. Let $\{v_1, \dots, v_k\} \subseteq V(H)$ and suppose $\chi_{G,k}(w_1, \dots, w_k) = \chi_{H,k}(v_1, \dots, v_k)$. Then $\{v_1, \dots, v_k\}$ forms a k-separator in H.*

Using the corollary, we can prove a strengthened version of Theorem 13 in [29]. Following [29], we say that the k-dimensional WL algorithm *determines orbits* in a graph class $\mathcal{G}$ if for all arc-colored graphs (G, λ), (G', λ') with $G, G' \in \mathcal{G}$, arc colorings λ, λ' and for all vertices $v \in V(G)$ and $v' \in V(G')$ there exists an isomorphism from (G, λ) to (G', λ') mapping v to v' if and only if $\chi_{G,k}(v, \dots, v) = \chi_{G',k}(v', \dots, v')$.

▶ **Theorem 18.** *Let $\mathcal{G}$ be a minor-closed graph class and assume $k \geq 2$. Suppose the k-dimensional WL algorithm determines orbits on all arc-colored 3-connected graphs in $\mathcal{G}$. Then the k-dimensional WL algorithm distinguishes all non-isomorphic graphs in $\mathcal{G}$.*

Thus, since by [29], the WL dimension of the class of planar graphs is 2 or 3, the concrete value only depends on the dimension needed to determine orbits on arc-colored triconnected planar graphs.

For a graph G and $v_1, v_2, v_3 \in V(G)$ we define $s_G(v_1, v_2, v_3) := |C|$, where C is the vertex set of the connected component of $G - v_1v_2$ that contains v_3 (if $v_3 \in \{v_1, v_2\}$, then $s_G(v_1, v_2, v_3) := 0$).

▶ **Theorem 19.** *Let G and H be two 2-connected graphs. Also let $v_1, v_2, v_3 \in V(G)$ and $w_1, w_2, w_3 \in V(H)$ such that $\chi_G(v_i, v_j) = \chi_H(w_i, w_j)$ for all $i, j \in \{1, 2, 3\}$. Then $s_G(v_1, v_2, v_3) = s_H(w_1, w_2, w_3)$.*

The last theorem can also be formulated in terms of the expressive power of the 3-variable fragment C^3 of first-order logic with counting quantifiers of the form $\exists^{\geq k} x \varphi(x)$. Indeed, it implies that for all $n, s \in \mathbb{N}$, there is a formula $\varphi_{n,s}(x_1, x_2, x_3) \in \mathsf{C}^3$ such that, for every 2-connected n-vertex graph G and $v_1, v_2, v_3 \in V(G)$, it holds that $G \models \varphi_{n,s}(v_1, v_2, v_3)$ if and only if $s_G(v_1, v_2, v_3) = s$ (for details about the connection between the WL algorithm and counting logics, see e.g. [9, 27]).

6 New Bounds for Graphs of Treewidth k

As an application of the results presented so far, we investigate the WL dimension of graphs of treewidth at most k. Up to this point, the best known upper bound on the WL dimension of such graphs has been $k+2$, i.e., the $(k+2)$-dimensional WL algorithm identifies every graph of treewidth at most k [18]. In this section, we present new upper and lower bounds.

6.1 Upper Bound

The basic idea for proving a new upper bound is to provide a winning strategy for Spoiler in the corresponding bijective pebble game and it works similarly to the proof that the $(k+2)$-dimensional WL algorithm identifies every graph of treewidth at most k [18]. The main difference is a much more careful implementation of the general strategy in order to get by with the desired number of pebbles. As a major ingredient, we exploit that separators can be detected using fewer pebbles.

For a $(k+1)$-tuple $(v_1, \dots, v_k, v_{k+1})$ of vertices of a graph G, we define $s_G(v_1, \dots, v_k, v_{k+1}) := |C|$ where C is the unique connected component of $G - \{v_1, \dots, v_k\}$ with $v_{k+1} \in C$.

▶ **Corollary 20.** *Suppose $k \geq 2$. Let G, H be two graphs and let $v_1, \dots, v_{k+1} \in V(G)$ and $w_1, \dots, w_{k+1} \in V(H)$ such that $s_G(v_1, \dots, v_k, v_{k+1}) \neq s_H(w_1, \dots, w_k, w_{k+1})$. Then Spoiler wins the game $\mathrm{BP}_{k+1}(G, H)$ from the initial position $\big((v_1, \dots, v_{k+1}), (w_1, \dots, w_{k+1})\big)$.*

To build Spoiler's strategy along a given tree decomposition, we use the following characterization of treewidth. Let G be a graph of treewidth k. For a k-separator $S \subseteq V(G)$ and the vertex set C of a connected component of $G - S$, we define $G(S, C)$ to be the graph on vertex set $S \cup C$ obtained by inserting a clique between the vertices in S into $G[S \cup C]$.

▶ **Lemma 21** (Arnborg et al. [1]). *Suppose $G(S, C)$ has at least $k+2$ vertices. Then $G(S, C)$ has treewidth at most k if and only if there exists $v \in C$ such that for every connected component A of $G[C \setminus \{v\}]$, there is a k-element separator $S_A \subseteq S \cup \{v\}$ such that*

1. *no vertex in A is adjacent to the unique element from $S \setminus S_A$, and*
2. *$G\big(S_A, V(A)\big)$ has treewidth at most k.*

Suppose $G(S, C)$ has treewidth at most k. Let $D_G(S, C)$ denote the set of possible vertices $v \in C$ that satisfy Lemma 21.

▶ **Theorem 22.** *Suppose $k \geq 2$. Let G be a graph of treewidth at most k. Then the k-dimensional WL algorithm identifies G.*

Proof Idea. Given a graph G of treewidth at most k and a second non-isomorphic graph H, we show that Spoiler wins the game $\mathrm{BP}_{k+1}(G, H)$. For simplicity of notation, we view tuples $\bar{a} = (a_1, \dots, a_k)$ also as sets $\{a_1, \dots, a_k\}$. Suppose the game is at a position $(\bar{a}, \bar{b}) \in \big(V(G)\big)^k \times \big(V(H)\big)^k$ such that $G(\bar{a}, C)$ has treewidth k for every connected component C of $G - \bar{a}$. Let $m := m(\bar{a}, \bar{b})$ be the smallest number such that $G - \bar{a}$ and $H - \bar{b}$ are not isomorphic when only considering connected components of size m. We prove by induction on m that Spoiler wins the game. The case $m = 1$ is easy, since there is still one pebble left. For the case $m > 1$, Spoiler finds a connected component C_G of $G - \bar{a}$ of size m that differs from the corresponding connected component C_H (specified by Duplicator's bijection) in the graph $H - \bar{b}$. By Corollary 20, we can assume that $|C_G| = |C_H|$. Then Spoiler places a pebble on a vertex in the set $D_G(\bar{a}, C_G)$. This splits C_G into smaller connected components. Then, it can be proved that one pebble can be removed so that afterwards, we find connected components of size at most $m-1$ in which G and H differ (again using Corollary 20). ◀

6.2 Lower Bound

For the lower bound we use a construction introduced by Cai, Fürer, and Immerman [9]. For a graph G, we let $\mathrm{CFI}(G)$ be the graph obtained by applying the standard Cai, Fürer, Immerman (CFI-) construction to G, and $\mathrm{CFI}^{\times}(G)$ be the graph obtained by applying the CFI-construction with one pair of edges twisted (see [9]). We shall actually not require any further details on how the graphs $\mathrm{CFI}(G)$ and $\mathrm{CFI}^{\times}(G)$ look, since we exploit the following theorem.

▶ **Theorem 23** (Dawar and Richerby [10])**.** *Let G be a connected graph such that $\mathrm{tw}(G) \geq k+1$ and $\deg(v) \geq 2$ for all $v \in V(G)$. Then $\mathrm{CFI}(G) \simeq_k \mathrm{CFI}^{\times}(G)$.*

The strategy to obtain a good lower bound is to find graphs G where we can show a sufficiently good upper bound on the treewidth of $\mathrm{CFI}(G)$ and $\mathrm{CFI}^{\times}(G)$. For $n \geq 2$, let $G_{n,n}$ be the $n \times n$ grid.

▶ **Lemma 24.** *For $n \geq 2$, it holds that $\mathrm{tw}(\mathrm{CFI}(G_{n,n})) \leq 2n+5$ and $\mathrm{tw}(\mathrm{CFI}^{\times}(G_{n,n})) \leq 2n+5$.*

▶ **Theorem 25.** *For every $k \geq 2$, there are non-isomorphic graphs G_k and H_k of treewidth at most $2k+7$ such that $G_k \simeq_k H_k$.*

Proof. Let $G_k := \mathrm{CFI}(G_{k+1,k+1})$ and $H_k := \mathrm{CFI}^{\times}(G_{k+1,k+1})$. Then the statement follows from Theorem 23 and Lemma 24. ◀

For a graph class $\mathcal{C}$, denote by $\dim_{\mathsf{WL}}(\mathcal{C})$ the *WL dimension* of $\mathcal{C}$, i.e., the minimum $k \in \mathbb{N} \cup \{\infty\}$ such that the k-dimensional WL algorithm identifies every graph $G \in \mathcal{C}$. As a corollary from Theorems 22 and 25, we obtain the following result.

▶ **Corollary 26.** *Let $k \geq 2$. Then $\lceil \frac{k}{2} \rceil - 3 \leq \dim_{\mathsf{WL}}(\mathcal{T}_k) \leq k$, where $\mathcal{T}_k$ denotes the class of graphs of treewidth at most k.*

7 Conclusion

We have proved that for $k \geq 2$, the k-dimensional WL algorithm implicitly computes the decomposition of its input graph into its triconnected components. As a by-product, we found that every connected constituent graph of an association scheme is either a cycle or 3-connected.

We have applied this insight to improve on the upper bound on the WL dimension of graphs of bounded treewidth and have also provided a lower bound that is asymptotically only a factor of 2 away from the upper bound.

A natural use case of our results may be determining the WL dimension of certain graph classes that satisfy the requirements of Theorem 18. We conjecture that the 2-dimensional WL algorithm identifies every planar graph. Indeed, using the results of this paper, it essentially suffices to show this for triconnected planar graphs.

References

1 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of Finding Embeddings in a K-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, April 1987. doi:10.1137/0608024.

2 Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. On Weisfeiler-Leman Invariance: Subgraph Counts and Related Graph Properties. *CoRR*, abs/1811.04801, 2018. arXiv:1811.04801.

3 Albert Atserias and Elitza N. Maneva. Sherali-Adams Relaxations and Indistinguishability in Counting Logics. *SIAM J. Comput.*, 42(1):112–137, 2013. doi:10.1137/120867834.

4 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.

5 László Babai, Xi Chen, Xiaorui Sun, Shang-Hua Teng, and John Wilmes. Faster Canonical Forms for Strongly Regular Graphs. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 157–166. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.25.

6 Andries E. Brouwer. Spectrum and connectivity of graphs. *CWI Quarterly*, 9(1-2):37–40, 1996.

7 Andries E. Brouwer and Jack H. Koolen. The vertex-connectivity of a distance-regular graph. *Eur. J. Comb.*, 30(3):668–673, 2009. doi:10.1016/j.ejc.2008.07.006.

8 Andries E. Brouwer and Dale M. Mesner. The Connectivity of Strongly Regular Graphs. *Eur. J. Comb.*, 6(3):215–216, 1985. doi:10.1016/S0195-6698(85)80030-5.

9 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992. doi:10.1007/BF01305232.

10 Anuj Dawar and David Richerby. The Power of Counting Logics on Restricted Classes of Finite Structures. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2007. doi:10.1007/978-3-540-74915-8_10.

11 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász Meets Weisfeiler and Leman. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 40:1–40:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.40.

12 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

13 Sergei Evdokimov and Ilia Ponomarenko. On the vertex connectivity of a relation in an association scheme. *Journal of Mathematical Sciences*, 134(5):2354–2357, May 2006. doi:10.1007/s10958-006-0112-z.

14 Martin Fürer. On the Combinatorial Power of the Weisfeiler-Lehman Algorithm. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 260–271, 2017. doi:10.1007/978-3-319-57586-5_22.

15 Martin Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. *J. ACM*, 59(5):27:1–27:64, 2012. doi:10.1145/2371656.2371662.

16 Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017.

17 Martin Grohe and Sandra Kiefer. A Linear Upper Bound on the Weisfeiler-Leman Dimension of Graphs of Bounded Genus. *CoRR*, abs/1904.07216, 2019. arXiv:1904.07216.

18 Martin Grohe and Julian Mariño. Definability and Descriptive Complexity on Databases of Bounded Tree-Width. In Catriel Beeri and Peter Buneman, editors, *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings.*, volume 1540 of *Lecture Notes in Computer Science*, pages 70–82. Springer, 1999. doi:10.1007/3-540-49257-7_6.

19 Martin Grohe and Daniel Neuen. Canonisation and Definability for Graphs of Bounded Rank Width. *CoRR*, abs/1901.10330, 2019. arXiv:1901.10330.

20 Martin Grohe and Martin Otto. Pebble Games and linear equations. *J. Symb. Log.*, 80(3):797–844, 2015. doi:10.1017/jsl.2015.28.

21 Lauri Hella. Logical Hierarchies in PTIME. *Inf. Comput.*, 129(1):1–19, 1996. doi:10.1006/inco.1996.0070.

22 John E. Hopcroft and Robert Endre Tarjan. A V^2 Algorithm for Determining Isomorphism of Planar Graphs. *Inf. Process. Lett.*, 1(1):32–34, 1971. doi:10.1016/0020-0190(71)90019-6.

23 John E. Hopcroft and Robert Endre Tarjan. Isomorphism of Planar Graphs. In Raymond E. Miller and James W. Thatcher, editors, *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, USA*, The IBM Research Symposia Series, pages 131–152. Plenum Press, New York, 1972.

24 John E. Hopcroft and Robert Endre Tarjan. A V log V Algorithm for Isomorphism of Triconnected Planar Graphs. *J. Comput. Syst. Sci.*, 7(3):323–331, 1973. doi:10.1016/S0022-0000(73)80013-3.

25 John E. Hopcroft and Robert Endre Tarjan. Dividing a Graph into Triconnected Components. *SIAM J. Comput.*, 2(3):135–158, 1973. doi:10.1137/0202012.

26 John E. Hopcroft and J. K. Wong. Linear Time Algorithm for Isomorphism of Planar Graphs (Preliminary Report). In Robert L. Constable, Robert W. Ritchie, Jack W. Carlyle, and Michael A. Harrison, editors, *Proceedings of the 6th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1974, Seattle, Washington, USA*, pages 172–184. ACM, 1974. doi:10.1145/800119.803896.

27 Neil Immerman and Eric Lander. Describing Graphs: A First-Order Approach to Graph Canonization. In *Complexity Theory Retrospective*, pages 59–81. Springer New York, 1990. doi:10.1007/978-1-4612-4478-3_5.

28 Sandra Kiefer and Daniel Neuen. The power of the Weisfeiler-Leman algorithm to decompose graphs. *CoRR*, abs/1908.05268, 2019. arXiv:1908.05268.

29 Sandra Kiefer, Ilia Ponomarenko, and Pascal Schweitzer. The Weisfeiler-Leman dimension of planar graphs is at most 3. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005107.

30 Brian G. Kodalen and William J. Martin. On the Connectivity of Graphs in Association Schemes. *Electr. J. Comb.*, 24(4):P4.39, 2017. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v24i4p39.

31 Brendan D. McKay. Practical graph isomorphism. *Congr. Numer.*, 30:45–87, 1981.

32 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

33 Christopher Morris and Petra Mutzel. Towards a practical k-dimensional Weisfeiler-Leman algorithm. *CoRR*, abs/1904.01543, 2019. arXiv:1904.01543.

34 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. *CoRR*, abs/1810.02244, 2018. arXiv:1810.02244.

35 Daniel Neuen. Graph Isomorphism for Unit Square Graphs. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 70:1–70:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.70.

36 William T. Tutte. *Graph theory.* Encyclopedia of mathematics and its applications. Addison-Wesley Pub. Co., Advanced Book Program, 1984.

37 Boris Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series 2*, 1968. English translation by G. Ryabov available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

The Domino Problem is Undecidable on Surface Groups

Nathalie Aubrun
LIP, ENS de Lyon – CNRS – UCBL – Université de Lyon, France
nathalie.aubrun@ens-lyon.fr

Sebastián Barbieri
University of British Columbia, Vancouver, Canada
sbarbieri@math.ubc.ca

Etienne Moutot
LIP, ENS de Lyon – CNRS – UCBL – Université de Lyon, France
University of Turku, Finland
etienne.moutot@ens-lyon.fr

Abstract

We show that the domino problem is undecidable on orbit graphs of non-deterministic substitutions which satisfy a technical property. As an application, we prove that the domino problem is undecidable for the fundamental group of any closed orientable surface of genus at least 2.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases tilings, substitutions, SFTs, decidability, domino problem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.46

Related Version `https://arxiv.org/abs/1811.08420`

Funding This work was partially supported by the ANR project CoCoGro (ANR-16-CE40-0005).

1 Introduction

Initially studied by Wang [22], the domino problem is the algorithmic question of determining if a finite set of unit square tiles with colored edges –called Wang tiles– can be used to tile the plane in such a way that adjacent edges of neighbor Wang tiles have the same color. It was originally conjectured by Wang that the domino problem was decidable. However, Berger [5] and later Robinson [21] both combined their constructions of an aperiodic set of Wang tiles and a reduction from the halting problem of Turing machines to show that the domino problem was in fact undecidable.

The domino problem can be naturally extended to a much broader context. Let Γ be a labeled directed infinite graph, $\mathcal{A}$ a finite set, and $F = \{p_1, \dots, p_n\}$ a finite list of colorings p_i of vertices of finite connected subgraphs of Γ by $\mathcal{A}$. The domino problem of Γ is the language of all codings of pairs $(\mathcal{A}, F)$ as above, for which there exists a coloring of the vertices of Γ such that none of the $p_i \in F$ embed as a colored labeled subgraph. Naturally, Wang's domino problem can be reinterpreted in this setting by letting Γ be the bi-infinite square grid, $\mathcal{A}$ the set of Wang tiles, and F the list of all horizontal or vertical pairs of tiles whose colors do not match.

A particularly interesting case is when Γ is a labeled directed Cayley graph of a finitely generated group G. In this case, there is a direct correspondence between colorings of the vertices of Γ by $\mathcal{A}$ which avoid a list of forbidden colored subgraphs as described above, and subshifts of finite type (SFT), that is, closed and translation invariant subsets of $\mathcal{A}^G$ which are determined by a finite list of forbidden patterns. What is more, it can be shown that the domino problems of all such Cayley graphs of G are computationally many-one

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 46; pp. 46:1–46:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

equivalent, and thus one may speak about the domino problem of G. In the particular case when $G = \mathbb{Z}$, the domino problem is decidable: every $\mathbb{Z}$-SFT can be represented by a labeled finite graph [18], and the existence of a configuration in the SFT (i.e. a bi-infinite word) is equivalent to the existence of a cycle in the graph. The case of $\mathbb{Z}^2$ coincides with the formalism of Wang tiles, and is thus undecidable.

The domino problem on graphs other than the Cayley graphs of $\mathbb{Z}$ and $\mathbb{Z}^2$ has been studied by several authors. The undecidability for a graph which models the hyperbolic plane was settled by Kari [15], and can also be obtained from the construction of a hierarchical aperiodic tiling on the hyperbolic plane by Goodman-Strauss [12] and by Margenstern [19]. There has also been research in the case of graphs which can be obtained by self-similar substitutions [4]. For finitely generated groups, the only groups where the domino problem is known to be decidable are virtually free groups, and it is even conjectured that they are the only ones [3].

▶ **Conjecture 1.** *A finitely generated group has decidable domino problem if and only if it is virtually free.*

An interesting take on this conjecture comes from the fact that the domino problem can be expressed in MSO logic [22, 2]. The MSO logic over the Cayley graph of a finitely generated group is decidable if and only if the group is context-free [17], and a group is virtually-free if and only if it is context-free and accessible [20]. Stated differently, (using that all finitely presented groups are accessible [11]) groups that are not virtually free have an undecidable MSO logic. Proven true, the domino problem conjecture would show that the domino problem fragment is "big enough" in MSO. Recent results support the conjecture: decidability of the domino problem is a quasi-isometry invariant for finitely presented groups [10] – i.e. a geometric property of the group – and the conjecture holds true for Baumslag-Solitar groups [1], polycyclic groups [13] and groups of the form $G_1 \times G_2$ [14]. A survey on the domino problem for finitely generated groups can be found in [6, Chapter 9].

The results of Aubrun and Kari [15, 1] share a common factor: the domino problem is shown to be undecidable on two specific structures that grow exponentially with integer or rational base. But what if the structure grows with a different base? The technique employed by those authors –reduction from the immortality problem of rational piecewise affine maps– seems difficult to adapt in this case. A class of structures which can grow non-regularly is given by orbit graphs of non-deterministic substitutions (Section 3). This class of structures includes the hyperbolic plane model of [15], which can be thought of as an orbit graph of the one-letter substitution $0 \mapsto 00$. Using a technique involving the superposition of two orbit graphs of substitutions, presented in [9], we show that the domino problem is undecidable on all orbit graphs of non-deterministic substitutions that satisfy a technical property (Section 4). As an application of the previous result we show that the domino problem of the fundamental group of any closed orientable surface of genus at least 2 is undecidable (Section 5). Finally, we discuss the case of word-hyperbolic groups (Section 6) and show that if a famous conjecture of Gromov –or a weaker version– holds, then the only word-hyperbolic groups with decidable domino problem are the virtually free groups, hence confirming the conjecture for this class of groups.

2 Subshifts on graphs and the domino problem

We define a *graph* Γ to be a triple $(V_\Gamma, E_\Gamma, L_\Gamma)$ where V_Γ is an infinite countable set of vertices, $E_\Gamma \subset V_\Gamma^2$ is the set of edges, such that $|\{u \in V_\Gamma \mid (u,v) \in E_\Gamma \text{ or } (v,u) \in E_\Gamma\}| < M$ for every vertex $v \in V_\Gamma$, where M is some constant, and $L_\Gamma \colon E_\Gamma \to L$ is a labeling function which assigns to every edge a label in a finite set L. Important examples of such graphs are Cayley graphs of finitely generated groups. More precisely, given a finitely generated group G and a finite set of generators $\mathcal{S}$, its Cayley graph is given by $V_\Gamma = G$, $E_\Gamma = \{(g, gs) \mid g \in G, s \in \mathcal{S}\}$, $L_\Gamma((g, gs)) = s$. Let $\Gamma = (V_\Gamma, E_\Gamma, L_\Gamma)$ be a graph as defined above. Let S, T be two finite subsets of V_Γ. A mapping $\phi\colon S \to T$ is a *label preserving graph isomorphism* if ϕ is a bijection and

- for all $u, v \in S$, $(u, v) \in E_\Gamma$ if and only if $(\phi(u), \phi(v)) \in E_\Gamma$;
- for all $u, v \in S$, $L_\Gamma((u, v)) = L_\Gamma((\phi(u), \phi(v)))$.

Let $\mathcal{A}$ be a finite alphabet and Γ a graph. The set of mappings from V_Γ to $\mathcal{A}$, denoted $\mathcal{A}^\Gamma$, is the set of *configurations* over Γ. Endowed with the prodiscrete topology, the set $\mathcal{A}^\Gamma$ is compact and metrizable. If $S \subset V_\Gamma$ is a finite and connected set of vertices, a *pattern with support* S is a mapping $p\colon S \to \mathcal{A}$. A pattern $p\colon S \to \mathcal{A}$ *appears* in a configuration $x \in \mathcal{A}^G$ (resp. in a pattern $p'\colon S' \to \mathcal{A}$) if there exists a finite set of vertices $T \subset V_\Gamma$ (resp. $T \subset S'$) and a label preserving graph isomorphism $\phi\colon S \to T$ such that $p_u = x_{\phi(u)}$ (resp. $p_u = p'_{\phi(u)}$) for every $u \in S$. In this case, we denote $p \sqsubset x$ (resp. $p \sqsubset p'$).

A *subshift* $X_F \subset \mathcal{A}^\Gamma$ is a set of configurations that avoid some set of forbidden patterns F, i.e. $X_F := \left\{x \in \mathcal{A}^\Gamma \mid \text{ no pattern of } F \text{ appears in } x\right\}$. This notion extends the classical definition of subshift for group actions to arbitrary graphs. A *subshift of finite type* (SFT) is a subshift for which F can be chosen finite – equivalently, an SFT may also be defined by a finite set of allowed patterns. In the case where the support of all the forbidden patterns in F consist of two vertices connected by an edge, we say X_F is a *nearest neighbor subshift*.

Given a graph Γ and a finite alphabet $\mathcal{A}$, a pattern as defined above can be encoded by a finite graph, which is an induced finite subgraph of Γ, with labels on edges and letters from $\mathcal{A}$ on vertices. This is what is meant in the sequel by *coding* of a pattern.

Let Γ be a graph in the previous sense. The *domino problem for* Γ is defined as the set $\texttt{DP}(\Gamma)$ of codings of finite sets of forbidden patterns F such that $X_F \neq \emptyset$. If the set $\texttt{DP}(\Gamma)$ is recursive, we say that Γ *has decidable domino problem.*

3 Substitutions, orbits and tilings

Inspired by [9], we associate a tiling of $\mathbb{R}^2$ to the orbit of an infinite word $w \in \mathcal{A}^\mathbb{Z}$ under the action of a substitution, in which every tile codes a production rule of the substitution.

3.1 Substitution systems

We first define parent functions, which will be used to give precise descriptions of orbits of infinite words under the action of a substitution. A *parent function* $P\colon \mathbb{Z} \to \mathbb{Z}$ is an onto and non-decreasing function. In particular, such a function P satisfies that for every $i \in \mathbb{Z}$, $P(i+1) - P(i) \in \{0, 1\}$.

A *non-deterministic substitution* is a couple $(\mathcal{A}, R)$ where $\mathcal{A}$ is a finite alphabet and $R \subset \mathcal{A} \times \mathcal{A}^*$ is a finite set called the *relation*, and whose elements are called *production rules*. We say that an infinite word $\omega \in \mathcal{A}^\mathbb{Z}$ *produces* the word $\omega' \in \mathcal{A}^\mathbb{Z}$ with respect to the parent function P if for every $i \in \mathbb{Z}$, one has $(\omega_i, \omega'_{|P^{-1}(i)}) \in R$, where $\omega'_{|P^{-1}(i)}$ is the finite subword of ω' that appears on indices $\{j \in \mathbb{Z} \mid P(j) = i\}$. In this case, by abuse of notation, we denote $(\omega, \omega') \in R$. An *orbit* of a non-deterministic substitution $(\mathcal{A}, R)$ is a set $\left\{(\omega^i, P_i)\right\}_{i \in \mathbb{Z}} \in \left(\mathcal{A}^\mathbb{Z} \times \mathbb{Z}^\mathbb{Z}\right)^\mathbb{Z}$ such that for every $i \in \mathbb{Z}$, P_i is a parent function, and the word ω^i produces the word ω^{i+1} with respect to P_i. A *deterministic substitution* (or substitution for short) is a non-deterministic substitution where the relation is a function. A non-deterministic substitution $(\mathcal{A}, R)$ *has an expanding eigenvalue* if there exist $\lambda > 1$ and $v\colon \mathcal{A} \to \mathbb{R}_{>0}$ such that for every $(a, w) \in R$ if we write $w = w_1 w_2 \dots w_{|w|}$ we have,

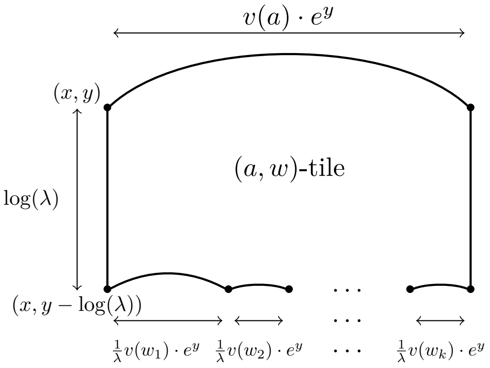
$$\lambda \cdot v(a) = \sum_{i=1}^{|w|} v(w_i).$$

▶ **Example 2.** The substitution given by the production rule $0 \mapsto 00$ has an expanding eigenvalue. This can be verified by choosing $\lambda = 2$ and $v(0) = 1$.

▶ **Example 3.** The substitution $\sigma_{\texttt{gold}}$ given by the production rules $a \mapsto aab$, $b \mapsto ba$ has an expanding eigenvalue. This can checked with $\lambda = \frac{3+\sqrt{5}}{2}$ and $v(a) = \frac{1+\sqrt{5}}{2}$, $v(b) = 1$.

3.2 Orbits as tilings of $\mathbb{R}^2$

Let $(\mathcal{A}, R)$ be a non-deterministic substitution with an expanding eigenvalue $\lambda > 1$ and $v\colon \mathcal{A} \to \mathbb{R}_{>0}$. For every production rule $(a, w) \in R$, define the (a, w)*-tile in position* $(x, y) \in \mathbb{R}^2$ as the square polygon with $|w| + 3$ edges pictured below, where $w = w_1 \dots w_k$ (horizontal edges are curved to be more visible, but are in fact just straight lines).


$v(a) \cdot e^y$
(x, y)
(a, w)-tile
$\log(\lambda)$
$(x, y - \log(\lambda))$
. . .
$\frac{1}{\lambda}v(w_1) \cdot e^y$
$\frac{1}{\lambda}v(w_2) \cdot e^y$
$\frac{1}{\lambda}v(w_k) \cdot e^y$

▶ Remark 4. The length of the top edge and the sum of lengths of bottom edges of this tile are the same. Since $(\mathcal{A}, R)$ has an expanding eigenvalue $\lambda > 1$ with v, one has

$$\sum_{j=1}^{k} \frac{1}{\lambda} v(w_i) \cdot e^y = \frac{e^y}{\lambda} \cdot \lambda \cdot v(a) = v(a) \cdot e^y,$$

so that the bottom right vertex $(x + \frac{1}{\lambda}(v(w_1) + \dots + v(w_k))\, e^y, y - \log(\lambda))$ is indeed $(x + v(a) \cdot e^y, y - \log(\lambda))$.

The $(\mathcal{A}, R)$*-tiles* is the set of all (a, w)-tiles in position (x, y) for all possible $(a, w) \in R$ and $(x, y) \in \mathbb{R}^2$. A *tiling of* $\mathbb{R}^2$ *with* $(\mathcal{A}, R)$*-tiles*, or $(\mathcal{A}, R)$-tiling for short, is a countable collection of $(\mathcal{A}, R)$-tiles that covers $\mathbb{R}^2$ and have pairwise disjoint interiors, such that tiles are edge-to-edge –the intersection of two tiles is either empty or a full edge and two vertices. In Figure 2 we illustrate a $\sigma_{\texttt{gold}}$-tiling (in blue) and a $0 \mapsto 00$-tiling (in grey).

▶ **Proposition 5.** *If a substitution* $(\mathcal{A}, R)$ *with an expanding eigenvalue admits orbits, then there exists a tiling of* $\mathbb{R}^2$ *with* $(\mathcal{A}, R)$*-tiles.*

4 Undecidability of the domino problem on orbit graphs

Let $u = (u_i)_{i\in\mathbb{Z}}$ be a bi-infinite sequence of positive integers. The *accumulation function of* u is the function $\Delta\colon \mathbb{Z} \to \mathbb{Z}$ given by

$$\Delta(i) = \begin{cases} \sum_{k=0}^{i-1} u_k & \text{if } i \geq 1 \\ 0 & \text{if } i = 0 \\ -\sum_{k=i}^{-1} u_k & \text{if } i \leq -1 \end{cases}.$$

Note that the family of discrete intervals $(I_k)_{k\in\mathbb{Z}}$ where $I_k = [\Delta(k); \Delta(k+1) - 1]$ forms a partition of $\mathbb{Z}$. If P is a parent function, and if we define the sequence u by $u_i = |P^{-1}(i)|$ for every $i \in \mathbb{Z}$, then we get that $P(j) = i$ for every $j \in [\Delta(i); \Delta(i+1) - 1]$, where Δ is the accumulation function of u.

Let $(\mathcal{A}, R)$ be a non-deterministic substitution. Denote $M = \max_{(a,w)\in R} |w|$.

▶ **Definition 6.** *The* orbit graph *associated with the orbit* $\Omega = \left\{(\omega^i, P_i)\right\}_{i\in\mathbb{Z}}$ *of* $(\mathcal{A}, R)$ *is the graph* Γ_Ω *with set of vertices* $\mathbb{Z}^2$*, edges* E_Ω *and labeling function* $L_\Omega\colon E_\Omega \to \{\texttt{next}\} \cup [0; M-1]$ *given by*

- *for every* $i, j \in \mathbb{Z}$, $((i,j),(i,j+1)) \in E_\Omega$ *and* $L_\Omega(((i,j),(i,j+1))) = \texttt{next}$*;*
- *for every* $i \in \mathbb{Z}$ *and every* $k \in [\Delta_{i+1}(j); \Delta_{i+1}(j+1) - 1]$, $((i,j),(i+1,k)) \in E_\Omega$ *and* $L_\Omega(((i,j),(i+1,k))) = k - \Delta_{i+1}(j)$*,*

where Δ_i *is the accumulation function associated with* $\left(|P_i^{-1}(j)|\right)_{j\in\mathbb{Z}}$ *for every* $i \in \mathbb{Z}$*.*

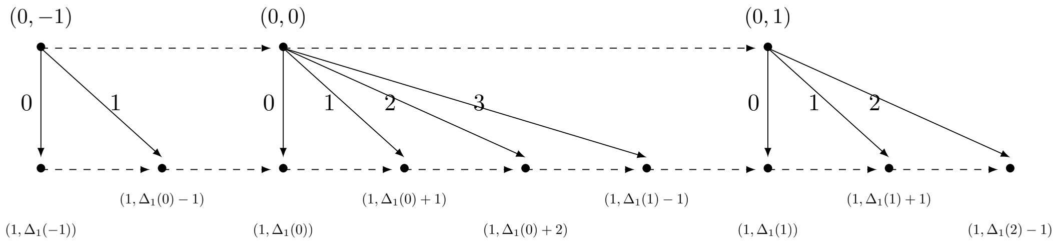


Figure 1 Part of an orbit graph. Dashed arrow are edges of the graph labeled with `next`.

In this formalism, Kari's result for the hyperbolic plane [15] is equivalent to the statement that all orbit graphs of the one-letter substitution $0 \mapsto 00$ have undecidable domino problem.

▶ **Theorem 7** (Kari [15])**.** *For all orbit graphs of the substitution* $(\{0\}, 0 \mapsto 00)$ *the domino problem is undecidable.*

The goal of this section is to show that the domino problem of any orbit graph associated to an orbit of a non-deterministic substitution $(\mathcal{A}, R)$ with an expanding eigenvalue λ is undecidable. The general idea is to show that, given any SFT over an orbit graph of $0 \mapsto 00$, it is possible to encode it in an orbit graph of $(\mathcal{A}, R)$. We do this in two steps. First (Section 4.1), we show a variation of the "Technical Lemma" of Cohen and Goodman-Strauss [9], where we prove that it is possible to encode the structure of orbit graphs of $0 \mapsto 00$ in an SFT over $(\mathcal{A}, R)$. Then (Section 4.2), we prove that in addition to the structure of its orbit graph, it is also possible to encode any SFT over orbit graphs of $0 \mapsto 00$ in an SFT over $(\mathcal{A}, R)$.

4.1 Superposition of orbits

Let us fix a non-deterministic substitution $(\mathcal{A}, R)$ with an expanding eigenvalue $\lambda > 2$ – this latter assumption ensures that letters of an alphabet $\mathcal{B}$ described below are non-degenerate, and will be suppressed later. Without loss of generality, we may choose the function $v\colon \mathcal{A} \to \mathbb{R}_{>0}$ associated to λ such that $v(a) > 4$ for each $a \in \mathcal{A}$.

Let Ω be an orbit of $(\mathcal{A}, R)$. We shall construct a finite alphabet $\mathcal{B}$ and a finite set of forbidden patterns F such that the subshift $Y \subset \mathcal{B}^{\Gamma_\Omega}$ over orbit graph Γ_Ω, defined by the set of forbidden patterns F has the following properties:

1. Y is non-empty,
2. every configuration $y \in Y$ encodes an orbit graph of the substitution $(\{0\}, 0 \mapsto 00)$.

Consider an orbit Ω of $(\mathcal{A}, R)$ and Φ an orbit of $(\{0\}, 0 \mapsto 00)$. By Proposition 5 both of these orbits can be realized as tilings of $\mathbb{R}^2$. Symbols from $\mathcal{B}$ will encode non-empty finite regions of the tiling with $(\{0\}, 0 \mapsto 00)$ that are witnessed by $(\mathcal{A}, R)$-tiles. These regions will be chosen in such a way that their union recovers the whole tiling and they are pairwise disjoint (see Figure 2).

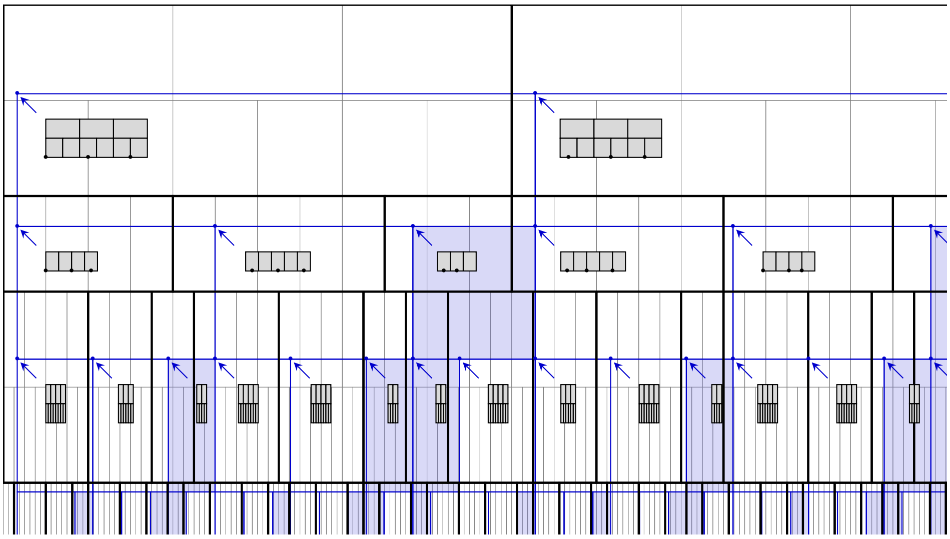
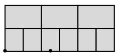
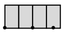


Figure 2 Γ_Φ (in black) superimposed over an orbit graph of $(\mathcal{A}, R)$ (in blue) leads to the construction of the subshift $Y \subset \mathcal{B}^{\Gamma_\Omega}$: letters of the alphabet $\mathcal{B}$ are blocks of the orbit graph Γ_Φ, such as  or  .A careful examination shows that dimensions of blocks in $\mathcal{B}$ are bounded: if (t, h) denotes the width and height of a block in $\mathcal{B}$, we get that $\min_{a \in \mathcal{A}} \lfloor \frac{v(a)}{4} \rfloor \leq t \leq \max_{a \in \mathcal{A}} \lfloor 1 + \frac{v(a)}{2} \rfloor$ and $\frac{\log(\lambda)}{\log(2)} - 1 < h \leq \frac{\log(\lambda)}{\log(2)} + 1$. Thus $\mathcal{B}$ is finite, and the assumption $\lambda > 2$ and $v(a) > 4$ ensures that blocks have non-degenerate dimensions.

The set of forbidden patterns F is the finite set of patterns that do not correspond to a valid encoding of the orbit graph of $(\{0\}, 0 \mapsto 00)$. In other words, we allow only patterns in which the finite regions of $(\{0\}, 0 \mapsto 00)$ that are encoded are consistent from one neighbor to one other. We thus obtain an SFT Y on Γ_Ω which encodes Φ and is non-empty.

▶ **Lemma 8.** *For every orbit Ω of $(\mathcal{A}, R)$ the subshift of finite type $Y \subset \mathcal{B}^{\Gamma_\Omega}$ is non-empty.*

4.2 Simulation of SFTs over of $0 \mapsto 00$ on orbit graphs of $(\mathcal{A}, R)$

Let Ω and Φ be orbits of $(\mathcal{A}, R)$ and $(\{0\}, 0 \mapsto 00)$ respectively, and Γ_Ω, Γ_Φ be orbit graphs of Ω and Φ respectively. Let Σ be a finite alphabet and F_Σ a set of nearest neighbor forbidden patterns on Γ_Φ over the alphabet Σ. We denote by X_Σ the SFT defined by forbidden patterns F_Σ. In order to encode X_Σ into Γ_Ω, we use the same method as above for Y, but enrich the patterns of $0 \mapsto 00$ with colors taken from Σ. We then construct forbidden patterns that ensure that no forbidden patterns from F_Σ appear.

More formally, we define $\mathcal{B}_\Sigma$ as the set of pairs (b, p_b) such that $b \in \mathcal{B}$ and $p_b \colon \Gamma_b \to \Sigma$ is a pattern. For a pattern p on Γ_Ω with alphabet B_Σ denote by $\pi_{\mathcal{B}}(p)$ the restriction to the first coordinate of $\mathcal{B}_\Sigma$. And denote by $q(p) \colon \Gamma_{\pi_{\mathcal{B}}(p)} \to \Sigma$ the pattern over $(\{0\}, 0 \mapsto 00)$ whose support is the graph $\Gamma_{\pi_{\mathcal{B}}(p)}$ and is obtained by pasting together the corresponding patterns p_b on the second coordinate of B_Σ.

Define $F_{\mathcal{B},\Sigma}$ as the set of all patterns p over the alphabet $\mathcal{B}_\Sigma$ which have supports which consist in three vertices $\{u, v, w\}$ in Γ_Ω such that $(u, v), (u, w)$ are edges, $L((u, v)) = \texttt{next}$ and $L((u, w)) = \ell$ for some ℓ appearing in the parent matching labels of the orbit graph Γ_Ω, and that satisfy one of the following two properties:

1. The pattern $\pi_{\mathcal{B}}(p)$ obtained by restricting p to the first coordinate of $\mathcal{B}_\Sigma$ is in F;
2. The pattern $q(p)$ obtained by pasting the patterns of p described by the second coordinate of $\mathcal{B}_\Sigma$ contains a forbidden pattern from F_Σ.

Clearly $F_{\mathcal{B},\Sigma}$ has finitely many patterns (up to label preserving graph isomorphism). For any orbit Ω of $(\mathcal{A}, R)$ we define the subshift of finite type $Y_\Sigma \subset (B_\Sigma)^{\Gamma_\Omega}$ as the set of all colorings of Γ_Ω by B_Σ where no pattern from $F_{\mathcal{B},\Sigma}$ appears.

This construction leads to the following Lemma, expressing the fact that X_Σ is indeed encoded into Y_Σ.

▶ **Lemma 9.** *Let X_Σ be the subshift on Γ_Φ with alphabet Σ defined by the nearest neighbor forbidden patterns F_Σ and let $Y_\Sigma \subset (B_\Sigma)^{\Gamma_\Omega}$ be defined as above. Then $Y_\Sigma = \emptyset$ if and only if $X_\Sigma = \emptyset$.*

▶ Remark 10. The alphabet B_Σ and the set of forbidden patterns $F_{\mathcal{B},\Sigma}$ which define Y_Σ only depend upon Σ, F_Σ and the substitution $(\mathcal{A}, R)$, and not on the choice of the orbit Ω of $(\mathcal{A}, R)$.

▶ **Theorem 11.** *The domino problem is undecidable on any orbit graph of a non-deterministic substitution with an expanding eigenvalue.*

Proof. Let us first assume that the expanding eigenvalue λ associated to $(\mathcal{A}, R)$ satisfies $\lambda > 2$. Let Σ and F_Σ be respectively an alphabet and a nearest neighbor set of forbidden patterns for an orbit graph Γ_Φ of an orbit Φ of $(\{0\}, 0 \mapsto 00)$ which define a nearest neighbor SFT X_Σ. By Lemma 9 we know that $X_\Sigma = \emptyset$ if and only if $Y_\Sigma = \emptyset$. Furthermore, we claim that the alphabet and set of forbidden patterns which define Y_Σ can be constructed effectively from Σ and F_Σ. Indeed, the subshift Y does not depend upon Σ and thus its alphabet $\mathcal{B}$ and forbidden patterns F can be hard-coded in the algorithm. It is easy to see that from $\mathcal{B}$ one can effectively construct the alphabet B_Σ and the forbidden patterns $F_{\mathcal{B},\Sigma}$ which define Y_Σ.

These two facts together show that if $\texttt{DP}(\Gamma_\Omega)$ is decidable and $\lambda > 2$, then so is $\texttt{DP}(\Gamma_\Phi)$. Using the result of Kari (Theorem 7) we have that $\texttt{DP}(\Gamma_\Phi)$ is undecidable, hence $\texttt{DP}(\Gamma_\Omega)$ is also undecidable.

For the case where $1 < \lambda \leq 2$, we consider the relation R^m defined recursively by:

- $R^1 = R$.

- R^{k+1} is the set of all pairs $(a, (c^1_1 \dots c^1_{\ell_1})(c^2_1 \dots c^2_{\ell_2}) \dots (c^1_k \dots c^1_{\ell_k}))$ in $\mathcal{A} \times \mathcal{A}^*$ for which there is a pair $(a, b_1 \dots b_k) \in R^k$ such that $(b_i, c^i_1 \dots c^i_{\ell_i}) \in R$ for each $i \in \{1, \dots, k\}$.

In other words, R^m is the set of all relations that can be obtained by starting with a symbol $a \in \mathcal{A}$ and replacing m times each letter by the right hand side of a production rule of R. Let $n \in \mathbb{N}$ such that $\lambda^n > 2$ and note that the substitution $(\mathcal{A}, R^n)$ has the expanding eigenvalue $\lambda^n > 2$. Lemma 9 then provides an encoding of the substitution $(\{0\}, 0 \mapsto 00)$ on orbit graphs of $(\mathcal{A}, R^k)$ for any $k \in \{0, \dots, n-1\}$, in the form of SFTs $Y^{n,k}_\Sigma$ such that $Y^{n,k}_\Sigma = \emptyset$ if and only if $X_\Sigma = \emptyset$. Then we are able to build an SFT Z on Γ_Ω which encodes a copy of $Y^{k,n}_\Sigma$ for each $k \in \{0, \dots, n-1\}$. One can verify that $Z = \emptyset$ if and only if $X_\Sigma = \emptyset$, leading to the same reduction as in the case $\lambda > 2$. ◀

5 The domino problem for surface groups

A fundamental result of geometry is that up to homeomorphism, closed orientable surfaces are completely classified by their genus g: any such surface is either homeomorphic to a sphere or to a finite connected sum of tori. In this section we shall classify the domino problem of their fundamental groups.

5.1 Surface groups

The *surface group of genus* g is the group defined by the following presentation:

$$G_g = \langle a_1, b_1, \dots, a_g, b_g \mid [a_1, b_1] \dots [a_g, b_g] \rangle,$$

where $[a, b] = aba^{-1}b^{-1}$ is the commutator of a and b. It is interesting to notice that $\mathbb{Z}^2 = \langle a, b \mid aba^{-1}b^{-1} \rangle$ is the surface group of genus 1, and hence by Berger's result [5] its associated domino problem is undecidable.

The domino problem for a finitely generated group is known to be a commensurability invariant [6, Corollary 9.53]. It turns out that all surface groups of genus $g \geq 2$ are commensurable [8, Proposition 6.7]. By combining these two facts, it would be enough to prove the undecidability of the domino problem for just the surface group of genus 2. In the sequel, we shall denote by G the surface group of genus 2, i.e. the group with finite presentation

$$G = \langle a, b, c, d \mid [a, b][c, d] \rangle,$$

denote by S the symmetric closure of its generating set $\{a, b, c, d\}$, and by 1_G its identity.

The Cayley graph of G associated with the presentation above is not an orbit graph of some substitution with an expanding eigenvalue, but can be seen as such just by assigning different labels to the edges. Moreover we shall see that these labels can be obtained locally, which means that we can code the relabeling inside an SFT.

5.2 Finding a substitution in the surface group of genus 2

The goal of this section is to establish a parallel between the Cayley graph of the surface group $\mathcal{C}_G := \Gamma(G, S)$ and the orbit graph of a particular substitution.

The group G has only one relation $[a, b][c, d] = 1_G$. Thus the only minimal cycles of the Cayley graph are cyclic permutations of $[a, b][c, d]$. We call them *elementary cycles*. Moreover, any edge in the Cayley graph is part of at least one elementary cycle, since all generators and their inverses appear in the relation. Let $d(g, h)$ be the smallest number of elementary cycles that must be crossed to go from g to h in $\mathcal{C}_G$. Let $B_i = \{g \in G \mid d(1_G, g) \leq i\}$ be the ball of radius i and $C_i = \{g \in G \mid d(1_G, g) = i\}$ be the sphere of radius i, so that $B_{i+1} \setminus B_i = C_{i+1}$.

Consider an element $g \in C_i$ for $i \geq 1$. There are exactly two elements $s \in S$ such that $gs \in C_i$. There can be either ($\mathtt{a}$) one or ($\mathtt{b}$) none $s \in S$ so that $gs \in C_{i-1}$. We must therefore have that there are 5 and 6 values $s \in S$ such that $gs \in C_{i+1}$ for types ($\mathtt{a}$) and ($\mathtt{b}$) respectively. More precisely, it can be verified that the sequence of elements of C_{i+1} that is obtained by following an elementary cycle from an element of type ($\mathtt{a}$) in C_i has the type sequence $\mathtt{ab}^5\mathtt{ab}^5\mathtt{ab}^5\mathtt{ab}^5\mathtt{ab}^4$ and the sequence of types for an element of type ($\mathtt{b}$) is $\mathtt{ab}^5\mathtt{ab}^5\mathtt{ab}^5\mathtt{ab}^5\mathtt{ab}^5\mathtt{ab}^4$.

This leads us to define the substitution $s\colon \{\mathtt{a}, \mathtt{b}\} \to \{\mathtt{a}, \mathtt{b}\}^*$ given by

$$\begin{cases} s(\mathtt{a}) = (\mathtt{ab}^5)^4\mathtt{ab}^4 \\ s(\mathtt{b}) = (\mathtt{ab}^5)^5\mathtt{ab}^4. \end{cases}$$

From now on, we fix $\Omega = \left(\omega^i, P_i\right)_{i\in\mathbb{Z}}$ an orbit of the substitution s defined above, and denote by Γ its associated orbit graph. Let us note that s admits an expanding eigenvalue ($\lambda = 17 + 12\sqrt{2}$ and $v(\mathtt{b})/v(\mathtt{a}) = \frac{1+\sqrt{2}}{2}$).

The similarities between the two graphs will allow us to perform a reduction from the domino problem on Γ (shown to be undecidable in Section 4) to the domino problem on the surface group of genus 2. In order to do this reduction, all we need is a computable map which sends sets of pattern codings over Γ into sets of pattern codings over $\mathcal{C}_G$ such that the sets defining a non-empty subshift are mapped into sets defining a non-empty subshift and vice-versa. This is not trivial because some of the edges are lost going from Γ to $\mathcal{C}_G$. In order to recover those edges, we shall construct an SFT X over G which locally recovers the lost information and use it to build the bijection needed for the reduction. Note that technically we do not need an SFT to do so, a computable bijection would be enough. However doing it with an SFT provides a locally computable mapping, which is a nice bonus.

Definition of X

To define the SFT X, we introduce a notion of directions that corresponds to following edges of the orbit graph. These directions depend on the element of the group we consider, but can nevertheless be defined by local rules. The alphabet of X contains the correspondence between generators and local directions. More formally, we first consider the general alphabet $\mathcal{A}_0$, consisting of the tuples $(c, (h_1, d_1), (h_2, d_2), \ldots, (h_8, d_8))$ such that

- $c \in \{\blacksquare, \square\}$ is the color of the cell,
- $(h_1, \ldots, h_8)$ is a permutation of $S \cup S^{-1} = \{a, a^{-1}, b, b^{-1}, c, c^{-1}, d,^{-1}\}$,
- $d_1, \ldots, d_8 \in \{\leftarrow, \rightarrow, \uparrow, \downarrow_1, \downarrow_2, \downarrow_3, \downarrow_4, \downarrow_5, \downarrow_6\}$ the directions associated to each generator.

Let $x \in \mathcal{A}_0^G$ be a configuration over $\mathcal{A}_0$. For every $g \in G$, if the first coordinate of x_g is $c = \blacksquare$ (resp. $\square$), we call x_g a black (resp. white) cell.

The alphabet $\mathcal{A}_1 \subseteq \mathcal{A}_0$ is made of three types of elements with more precise directions imposed, depending on the color c:

$$(\blacksquare, (h_1, \leftarrow), (h_2, \rightarrow), (h_3, \uparrow), (h_4, \downarrow_1), (h_5, \downarrow_2), (h_6, \downarrow_3), (h_7, \downarrow_4), (h_8, \downarrow_5))$$

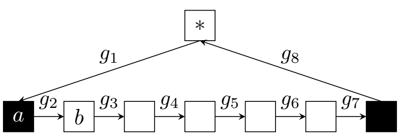
$$(\square, (h_1, \leftarrow), (h_2, \rightarrow), (h_3, \downarrow_1), (h_4, \downarrow_2), (h_5, \downarrow_3), (h_6, \downarrow_4), (h_7, \downarrow_5), (h_8, \downarrow_6))$$

Black cells have directions *left*, *right*, *up* and *down*, whereas whites ones have only *left*, *right* and *down*. Note that for black and white cells, *up*, *left* and *right* are unique. We can then define their top, left and right neighbors.

▶ **Definition 12.** *Let $x \in \mathcal{A}_1^G$ be a configuration over $\mathcal{A}_1$ and $g \in G$. We define:*

- *gh_1 the* left neighbor of *g in x, denoted by $\leftarrow_x(g)$,*
- *gh_2 is the* right neighbor of *g in x, denoted by $\rightarrow_x(g)$,*
- *if x_g is a black cell, gh_3 is the* top neighbor of *g in x, denoted by $\uparrow_x(g)$,*
- *for $i \in \{1, ..., 5\}$, gh_{3+i} (for a white cell, $i \in \{1, ..., 6\}$, gh_{2+i}) is the* i-th bottom neighbor of *g in x, denoted by $\downarrow_{i,x}(g)$.*

Using local rules, we forbid elementary cycles that do not have the colors shown on Figure 3. We also impose the orientations to be as drawn. For example, the right of $\colorbox{black}{$\textcolor{white}{a}$}$ is g_2, its top is g_1^{-1}, and the other directions of a are not constrained by this cycle. Similarly, the left of $\boxed{b}$ is g_2^{-1}, its right g_3 and other directions unconstrained. To do so, we call $\mathcal{F}_1$ the set of all elementary cycles that are not of the form of Figure 3.



🟨 **Figure 3** The two possible types of colorings of cycles. There are no color constraints on $\boxed{*}$, and the cycle $g_1 \dots g_8$ is any cyclic permutation of $[a, b][c, d]$.

We add the constraint that directions must be consistent between adjacent cells, by forbidding the finite set $\mathcal{F}_2$, which is the set of patterns on the support $\{1_G, h\}$ for $h \in G$, such that x_{1_G} and x_h are linked by mismatching directions. That is,

$$\mathcal{F}_2 = \left\{ \text{pattern } p \text{ of support}\{1_G, h\} \;\middle|\; \begin{array}{r} \leftarrow_p(1_G) = h \text{ and } \rightarrow_p(h) \neq 1_G \text{ or} \\ \rightarrow_p(1_G) = h \text{ and } \leftarrow_p(h) \neq 1_G \text{ or} \\ \uparrow_p(1_G) = h \text{ and } \forall i, \downarrow_{i,p}(h) \neq 1_G \text{ or} \\ \exists i, \downarrow_{i,p}(1_G) = h \text{ and } \uparrow_p(h) \neq 1_G \end{array} \right\}.$$

We define X as the set of all configurations over $\mathcal{A}_1$ where no forbidden patterns from $\mathcal{F}_1 \cup \mathcal{F}_2$ appear. By definition, X is an SFT.

The subshift of finite type X

We construct a configuration x in the SFT $X \subset \mathcal{A}_1^G$ as the limit of a sequence of configurations $(y_n)_{n \in \mathbb{N}}$ of another SFT $X_2 \subset (\mathcal{A}_1 \cup \{\texttt{orange}\})^G$, where

$$\texttt{orange} := \left(🟧, (a, \downarrow_1), (a^{-1}, \downarrow_2), (b, \downarrow_3), (b^{-1}, \downarrow_4), (c, \downarrow_5), (c^{-1}, \downarrow_6), (d, \downarrow_7), (d^{-1}, \downarrow_8)\right),$$

and we extend the definition of neighbors consistently. X_2 is defined by $\mathcal{F}_1 \cup \mathcal{F}_2$ the same finite set of forbidden patterns as X. Intuitively, because the letter `orange` has only bottom neighbors, an orange cell can only appear once in a configuration of X_2. Moreover, one can build bigger and bigger circles around the orange cell by simply sticking elementary cycles around, colored as on Figure 3, leading to the following lemma.

▶ **Lemma 13.** *For every i, there exists a pattern $p_i \in (\mathcal{A}_1 \cup \{\texttt{orange}\})^{B_i}$ containing no forbidden patterns of $\mathcal{F}$ and such that $(p_i)_g$ is an orange cell if and only if $g = 1_G$.*

From Lemma 13, we can build a configuration y in X_2 that contains only one orange cell at the origin. And from it, we can deduce the non-emptiness of X.

▶ **Proposition 14.** *The subshift X is non-empty.*

Proof. By compactness of $(\mathcal{A}_1 \cup \{\texttt{orange}\})^G$ and Lemma 13, there exists a configuration $y \in X_2$ which coincides with the pattern p_i on B_i for all $i \in \mathbb{N}$. In particular, the orange cell appears only at the origin. The SFT X consists of all configurations on X_2 where the orange tile does not appear. By definition of y, we can find arbitrarily large regions where 🟧 does not appear at all. We can then extract a sequence of configurations $(y_n)_{n\in\mathbb{N}}$ of X_2 such that the orange cell does not appear in $(y_n)_{|B_n}$. Any accumulation point of the sequence $(y_n)_{n\in\mathbb{N}}$ does not contain the orange cell and is thus in X. ◀

If we carefully look at configurations in X, we observe that they are all structured with infinite lines of $\rightarrow$ and $\leftarrow$. Moreover these infinite lines can be ordered with no ambiguity. This is expressed in Lemma 15: any element g of G can be reached from the identity 1_G by first going to the appropriate infinite line, and then moving to the left or right up to g. Fix some $x \in X$, and define $\rightarrow_x^{-1}(g) :=\leftarrow_x(g)$ and $\downarrow_{1,x}^{-1}(g) :=\uparrow_x(g)$.

▶ **Lemma 15.** *For any $g \in G$, there exists i, j such that $g =\rightarrow_x^j \circ \downarrow_{1,x}^i(1_G)$.*

Proof idea. The key idea here is that we can always reorder the operations by taking another way in the graph. Starting from any path from 1_G to g, we can transform it into this "normal form" by taking a longer path that uses only $\downarrow_1$ then $\rightarrow$ operations. ◀

5.2.1 A bijection between $\mathbb{Z}^2$ and the surface group

Let $x \in X$ be fixed. We define $f_x\colon \mathbb{Z}^2 \to G$ by $f_x(i,j) =\rightarrow_x^j \circ \downarrow_{1,x}^i(1_G)$ for every $i, j \in \mathbb{Z}$.

▶ **Lemma 16.** *For every $x \in X$, the function f_x is a bijection.*

Proof idea. It is enough to show that for any group element $g \in G$, there exist uniquely defined $i, j \in \mathbb{Z}$ such that $g =\rightarrow_x^j \circ \downarrow_{1,x}^i(1_G)$. The key ingredient to prove the uniqueness of this representation is to notice that any cycle in the Cayley graph of G contains as many $\uparrow$ as $\downarrow$ operations, which can be proven by induction on the size of the cycle and a careful examination of different cases. ◀

We can moreover prove that f_x also preserves locality, in the sense that neighborhoods are almost preserved, as stated in the following lemma.

▶ **Lemma 17.** *The following equivalences are true:*

1. $\begin{cases} (u,v) \in E_\Gamma \\ L_\Gamma(u,v) = \textit{next} \end{cases} \Leftrightarrow f_x(v) =\rightarrow_x(f_x(u))$
2. $\begin{cases} (u,v) \in E_\Gamma \\ L_\Gamma(u,v) = k \in \{0,\dots,M-1\} \end{cases} \Leftrightarrow f_x(v) =\rightarrow_x^k \circ \downarrow_{1,x}(f_x(u))$

   where M is the number of sons of u.

The bijection f_x itself cannot be a label preserving graph isomorphism, since we lack some edges of Γ in $\mathcal{C}_G$, but it nevertheless enjoys a useful property: if φ is a label preserving graph isomorphism for Γ, then so is $f_x \circ \varphi \circ f_x^{-1}$ for $\mathcal{C}_{G,x}$, and if φ is a label preserving graph isomorphism for $\mathcal{C}_{G,x}$, then so is $f_x^{-1} \circ \varphi \circ f_x$ for Γ, where $\mathcal{C}_{G,x}$ is a relabeling of $\mathcal{C}_g$ according to the configuration x. So roughly speaking, any local pattern is preserved by f_x or by f_x^{-1}.

▶ **Corollary 18.** *Let $\mathcal{A}$ be a finite alphabet. For any configuration $c \in \mathcal{A}^G$, $p \sqsubset c \Rightarrow f_x^{-1}(p) \sqsubset f_x^{-1}(c)$. Conversely for any $d \in \mathcal{A}^\Gamma$, $q \sqsubset d \Rightarrow f_x(q) \sqsubset f_x(d)$.*

5.3 The reduction

We now have everything in hand to prove the undecidability of the domino problem on the surface group of genus 2.

▶ **Theorem 19.** *The domino problem is undecidable on the surface group of genus 2.*

Proof. Recall that Γ is the orbit graph of an orbit of the substitution s defined on page 9. Let $\mathcal{A}$ be a finite alphabet and $Y \subseteq \mathcal{A}^{\Gamma}$ an SFT over Γ, given by a finite set of forbidden patterns $\mathcal{F}_Y$. We define Z the SFT over G with set of forbidden patterns $F_Z := f_x(F_Y)$, where f_x is defined in Lemma 16. We prove that $Z = \emptyset$ if and only if $Y = \emptyset$.

Assume $Z = \emptyset$ and consider a configuration $c \in \mathcal{A}^G$. The configuration $d := f^{-1}(c)$ is thus in $\mathcal{A}^{\Gamma}$. Since $Z = \emptyset$, necessarily c contains a forbidden pattern p from the set $\mathcal{F}_Z$. Since $p \sqsubset c$, Corollary 18 implies that $f_x^{-1}(p) \sqsubset f_x^{-1}(c) = d$. So a pattern $f_x^{-1}(p)$ from $\mathcal{F}_Y$ appears in any configuration $c \in \mathcal{A}^G$, i.e. the subshift Y is empty. Similar arguments show that if Y is empty then so is Z, and the reduction is completed. ◀

▶ **Corollary 20.** *The domino problem is undecidable for every surface group of positive genus.*

Proof. The undecidability of the domino problem is a commensurability invariant [6, Corollary 9.53], and all surface groups of genus $g \geq 2$ are commensurable [8, Proposition 6.7]. By combining these two facts with Theorem 19, we obtain the undecidability of domino problem for surface groups of any genus $g \geq 2$. As the domino problem on $\mathbb{Z}^2$ –the surface group of genus 1– is undecidable, we obtain our result. ◀

6 Remarks about word-hyperbolic groups

Surface groups of genus $g \geq 2$ are special cases of a larger class of groups called *word-hyperbolic*. They can be characterized as the finitely presented groups for which Dehn's algorithm solves the word problem. An important property of the domino problem is that groups which contain subgroups with undecidable domino problem have themselves undecidable domino problem [6, Proposition 9.3.30]. This means that every group which contains an embedded copy of a surface group has undecidable domino problem. This is of special relevance due to the following conjecture by Gromov.

▶ **Conjecture 21** (Gromov). *Every one-ended word-hyperbolic group contains an embedded copy of the surface group of genus* 2*.*

In particular, if Gromov's conjecture holds, every one-ended word-hyperbolic group would automatically have undecidable domino problem. A group can have either $0, 1, 2$ or infinitely many ends. In the case when it has 0 ends it is finite and thus its domino problem is trivially decidable, and whenever it has 2 ends it is virtually $\mathbb{Z}$ and thus it is also decidable. In the case of a finitely presented group G, a fundamental result by Dunwoody [11] shows that if G has infinitely many ends it can be expressed as the fundamental group of a finite graph of groups such that every edge is a finite group and all vertices are either finite or 1-ended. It can also be shown [16] that G is virtually free if and only if all of the vertex groups in its decomposition are finite. Therefore, if G is not virtually free, it must contain a one-ended subgroup. In the case of word-hyperbolic groups, every such group in the decomposition must also be word-hyperbolic [7]. In other words, every word-hyperbolic group which is not virtually free contains a one-ended word-hyperbolic group. This implies the following.

▶ **Proposition 22.** *If Gromov's conjecture holds then the domino problem conjecture holds for all word-hyperbolic groups.*

In fact, we could obtain the same result with an even weaker version of Gromov's conjecture. We say a group G acts *translation-like* on a metric space (X, d) if the action is free and $\sup_{x \in X} d(x, gx) < \infty$ for every $g \in G$. Clearly, if H is a subgroup of G then H acts translation like on any Cayley graph of G by multiplication. A theorem by Jeandel [14] shows that if a finitely presented group H acts translation like on a Cayley graph of a finitely generated group G, then the domino problem of H is many-one reducible to the domino problem on G, in particular, we obtain that any group on which the surface group of genus 2 acts translation-like has undecidable domino problem.

▶ **Proposition 23.** *If every* 1*-ended word-hyperbolic group admits a translation-like action of the surface group of genus* 2*, then the domino problem conjecture holds for all word-hyperbolic groups.*

References

1 Nathalie Aubrun and Jarkko Kari. Tiling Problems on Baumslag-Solitar groups. In *MCU'13*, pages 35–46, 2013.

2 Alexis Ballier and Emmanuel Jeandel. Tilings and model theory. *First Symposium on Cellular Automata Journées Automates Cellulaires.*, 2008.

3 Alexis Ballier and Maya Stein. The domino problem on groups of polynomial growth. *Groups, Geometry, and Dynamics*, 12(1):93–105, March 2018. doi:10.4171/ggd/439.

4 Sebastián Barbieri and Mathieu Sablik. The Domino Problem for Self-similar Structures. In *Pursuit of the Universal*, pages 205–214. Springer International Publishing, 2016. doi:10.1007/978-3-319-40189-8_21.

5 Robert Berger. *The Undecidability of the Domino Problem.* PhD thesis, Harvard University, 1964.

6 Valérie Berthé and Michel Rigo, editors. *Sequences, Groups, and Number Theory.* Springer International Publishing, 2018. doi:10.1007/978-3-319-69152-7.

7 Brian H. Bowditch. Cut points and canonical splittings of hyperbolic groups. *Acta Mathematica*, 180(2):145–186, 1998. doi:10.1007/bf02392898.

8 Vaughn Climenhaga and Anatole Katok. *From Groups to Geometry and Back (Student Mathematical Library).* American Mathematical Society, 2017.

9 David Cohen and Chaim Goodman-Strauss. Strongly aperiodic subshifts on surface groups. *Groups, Geometry, and Dynamics*, 11(3):1041–1059, 2017. doi:10.4171/ggd/421.

10 David Bruce Cohen. The large scale geometry of strongly aperiodic subshifts of finite type. *Advances in Mathematics*, 308:599–626, 2017.

11 Martin J. Dunwoody. The accessibility of finitely presented groups. *Inventiones Mathematicae*, 81(3):449–457, October 1985. doi:10.1007/bf01388581.

12 Chaim Goodman-Strauss. A hierarchical strongly aperiodic set of tiles in the hyperbolic plane. *Theoretical Computer Science*, 411(7):1085–1093, 2010. doi:10.1016/j.tcs.2009.11.018.

13 Emmanuel Jeandel. Aperiodic Subshifts on Polycyclic Groups. *CoRR*, abs/1510.02360, 2015. URL: http://arxiv.org/abs/1510.02360.

14 Emmanuel Jeandel. Translation-like Actions and Aperiodic Subshifts on Groups. *CoRR*, abs/1508.06419, 2015. URL: http://arxiv.org/abs/1508.06419.

15 Jarkko Kari. On the Undecidability of the Tiling Problem. In *Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 74–82, 2008.

16 Abraham Karrass, Alfred Pietrowski, and Donald Solitar. Finite and infinite cyclic extensions of free groups. *Journal of the Australian Mathematical Society*, 16(04):458, December 1973. doi:10.1017/s1446788700015445.

17 Dietrich Kuske and Markus Lohrey. Logical aspects of Cayley-graphs: the group case. *Annals of Pure and Applied Logic*, 131(1–3):263–286, 2005. doi:10.1016/j.apal.2004.06.002.

18 Douglas A Lind and Brian Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.

19 Maurice Margenstern. The domino problem of the hyperbolic plane is undecidable. *Theoretical Computer Science*, 407(1-3):29–84, November 2008. doi:10.1016/j.tcs.2008.04.038.

20 David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37(0):51–75, 1985. doi:10.1016/0304-3975(85)90087-8.

21 Raphael M Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12:177–209, 1971.

22 Hao Wang. Proving theorems by pattern recognition II. *Bell System Technical Journal*, 40(1-3):1–41, 1961.

P-Optimal Proof Systems for Each NP-Set but no Complete Disjoint NP-Pairs Relative to an Oracle

Titus Dose
Julius-Maximilians-Universität Würzburg, Germany

Abstract

Pudlák [17] lists several major conjectures from the field of proof complexity and asks for oracles that separate corresponding relativized conjectures. Among these conjectures are:

- DisjNP: The class of all disjoint NP-pairs has no many-one complete elements.
- SAT: NP contains no many-one complete sets that have P-optimal proof systems.
- UP: UP has no many-one complete problems.
- $\mathsf{NP} \cap \mathsf{coNP}$: $\mathrm{NP} \cap \mathrm{coNP}$ has no many-one complete problems.

As one answer to this question, we construct an oracle relative to which DisjNP, $\neg\mathsf{SAT}$, UP, and $\mathsf{NP}\cap\mathsf{coNP}$ hold, i.e., there is no relativizable proof for the implication $\mathsf{DisjNP}\wedge\mathsf{UP}\wedge\mathsf{NP}\cap\mathsf{coNP} \Rightarrow \mathsf{SAT}$. In particular, regarding the conjectures by Pudlák this extends a result by Khaniki [9]. Since Khaniki [9] constructs an oracle showing that the implication $\mathsf{SAT} \Rightarrow \mathsf{DisjNP}$ has no relativizable proof, we obtain that the conjectures DisjNP and SAT are independent in relativized worlds, i.e., none of the implications $\mathsf{DisjNP} \Rightarrow \mathsf{SAT}$ and $\mathsf{SAT} \Rightarrow \mathsf{DisjNP}$ can be proven relativizably.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Proof complexity; Theory of computation → Oracles and decision trees

Keywords and phrases NP-complete, proof systems, disjoint NP-pair, oracle, UP

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.47

Related Version A full version [4] of the paper is available at `https://arxiv.org/abs/1904.06175`.

1 Introduction

The main motivation for the present paper is an article by Pudlák [17] that is “motivated by the problem of finding finite versions of classical incompleteness theorems”, investigates major conjectures in the field of proof complexity, discusses their relations, and draws new connections between the conjectures. Among others, Pudlák conjectures the following assertions (note that within the present paper all reductions are polynomial-time-bounded):

- CON (resp., SAT): coNP (resp., NP) contains no many-one complete sets that have P-optimal proof systems
- $\mathsf{CON}^{\mathsf{N}}$: coNP contains no many-one complete sets that have optimal proof systems, (note that $\mathsf{CON}^{\mathsf{N}}$ is the non-uniform version of CON)
- DisjNP (resp., $\mathsf{DisjCoNP}$): The class of all disjoint NP-pairs (resp., coNP-pairs) has no many-one complete elements,
- TFNP: The class of all total polynomial search problems has no complete elements,
- $\mathsf{NP} \cap \mathsf{coNP}$ (resp., UP): $\mathrm{NP} \cap \mathrm{coNP}$ (resp., UP, the class of problems accepted by NP machines with at most one accepting path for each input) has no many-one complete elements.

The following figure contains the conjectures by Pudlák and illustrates the state of the art regarding (i) known implications and (ii) separations in terms of oracles that prove the non-existence of relativizable proofs for implications. O denotes the oracle constructed in the present paper.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 47; pp. 47:1–47:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

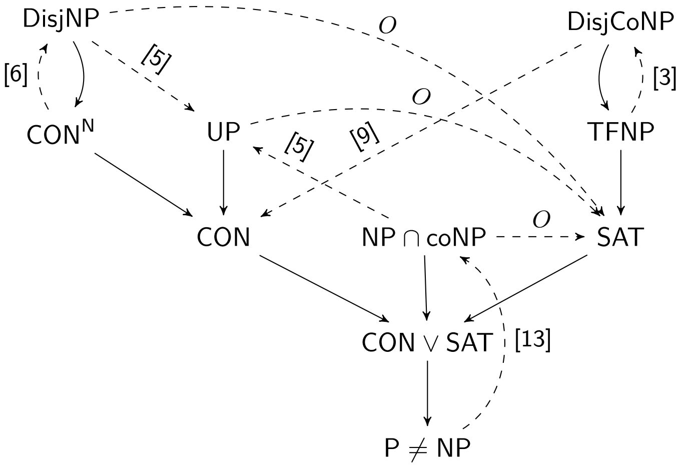


Figure 1 Solid arrows mean implications. All implications in the graphic can be proven with relativizable proofs. A dashed arrow from A to B means that there is an oracle X against the implication A ⇒ B, i.e., relative to X, it holds A ∧ ¬B. Pudlák [17] also defines the conjecture RFN$_1$ and lists it between CON ∨ SAT and P ≠ NP, i.e., CON ∨ SAT ⇒ RFN$_1$ ⇒ P ≠ NP. Khaniki [9] even shows CON ∨ SAT ⇔ RFN$_1$, which is why we omit RFN$_1$ in the figure.

The main conjectures of [17] are CON and TFNP. Let us give some background on these conjectures (for details we refer to [16]) and on the notion of disjoint pairs. The first main conjecture CON refers to the notion of proof systems introduced by Cook and Reckhow [2], who define a proof system for a set A to be a polynomial-time computable function whose range is A. The remainder of this paragraph originates from [5]. CON has an interesting connection to some finite version of an incompleteness statement. Denote by $\mathrm{Con}_T(n)$ the finite consistency of a theory T, i.e., $\mathrm{Con}_T(n)$ is the statement that T has no proofs of contradiction of length $\leq n$. Krajícek and Pudlák [12] raise the conjectures CON and CON$^\mathsf{N}$ and show that the latter is equivalent to the statement that there is no finitely axiomatized theory S which proves the finite consistency $\mathrm{Con}_T(n)$ for every finitely axiomatized theory T by a proof of polynomial length in n. In other words, ¬CON$^\mathsf{N}$ expresses that a weak version of Hilbert's program (to prove the consistency of all mathematical theories) is possible [15]. Correspondingly, ¬CON is equivalent to the existence of a theory S such that, for any fixed theory T, proofs of $\mathrm{Con}_T(n)$ in S can be constructed in polynomial time in n [12].

The conjecture TFNP was raised by Megiddo and Papadimitriou, is implied by the non-existence of disjoint coNP-pairs [1, 17], and implies that no NP-complete set has P-optimal proof systems [1, 17]. It states the non-existence of total polynomial search problems that are complete with respect to polynomial reductions.

The notion of disjoint NP-pairs, i.e., pairs (A, B) with $A \cap B = \emptyset$ and $A, B \in \mathrm{NP}$, was introduced by Grollman and Selman [7] in order to characterize promise problems. Razborov [18] connects it with the concept of propositional proof systems (pps), i.e., proof systems for the set of propositional tautologies TAUT, defining for each pps f a disjoint NP-pair, the so-called canonical pair of f, and showing that the canonical pair of an optimal pps f is complete. Hence, DisjNP ⇒ CON$^\mathsf{N}$.

In contrast to the many implications only very few oracles were known separating two of the relativized conjectures [17], which is why Pudlák asks for further oracles showing relativized conjectures to be different.

Khaniki [9] partially answers this question: besides proving two of the conjectures to be equivalent, he presents two oracles $\mathcal{V}$ and $\mathcal{W}$ showing that SAT and CON (just as TFNP and CON) are independent in relativized worlds, which means that none of the two possible implications between the two conjectures has a relativizable proof. To be more precise, relative to $\mathcal{V}$, there exist P-optimal propositional proof systems but no many-one complete disjoint coNP-pairs, where, as mentioned above, the latter implies TFNP and SAT. Relative to $\mathcal{W}$, there exist no P-optimal propositional proof systems and each total polynomial search problem has a polynomial-time solution, where the latter implies ¬SAT [10].

Dose and Glaßer [5] construct an oracle X that also separates some of the above relativized conjectures. Relative to X there exist no many-one complete disjoint NP-pairs, UP has many-one complete problems, and NP ∩ coNP has no many-one complete problems. In particular, relative to X, there do not exist P-optimal propositional proof systems. Thus, among others, X shows that the conjectures CON and UP as well as NP ∩ coNP and UP cannot be proven equivalent with relativizable proofs.

In another paper [3], the author adds one more oracle to this list proving that there is no relativizable proof for the implication TFNP ⇒ DisjCoNP.

Our Contribution

In the present paper we construct an oracle O relative to which

1. The class of all disjoint NP-pairs has no many-one complete elements.
2. NP contains no many-one complete sets that have P-optimal proof systems.
3. UP has no many-one complete problems.
4. NP ∩ coNP has no many-one complete problems.

Indeed, relative to O there even exist no disjoint NP-pairs hard for NP ∩ coNP, which implies both 1 and 4. Among others, the oracle shows that there are no relativizable proofs for the implications NP ∩ coNP ⇒ SAT and UP ⇒ SAT. Let us now focus on the properties 1 and 2 of the oracle. Regarding these, our oracle has similar properties as the aforementioned oracle $\mathcal{W}$ by Khaniki [9]: both oracles show that there is no relativizable proof for the implication CON ⇒ SAT. Relative to Khaniki's oracle $\mathcal{W}$ it even holds that each total polynomial search problem has a polynomial time solution, which implies not only ¬SAT but also that all optimal proof systems for SAT are P-optimal [10]. Regarding Pudlák's conjectures, however, our oracle O extends Khaniki's result as relative to O we have the even stronger result that there is no relativizable proof for the implication DisjNP ⇒ SAT. Since due to the oracle $\mathcal{V}$ by Khaniki [9] none of the implications DisjCoNP ⇒ DisjNP, TFNP ⇒ DisjNP, and SAT ⇒ DisjNP can be proven relativizably, our oracle shows that DisjNP is independent of each of the conjectures DisjCoNP, TFNP, and SAT in relativized worlds, i.e., none of the six possible implications has a relativizable proof.

2 Preliminaries

Major parts of this section are copied from our paper [5] coauthored by Christian Glaßer. Throughout this paper let Σ be the alphabet $\{0, 1\}$. We denote the length of a word $w \in \Sigma^*$ by $|w|$. Let $\Sigma^{\leq n} = \{w \in \Sigma^* \mid |w| \leq n\}$ and $\Sigma^{>n} = \{w \in \Sigma^* \mid |w| > n\}$. The empty word is denoted by ε and the i-th letter of a word w for $0 \leq i < |w|$ is denoted by $w(i)$, i.e., $w = w(0)w(1)\cdots w(|w| - 1)$. If v is a prefix of w, i.e., $|v| \leq |w|$ and $v(i) = w(i)$ for all $0 \leq i < |v|$, then we write $v \sqsubseteq w$. For any finite set $Y \subseteq \Sigma^*$, let $\ell(Y) \overset{df}{=} \sum_{w \in Y} |w|$.

$\mathbb{Z}$ denotes the set of integers, $\mathbb{N}$ denotes the set of natural numbers, and $\mathbb{N}^+ = \mathbb{N} - \{0\}$. The set of primes is denoted by $\mathbb{P} = \{2, 3, 5, \dots\}$ and $\mathbb{P}^{\geq 3}$ denotes the set $\mathbb{P} - \{2\}$.

We identify Σ^* with $\mathbb{N}$ via the polynomial-time computable, polynomial-time invertible bijection $w \mapsto \sum_{i<|w|}(1 + w(i))2^{|w|-1-i}$, which is a variant of the dyadic encoding. Hence, notations, relations, and operations for Σ^* are transferred to $\mathbb{N}$ and vice versa. In particular,

$|n|$ denotes the length of $n \in \mathbb{N}$. We eliminate the ambiguity of the expressions 0^i and 1^i by always interpreting them over Σ^*.

Let $\langle\cdot\rangle : \bigcup_{i\geq 0} \mathbb{N}^i \to \mathbb{N}$ be an injective, polynomial-time computable, polynomial-time invertible pairing function such that $|\langle u_1, \dots, u_n\rangle| = 2(|u_1| + \dots + |u_n| + n)$.

Given two sets A and B, $A - B$ denotes the set difference between A and B. The complement of a set A relative to the universe U is denoted by $\overline{A} = U - A$. The universe will always be apparent from the context. Furthermore, the symmetric difference is denoted by $\triangle$, i.e., $A \triangle B = (A - B) \cup (B - A)$ for arbitrary sets A and B.

The domain and range of a function t are denoted by $\mathrm{dom}(t)$ and $\mathrm{ran}(t)$, respectively.

FP, P, and NP denote standard complexity classes [14]. Define $\mathrm{co}\mathcal{C} = \{A \subseteq \Sigma^* \mid \overline{A} \in \mathcal{C}\}$ for a class $\mathcal{C}$. UP is the class of all problems accepted by nondeterministic polynomial-time Turing machines with at most one accepting path for each input. If $A, B \in \mathrm{NP}$ (resp., $A, B \in \mathrm{coNP}$) and $A \cap B = \emptyset$, then we call (A, B) a disjoint NP-pair (resp., a disjoint coNP-pair). The set of all disjoint NP-pairs (resp., coNP-pairs) is denoted by DisjNP (resp., DisjCoNP). We also consider all these complexity classes in the presence of an oracle O and denote the corresponding classes by FP^O, P^O, NP^O, and so on.

Let M be a Turing machine. $M^D(x)$ denotes the computation of M on input x with D as an oracle. For an arbitrary oracle D we let $L(M^D) = \{x \mid M^D(x) \text{ accepts}\}$. For a deterministic polynomial-time Turing transducer (i.e., a Turing machine computing a function), depending on the context, $F^D(x)$ either denotes the computation of F on input x with D as an oracle or the output of this computation.

▶ **Definition 1.** *A sequence $(M_i)_{i\in\mathbb{N}^+}$ is called* standard enumeration *of nondeterministic, polynomial-time oracle Turing machines, if it has the following properties:*

1. *All M_i are nondeterministic, polynomial-time oracle Turing machines.*
2. *For all oracles D and all inputs x the computation $M_i^D(x)$ stops within $|x|^i + i$ steps.*
3. *For every nondeterministic, polynomial-time oracle Turing machine M there exist infinitely many $i \in \mathbb{N}$ such that for all oracles D it holds that $L(M^D) = L(M_i^D)$.*
4. *There exists a nondeterministic, polynomial-time oracle Turing machine M such that for all oracles D and all inputs x it holds that $M^D(\langle i, x, 0^{|x|^i+i}\rangle)$ nondeterministically simulates the computation $M_i^D(x)$.*

Analogously we define standard enumerations of deterministic, polynomial-time oracle Turing transducers.

Throughout this paper, we fix some standard enumerations. Let $M_1, M_2, \dots$ be a standard enumeration of nondeterministic polynomial-time oracle Turing machines. Then for every oracle D, the sequence $(M_i)_{i\in\mathbb{N}^+}$ represents an enumeration of the languages in NP^D, i.e., $\mathrm{NP}^D = \{L(M_i^D) \mid i \in \mathbb{N}\}$, where as usual a computation $M_i^D(x)$ accepts if and only if it has at least one accepting path. Let $F_1, F_2, \dots$ be a standard enumeration of polynomial time oracle Turing transducers. By the properties of standard enumerations, for each oracle D the following problem is NP^D-complete (in particular it is in NP^D):

$$K^D = \{\langle 0^i, 0^t, x\rangle \mid M_i^D(x) \text{ accepts within } t \text{ steps}\}.$$

Let D be an oracle and $A, B, A', B' \subseteq \Sigma^*$ such that $A \cap B = A' \cap B' = \emptyset$. In this paper we always use the following reducibility for disjoint pairs [18]. (A', B') is polynomially many-one reducible to (A, B), denoted by $(A', B') \leq_{\mathrm{m}}^{\mathrm{pp},D} (A, B)$, if there exists $f \in \mathrm{FP}^D$ with $f(A') \subseteq A$ and $f(B') \subseteq B$. If $A' = \overline{B'}$, then we also write $A' \leq_{\mathrm{m}}^{\mathrm{p},D} (A, B)$ instead of $(A', B') \leq_{\mathrm{m}}^{\mathrm{pp},D} (A, B)$.

We say that (A, B) is $\leq_{\mathrm{m}}^{\mathrm{pp},D}$*-hard* ($\leq_{\mathrm{m}}^{\mathrm{pp},D}$*-complete*) *for* DisjNP^D if $(A', B') \leq_{\mathrm{m}}^{\mathrm{pp},D} (A, B)$ for all $(A', B') \in \mathrm{DisjNP}^D$ (and $(A, B) \in \mathrm{DisjNP}^D$). Moreover, a pair (A, B) is $\leq_{\mathrm{m}}^{\mathrm{p},D}$*-hard for* $\mathrm{NP}^D \cap \mathrm{coNP}^D$ if $A' \leq_{\mathrm{m}}^{\mathrm{p},D} (A, B)$ for every $A \in \mathrm{NP}^D \cap \mathrm{coNP}^D$.

▶ **Definition 2** ([2]). *A function $f \in \mathrm{FP}$ is called* proof system *for the set* $\mathrm{ran}(f)$*. For* $f, g \in \mathrm{FP}$ *we say that* f is simulated by g *(resp.,* f is P-simulated by g*) denoted by* $f \leq g$ *(resp.,* $f \leq^{\mathrm{p}} g$*), if there exists a function* π *(resp., a function* $\pi \in \mathrm{FP}$*) and a polynomial* p *such that* $|\pi(x)| \leq p(|x|)$ *and* $g(\pi(x)) = f(x)$ *for all* x*. A function* $g \in \mathrm{FP}$ *is* optimal *(resp.,* P-optimal*), if* $f \leq g$ *(resp.,* $f \leq^{\mathrm{p}} g$*) for all* $f \in \mathrm{FP}$ *with* $\mathrm{ran(f)} = \mathrm{ran(g)}$*. Corresponding relativized notions are obtained by using* P^O*,* FP^O*, and* $\leq^{\mathrm{p},O}$ *in the definitions above.*

The following proposition states the relativized version of a result by Köbler, Messner, and Torán [11], which they show with a relativizable proof.

▶ **Proposition 3** ([11]). *For every oracle* O*, if* A *has a* P^O*-optimal (resp., optimal) proof system and* $B \leq_{\mathrm{m}}^{\mathrm{p},O} A$*, then* B *has a* P^O*-optimal (resp., optimal) proof system.*

▶ **Corollary 4.** *For every oracle* O*, if there exists a* $\leq_{\mathrm{m}}^{\mathrm{p},O}$*-complete* $A \in \mathrm{NP}^O$ *that has a* P^O*-optimal (resp., optimal) proof system, then all sets in* NP^O *have* P^O*-optimal (resp., optimal) proof systems.*

Let us introduce some notations that are designed for the construction of oracles [5]. The support $\mathrm{supp}(t)$ of a real-valued function t is the subset of the domain that consists of all values that t does not map to 0. We say that a partial function t is injective on its support if $t(i, j) = t(i', j')$ for $(i, j), (i', j') \in \mathrm{supp}(t)$ implies $(i, j) = (i', j')$. If a partial function t is not defined at point x, then $t \cup \{x \mapsto y\}$ denotes the extension of t that at x has value y.

If A is a set, then $A(x)$ denotes the characteristic function at point x, i.e., $A(x)$ is 1 if $x \in A$, and 0 otherwise. An oracle $D \subseteq \mathbb{N}$ is identified with its characteristic sequence $D(0)D(1)\cdots$, which is an ω-word. In this way, $D(i)$ denotes both, the characteristic function at point i and the i-th letter of the characteristic sequence, which are the same. A finite word w describes an oracle that is partially defined, i.e., only defined for natural numbers $x < |w|$. We can use w instead of the set $\{i \mid w(i) = 1\}$ and write for example $A = w \cup B$, where A and B are sets. For nondeterministic oracle Turing machines M we use the following phrases: a computation $M^w(x)$ *definitely accepts*, if it contains a path that accepts and the queries on this path are $< |w|$. A computation $M^w(x)$ *definitely rejects*, if all paths reject and all queries are $< |w|$. For a nondeterministic Turing machine M we say that the computation $M^w(x)$ *is defined*, if it definitely accepts or definitely rejects. For a polynomial time oracle transducer F, the computation $F^w(x)$ *is defined*, if all queries are $< |w|$.

3 Oracle Construction

The following lemma is a slightly adapted variant of a result from [5].

▶ **Lemma 5.** *For all* $y \leq |w|$ *and all* $v \sqsupseteq w$ *it holds* $y \in K^v \Leftrightarrow y \in K^w$*.*

▶ **Theorem 6.** *There exists an oracle* O *such that the following statements hold:*

- DisjNP^O *contains no pairs that are* $\leq_{\mathrm{m}}^{\mathrm{p},O}$*-hard for* $\mathrm{NP}^O \cap \mathrm{coNP}^O$*.*
- *Each* $L \in \mathrm{NP}^O$ *has* P^O*-optimal proof systems.*
- UP^O *contains no* $\leq_{\mathrm{m}}^{\mathrm{p},O}$*-complete problems.*

Observe that the first of the three statements implies that both DisjNP^O contains no $\leq_{\mathrm{m}}^{\mathrm{pp},O}$-complete pairs and $\mathrm{NP}^O \cap \mathrm{coNP}^O$ contains no $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete problems.

Proof. Let D be a (possibly partial) oracle and p (resp., q) be in $\mathbb{P}_3$ (resp., $\mathbb{P}_1$). We define

$$\begin{aligned} A_p^D &:= \{0^{p^k} \mid k \in \mathbb{N}^+, \exists_{x\in\Sigma^{p^k}} x \in D \text{ and } x \text{ odd}\} \cup \overline{\{0^{p^k} \mid k \in \mathbb{N}^+\}} \\ B_p^D &:= \{0^{p^k} \mid k \in \mathbb{N}^+, \exists_{x\in\Sigma^{p^k}} x \in D \text{ and } x \text{ even}\} \\ C_q^D &:= \{0^{q^k} \mid k \in \mathbb{N}^+, \exists_{x\in\Sigma^{q^k}} x \in D\} \end{aligned}$$

Note that $A_p^D, B_p^D \in \mathrm{NP}^D$ and $A_p^D = \overline{B_p^D}$ if $|\Sigma^{p^k} \cap D| = 1$ for each $k \in \mathbb{N}^+$. In that case $A_p^D \in \mathrm{NP}^D \cap \mathrm{coNP}^D$. Moreover, $C_q^D \in \mathrm{UP}^D$ if and only if $|\Sigma^{q^k} \cap D| \leq 1$ for each $k \in \mathbb{N}^+$.

For the sake of simplicity, let us call a pair (M_i, M_j) an $\mathrm{NP}^D \cap \mathrm{coNP}^D$-machine if $L(M_i^D) = \overline{L(M_j^D)}$. For $i \in \mathbb{N}^+$ and $x, y \in \mathbb{N}$ we write $c(i,x,y) := \langle 0^i, 0^{|x|^i+i}, x, y\rangle$. Thus, $|c(i,x,y)|$ is even. Note that throughout this proof we sometimes omit the oracles in the superscript, e.g., we write NP or A_p instead of NP^D or A_p^D. However, we do not do that in the "actual" proof but only when explaining ideas in a loose way in order to give the reader the intuition behind the occasionally very technical arguments.
Preview of construction. We sketch some of the very basic ideas our construction uses.

1. For all positive $i \neq j$ the construction tries to achieve that (M_i, M_j) is not an $\mathrm{NP} \cap \mathrm{coNP}$-machine. If this is not possible, then $(L(M_i), L(M_j))$ inherently is an $\mathrm{NP} \cap \mathrm{coNP}$-machine. Once we know this, we choose some odd prime p and diagonalize against all FP-functions such that $A_p = \overline{B_p}$ and A_p is not $\leq_\mathrm{m}^\mathrm{p}$-reducible to $(L(M_i), L(M_j))$.
2. For all $i \geq 1$ the construction intends to make sure that F_i is not a proof system for K. If this is not possible, then F_i inherently is a proof system for K and then we start to encode the values of F_i into the oracle. However, it is important to also allow encodings for functions that are not known to be proof systems for K yet. Regarding the P-optimal proof systems, our construction is based on ideas by Dose and Glaßer [5].
3. For all $i \geq 1$ the construction tries to ensure that M_i is not a UP-machine. If this is not possible, we know that M_i inherently is a UP machine, which enables us to diagonalize against all FP-functions so that C_q for some q that we choose is not reducible to $L(M_i)$.

▷ Claim 7. Let $w \in \Sigma^*$ be an oracle, $i \in \mathbb{N}^+$, and $x, y \in \mathbb{N}$ such that $c(i,x,y) \leq |w|$. Then the following holds.

1. $F_i^w(x)$ is defined and $F_i^w(x) < |w|$.
2. $F_i^w(x) \in K^w \Leftrightarrow F_i^w(x) \in K^v$ for all $v \sqsupseteq w$.

During the construction we maintain a growing collection of requirements that is represented by a partial function belonging to the set

$$\mathcal{T} = \Big\{ t : \mathbb{N}^+ \cup (\mathbb{N}^+)^2 \to \mathbb{Z} \mid \mathrm{dom}(t) \text{ is finite, } t \text{ is injective on its support,} \\ \blacksquare\ t(\mathbb{N}^+) \subseteq \{0\} \cup \mathbb{N}^+ \\ \blacksquare\ t(\{(i,i) \mid i \in \mathbb{N}^+\}) \subseteq \{0\} \cup \{-q \mid q \in \mathbb{P}_1\} \\ \blacksquare\ t(\{(i,j) \in (\mathbb{N}^+)^2 \mid i \neq j\}) \subseteq \{0\} \cup \{-p \mid p \in \mathbb{P}_3\} \Big\}.$$

A partial oracle w is called t-valid for $t \in \mathcal{T}$ if it satisfies the following properties.

V1 For all $i \in \mathbb{N}^+$ and all $x, y \in \mathbb{N}$, if $c(i,x,y) \in w$, then $F_i^w(x) = y \in K^w$.
(meaning: if the oracle contains the codeword $c(i,x,y)$, then $F_i^w(x)$ outputs $y \in K^w$; hence, $c(i,x,y) \in w$ is a proof for $y \in K^w$)

V2 For all distinct $i, j \in \mathbb{N}^+$, if $t(i,j) = 0$, then there exists x such that $M_i^w(x)$ and $M_j^w(x)$ definitely accept.
(meaning: for every extension of the oracle, $(L(M_i), L(M_j))$ is not a disjoint NP-pair.)

V3 For all distinct $i, j \in \mathbb{N}^+$ with $t(i,j) = -p$ for some $p \in \mathbb{P}_3$ and each $k \in \mathbb{N}^+$, it holds (i) $|\Sigma^{p^k} \cap w| \leq 1$ and (ii) if w is defined for all words of length p^k, then $|\Sigma^{p^k} \cap w| = 1$.
(meaning: if $t(i,j) = -p$, then ensure that $A_p = \overline{B_p}$ (i.e., $A_p \in \mathrm{NP} \cap \mathrm{coNP}$) relative to the final oracle.)

V4 For all $i \in \mathbb{N}^+$ with $t(i) = 0$, there exists x such that $F_i^w(x)$ is defined and $F_i^w(x) \notin K^v$ for all $v \sqsupseteq w$.
(meaning: for every extension of the oracle, F_i is not a proof system for K)

V5 For all $i \in \mathbb{N}^+$ and $x \in \mathbb{N}$ with $0 < t(i) \leq c(i, x, F_i^w(x)) < |w|$, it holds $c(i, x, F_i^w(x)) \in w$.
(meaning: if $t(i) > 0$, then from $t(i)$ on, we encode the values of F_i into the oracle.
Note that V5 is not in contradiction with e.g. V3 or V7 as $|c(\cdot,\cdot,\cdot)|$ is even.)

V6 For all $i \in \mathbb{N}^+$ with $t(i,i) = 0$, there exists x such that $M_i^w(x)$ is defined and has two accepting paths.
(meaning: for every extension of the oracle, M_i is not a UP-machine.)

V7 For all $i \in \mathbb{N}^+$ with $t(i,i) = -q \in \mathbb{P}_1$ and each $k \in \mathbb{N}^+$, it holds $|\Sigma^{q^k} \cap w| \leq 1$.
(meaning: if $t(i,i) = -q$, ensure that C_q is in UP.)

▷ Claim 8. Let $t, t' \in \mathcal{T}$ such that t' is an extension of t. For all oracles $w \in \Sigma^*$, if w is t'-valid, then w is t-valid.

▷ Claim 9. Let $t \in \mathcal{T}$ and $u, v, w \in \Sigma^*$ be oracles such that $u \sqsubseteq v \sqsubseteq w$ and both u and w are t-valid. Then v is t-valid.

Oracle construction. Let T be an enumeration of $\bigcup_{i=1}^{3} (\mathbb{N}^+)^i$ having the property that (i,j) appears earlier than (i,j,r) for all $i,j,r \in \mathbb{N}^+$. Each element of T stands for a task. We treat the tasks in the order specified by T and after treating a task we remove it and possibly other tasks from T. We start with the nowhere defined function t_0 and the t_0-valid oracle $w_0 = \varepsilon$. Then we define functions $t_1, t_2, \ldots$ in $\mathcal{T}$ such that t_{i+1} is an extension of t_i and partial oracles $w_0 \sqsubsetneq w_1 \sqsubsetneq w_2 \sqsubsetneq \ldots$ such that each w_i is t_i-valid. Finally, we choose $O = \bigcup_{i=0}^{\infty} w_i$ (note that O is totally defined since in each step we strictly extend the oracle). We describe step $s > 0$, which starts with a t_{s-1}-valid oracle w_{s-1} and extends it to a t_s-valid $w_s \sqsupsetneq w_{s-1}$ (it will be argued later that all these steps are indeed possible). Let us recall that each task is immediately deleted from T after it is treated.

- Task i: Let $t' = t_{s-1} \cup \{i \mapsto 0\}$. If there exists a t'-valid $v \sqsupsetneq w_{s-1}$, then let $t_s = t'$ and $w_s = v$. Otherwise, let $t_s = t_{s-1} \cup \{i \mapsto |w_{s-1}|\}$ and choose $w_s = w_{s-1}b$ for $b \in \{0,1\}$ such that w_s is t_s-valid.
(meaning: try to ensure that F_i is not a proof system for K. If this is impossible, require that the values of F_i are encoded into the oracle.)
- Task (i,j) with $i \neq j$: Let $t' = t_{s-1} \cup \{(i,j) \mapsto 0\}$. If there exists a t'-valid $v \sqsupsetneq w_{s-1}$, then let $t_s = t'$ as well as $w_s = v$ and delete all tasks $(i,j,\cdot)$ from T. Otherwise, let $z = |w_{s-1}|$, choose $p \in \mathbb{P}_3$ greater than $|z|$ and all p' with $p' \in \mathbb{P}^{\geq 3}$ and $-p' \in \mathrm{ran}(t_{s-1})$, let $t_s = t_{s-1} \cup \{(i,j) \mapsto -p\}$, and choose $w_s = w_{s-1}b$ for $b \in \{0,1\}$ such that w_s is t_s-valid.
(meaning: try to ensure that $(L(M_i), L(M_j))$ is not a disjoint NP-pair. If this is impossible, choose a sufficiently large prime p. It will be made sure later that A_p does not reduce to $(L(M_i), L(M_j))$.)
- Task (i,j,r) with $i \neq j$: It holds $t_{s-1}(i,j) = -p$ for a prime $p \in \mathbb{P}_3$, since otherwise, this task would have been deleted in the treatment of task (i,j). Define $t_s = t_{s-1}$ and choose a t_s-valid $w_s \sqsupsetneq w_{s-1}$ such that for some $n \in \mathbb{N}^+$ one of the following two statements holds:

  - $0^n \in A_p^v$ for all $v \sqsupseteq w_s$ and $M_i^{w_s}(F_r^{w_s}(0^n))$ definitely rejects.
  - $0^n \in B_p^v$ for all $v \sqsupseteq w_s$ and $M_j^{w_s}(F_r^{w_s}(0^n))$ definitely rejects.

  (meaning: make sure that it does not hold $A_p \leq_{\mathrm{m}}^{\mathrm{p}} (L(M_i), L(M_j))$ via F_r)
- Task (i,i): Let $t' = t_{s-1} \cup \{(i,i) \mapsto 0\}$. If there exists a t'-valid $v \sqsupsetneq w_{s-1}$, then let $t_s = t'$ as well as $w_s = v$ and delete all tasks $(i,i,\cdot)$ from T. Otherwise, let $z = |w_{s-1}|$, choose $q \in \mathbb{P}_1$ greater than $|z|$ and all p' with $p' \in \mathbb{P}^{\geq 3}$ and $-p' \in \mathrm{ran}(t_{s-1})$, let $t_s = t_{s-1} \cup \{(i,i) \mapsto -q\}$, and choose $w_s = w_{s-1}b$ for $b \in \{0,1\}$ such that w_s is t_s-valid.

  (meaning: try to ensure that M_i is not a UP-machine. If this is impossible, choose a sufficiently large prime q. It will be made sure later that C_q does not reduce to $L(M_i)$.)
- Task (i,i,r): It holds $t_{s-1}(i,j) = -q$ for a prime $q \in \mathbb{P}_1$, since otherwise, this task would have been deleted in the treatment of task (i,i). Define $t_s = t_{s-1}$ and choose a t_s-valid $w_s \sqsupsetneq w_{s-1}$ such that for some $n \in \mathbb{N}^+$ one of the following conditions holds:
  - $0^n \in C_q^v$ for all $v \sqsupseteq w_s$ and $M_i^{w_s}(F_r^{w_s}(0^n))$ definitely rejects.
  - $0^n \notin C_q^v$ for all $v \sqsupseteq w_s$ and $M_i^{w_s}(F_r^{w_s}(0^n))$ definitely accepts.

  (meaning: make sure that it does not hold $C_q \leq_{\mathrm{m}}^{\mathrm{p}} L(M_i)$ via F_r)

We now show that the construction is possible. For that purpose, we first describe how a valid oracle can be extended by one bit such that it remains valid.

▷ Claim 10. Let $s \in \mathbb{N}$ and $w \in \Sigma^*$ be a t_s-valid oracle with $w \sqsupseteq w_s$. It holds for $z = |w|$:

1. If $z = c(i,x,y)$ for $i \in \mathbb{N}^+$ and $x, y \in \mathbb{N}$, $0 < t_s(i) \leq z$, and $F_i^w(x) = y$, then $y \in K^v$ for all $v \sqsupseteq w$.
2. There exists $b \in \{0,1\}$ such that wb is t_s-valid. In detail, the following statements hold.
   a. If $|z|$ is odd and for all $p \in \mathbb{P}$ and $k \in \mathbb{N}^+$ with $-p \in \mathrm{ran}(t_s)$ it holds $|z| \neq p^k$, then $w0$ and $w1$ are t_s-valid.
   b. If there exist $p \in \mathbb{P}_3$ and $k \in \mathbb{N}^+$ with $-p \in \mathrm{ran}(t_s)$ such that $|z| = p^k$, $z \neq 1^{p^k}$, and $w \cap \Sigma^{p^k} = \emptyset$, then $w0$ and $w1$ are t_s-valid.
   c. If there exist $p \in \mathbb{P}_3$ and $k \in \mathbb{N}^+$ with $-p \in \mathrm{ran}(t_s)$ such that $z = 1^{p^k}$ and $w \cap \Sigma^{p^k} = \emptyset$, then $w1$ is t_s-valid.
   d. If there exist $q \in \mathbb{P}_1$ and $k \in \mathbb{N}^+$ with $-q \in \mathrm{ran}(t_s)$ such that $|z| = q^k$ and $w \cap \Sigma^{q^k} = \emptyset$, then $w0$ and $w1$ are t_s-valid.
   e. If $z = c(i,x,y)$ for $i \in \mathbb{N}^+$ and $x, y \in \mathbb{N}$, $0 < t_s(i) \leq z$, and $F_i^w(x) = y$, then $w1$ is t_s-valid and $F_i^{w1}(x) = y$.
   f. In all other cases (i.e., none of the assumptions in (a)–(e) holds) $w0$ is t_s-valid.

In order to show that the above construction is possible, assume that it is not possible and let $s > 0$ be the least number such that it fails in step s.

If step s treats a task $t \in \mathbb{N}^+ \cup (\mathbb{N}^+)^2$, then $t_{s-1}(t)$ is not defined, since the value of t is defined in the unique treatment of the task t. If $t_s(t)$ is chosen to be 0, then the construction clearly is possible. Otherwise, due to the choice of $t_s(t)$, the t_{s-1}-valid oracle w_{s-1} is even t_s-valid and Claim 10 ensures that there exists a t_s-valid $w_{s-1}b$ for some $b \in \{0,1\}$. Hence, the construction does not fail in step s, a contradiction.

For the remainder of the proof that the construction above is possible we assume that step s treats a task $(i,j,r) \in (\mathbb{N}^+)^3$. We treat the cases $i = j$ and $i \neq j$ simultaneously whenever it is possible. Recall that in the case $i = j$ we work for the diagonalization ensuring that $L(M_i)$ is not a complete UP-set and in the case $i \neq j$ we work for the diagonalization ensuring that the pair $(L(M_i), L(M_j))$ is not hard for NP $\cap$ coNP.

In any case, $t_s = t_{s-1}$ and $t_s(i,j) = -p$ for some $p \in \mathbb{P}^{\geq 3}$ (recall $p \in \mathbb{P}_1$ if $i = j$ and $p \in \mathbb{P}_3$ if $i \neq j$). Let $\gamma(x) = (x^r + r)^{i+j} + i + j$ and choose $n = p^k$ for some $k \in \mathbb{N}^+$ such that

$$2^{2n-2} > 2^{n+1} \cdot \gamma(n) \tag{1}$$

and w_{s-1} is not defined for any word of length n. Note that $\gamma(n)$ is not less than the running time of each of the computations $M_i^D(F_r^D(0^n))$ and $M_j^D(F_r^D(0^n))$ for any oracle D.

We define $u \sqsupseteq w_{s-1}$ to be the minimal t_s-valid oracle that is defined for all words of length $< n$. Such an oracle exists by Claim 10.2. Moreover, for $z \in \Sigma^n$, let $u_z \sqsupsetneq u$ be the minimal t_s-valid oracle that contains z and that is defined for all words of length $\leq \gamma(n)$. By Claim 10.2, such oracles exist and $u_z \cap \Sigma^n = \{z\}$ (in detail, the second part follows from (2b, 2c, and 2f) or (2d and 2f) depending on whether $p \in \mathbb{P}_3$ or $p \in \mathbb{P}_1$, for the first part we also need that each valid oracle can be extended by one bit without losing its validity).

▷ Claim 11. Let $z \in \Sigma^n$.

1. For each $\alpha \in u_z \cap \Sigma^{>n}$ one of the following statements holds.
   - $\alpha = 1^{p'^\kappa}$ for some $p' \in \mathbb{P}_3$ with $-p' \in \mathrm{ran}(t_s)$ and some $\kappa > 0$.
   - $\alpha = c(i', x, y)$ for some $i', x, y \in \mathbb{N}$ with $i' > 0$, $0 < t_s(i') \leq c(i', x, y)$, $F_{i'}^{u_z}(x) = y$, and $y \in K^{u_z}$.
2. For all $p' \in \mathbb{P}_3$ with $-p' \in \mathrm{ran}(t_s)$ and all $\kappa > 0$, if $n < p'^\kappa \leq \gamma(n)$, then $u_z \cap \Sigma^{p'^\kappa} = \{1^{p'^\kappa}\}$.

Proof. 1. Let $\alpha \in u_z \cap \Sigma^{>n}$. Moreover, let u' be the prefix of u_z that has length α, i.e., α is the least word that u' is not defined for. In particular, it holds $u' \cap \Sigma^{\leq n} = u_z \cap \Sigma^{\leq n}$ and thus, $z \in u'$. As $u \sqsubseteq u' \sqsubseteq u_z$ and both u and u_z are t_s-valid, Claim 9 yields that u' is also t_s-valid. Let us apply Claim 10.2 to the oracle u'. If one of the cases 2a, 2b, 2d, and 2f can be applied, then $u'0$ is t_s-valid and can be extended to a t_s-valid oracle u'' with $|u''| = |u_z|$ by Claim 10.2. As u'' and u_z agree on all words $< \alpha$ and $\alpha \in u'' - u_z$, we obtain $z \in u''$ and $u'' < u_z$. This is a contradiction to the choice of u_z (recall that u_z is the minimal t_s-valid oracle that is defined for all words of length $\leq \gamma(n)$ and contains z).

Hence, by Claim 10.2, either (i) $\alpha = 1^{p'^\kappa}$ for some $p' \in \mathbb{P}_3$ and $\kappa > 0$ with $-p' \in \mathrm{ran}(t_s)$ or (ii) $\alpha = c(i', x, y)$ for $i', x, y \in \mathbb{N}$, $i' > 0$, $0 < t_s(i') \leq \alpha$, and $F_{i'}^{u'}(x) = y$. In the latter case $F_{i'}^{u'}(x)$ is defined by Claim 7 and $y \in K^v$ for all $v \sqsupseteq u'$ by Claim 10.1, which implies $F_{i'}^{u_z}(x) = y \in K^{u_z}$.

2. As $-p' \in \mathrm{ran}(t_s)$ and u_z is t_s-valid, V3 yields that there exists $\beta \in \Sigma^{p'^\kappa} \cap u_z$. Let β be the minimal element of $\Sigma^{p'^\kappa} \cap u_z$. It suffices to show $\beta = 1^{p'^\kappa}$. For a contradiction, we assume $\beta < 1^{p'^\kappa}$. Let u' be the prefix of u_z that is defined for exactly the words of length $< p'^\kappa$. Then $u \sqsubseteq u' \sqsubseteq u_z$ and both u and u_z are t_s-valid. Then by Claim 9, the oracle u' is t_s-valid as well. By Claim 10.2 u' can be extended to a t_s-valid oracle u'' that satisfies $|u''| = |u_z|$ and $u'' \cap \Sigma^{p'^\kappa} = \{1^{p'^\kappa}\}$. The last property guarantees that $u'' < u_z$ because $\beta \in u_z - u''$ and the oracles u'' and u_z agree on all words smaller than β. As furthermore $z \in u''$, we obtain a contradiction to the choice of u_z. This finishes the proof of Claim 11. ◁

We study the case that for some odd (resp., even) $z \in \Sigma^n$ the computation $M_i^{u_z}(F_r^{u_z}(0^n))$ (resp., $M_j^{u_z}(F_r^{u_z}(0^n))$) rejects. Then it even definitely rejects since u_z is defined for all words of length $\gamma(n)$. If $i \neq j$, then $p \in \mathbb{P}_3$ and since $z \in u_z$, we have $0^n \in A_p^v$ for all $v \sqsupseteq u_z$ (resp., $0^n \in B_p^v$ for all $v \sqsupseteq u_z$ if z is even). Analogously, if $i = j$, then $p \in \mathbb{P}_1$ and as $z \in u_z$, we have $0^n \in C_p^v$ for all $v \sqsupseteq u_z$. Hence, in all these cases we can choose $w_s = u_z$ and obtain a contradiction to the assumption that step s of the construction fails.

Therefore, for the remainder of the proof that the construction is possible we assume the following: For each $z \in \Sigma^n$ odd (resp., even) the computation $M_i^{u_z}(F_r^{u_z}(0^n))$ (resp., $M_j^{u_z}(F_r^{u_z}(0^n))$) definitely accepts.

Let U_z for $z \in \Sigma^n$ odd (resp., $z \in \Sigma^n$ even) be the set of all those oracle queries of the least accepting path of $M_i^{u_z}(F_r^{u_z}(0^n))$ (resp., $M_j^{u_z}(F_r^{u_z}(0^n))$) that are of length $\geq n$. Observe

$\ell(U_z) \leq \gamma(n)$. Moreover, define $Q_0(U_z) := U_z$ and for $m \in \mathbb{N}$, define

$$Q_{m+1}(U_z) := \bigcup_{\substack{c(i',x,y) \in Q_m(U_z) \\ i',x,y \in \mathbb{N}, i'>0}} \{q \in \Sigma^{\geq n} \mid q \text{ is queried by } F_{i'}^{u_z}(x)\}.$$

Let $Q(U_z) := \bigcup_{m \in \mathbb{N}} Q_m(U_z)$. Note that all words in $Q(U_z)$ have length $\geq n$.

▷ Claim 12. Let $z \in \Sigma^n$. Then $\ell(Q(U_z)) \leq 2\ell(U_z) \leq 2\gamma(n)$ and for all $q \in Q(U_z)$, $|q| \leq \gamma(n)$.

Proof. We show that for all $m \in \mathbb{N}$, $\ell(Q_{m+1}(U_z)) \leq 1/2 \cdot \ell(Q_m(U_z))$. Then $\sum_{m=0}^{s} 1/2^m \leq 2$ for all $s \in \mathbb{N}$ implies $\ell(Q(U_z)) \leq 2 \cdot \ell(U_z)$. Moreover, from $\ell(U_z) \leq \gamma(n)$ and $\ell(Q_{m+1}(U_z)) \leq 1/2 \cdot \ell(Q_m(U_z))$ the second part of the claim follows.

Let $m \in \mathbb{N}$ and consider an arbitrary element α of $Q_m(U)$. If α is not of the form $c(i', x, y)$ for $i' \in \mathbb{N}^+$ and $x, y \in \mathbb{N}$, then α generates no elements in $Q_{m+1}(U)$. Assume $\alpha = c(i', x, y)$ for $i' \in \mathbb{N}^+$ and $x, y \in \mathbb{N}$. The computation $F_{i'}^{u_z}(x)$ runs for at most $|x|^{i'} + i' < |\alpha|/2$ steps, where "$<$" holds by the definition of $c(\cdot,\cdot,\cdot)$ and the properties of the pairing function $\langle\cdot\rangle$. Hence, the set of queries Q of $F_{i'}^{u_z}(x)$ satisfies $\ell(Q) < |\alpha|/2$. Consequently,

$$\begin{aligned}\ell(Q_{m+1}(U)) &\leq \sum_{\substack{c(i',x,y) \in Q_m(U_z) \\ i',x,y \in \mathbb{N}, i'>0}} \ell\big(\{q \in \Sigma^{\geq n} \mid q \text{ is queried by } F_{i'}^{u_z}(x)\}\big) \\ &\leq \sum_{\substack{c(i',x,y) \in Q_m(U_z) \\ i',x,y \in \mathbb{N}, i'>0}} |c(i',x,y)|/2 \leq \ell(Q_m(U_z))/2,\end{aligned}$$

which finishes the proof of Claim 12. ◁

For $z, z' \in \Sigma^n$ we say that $Q(U_z)$ and $Q(U_{z'})$ *conflict* if there is a word $\alpha \in Q(U_z) \cap Q(U_{z'})$ in $u_z \triangle u_{z'}$. In that case, we say $Q(U_z)$ and $Q(U_{z'})$ conflict in α. Note that whenever $Q(U_z)$ and $Q(U_{z'})$ conflict in a word α, then $\alpha \in u_z \cup u_{z'}$. The next three claims show that for all $z \in \Sigma^n$ odd and $z' \in \Sigma^n$ even, the sets $Q(U_z)$ and $Q(U_{z'})$ conflict in a word of length n. Indeed, then they conflict in z or z' as these are the only words of length n in $u_z \cup u_{z'}$.

▷ Claim 13. Let $z, z' \in \Sigma^n$ such that z is odd and z' is even. If $Q(U_z)$ and $Q(U_{z'})$ conflict, then they conflict in a word of length n.

Proof. Let α be the least word in which $Q(U_z)$ and $Q(U_{z'})$ conflict. Then $\alpha \in u_z \triangle u_{z'}$. By symmetry, it suffices to consider the case $\alpha \in u_z - u_{z'}$. For a contradiction, assume that $|\alpha| > n$. Then by Claim 11, two situations are possible.

1. Assume $\alpha = 1^{p'^\kappa}$ for $p' \in \mathbb{P}_3$ with $-p' \in \mathrm{ran}(t_s)$ and $\kappa > 0$. Then by Claim 11.2, $\alpha \in u_{z'}$, a contradiction. Hence, $\alpha \neq 1^{p'^\kappa}$ for all $p' \in \mathbb{P}_3$ with $-p' \in \mathrm{ran}(t_s)$ and $\kappa > 0$.

2. Here, $\alpha = c(i', x, y)$ for $i' \in \mathbb{N}^+$ and $x, y \in \mathbb{N}$ with $0 < t_s(i') \leq c(i', x, y)$ and $F_{i'}^{u_z}(x) = y \in K^{u_z}$. By construction, it holds $t_s(i') = t_{s-1}(i') \leq |w_{s-1}| \leq |u| < \alpha$. Thus, $F_{i'}^{u_{z'}}(x) \neq y$, since otherwise, by the t_s-validity of $u_{z'}$ and V5, it would hold $\alpha \in u_{z'}$. Consequently, $F_{i'}^{u_{z'}}(x) \neq F_{i'}^{u_z}(x)$. Hence, there exists a query β that is asked by both $F_{i'}^{u_z}(x)$ and $F_{i'}^{u_{z'}}(x)$ and that is in $u_z \triangle u_{z'}$ (otherwise, both computations would output the same word). By definition of $Q(U_z)$ and $Q(U_{z'})$, it holds $\beta \in Q(U_z) \cap Q(U_{z'})$. Hence, $Q(U_z)$ and $Q(U_{z'})$ conflict in β and $|\beta| \leq |x|^{i'} + i' < |c(i', x, y)| = |\alpha|$, in contradiction to the assumption that α is the least word which $Q(U_z)$ and $Q(U_{z'})$ conflict in. ◁

For showing that for all odd $z \in \Sigma^n$ and all even $z' \in \Sigma^n$ the sets $Q(U_z)$ and $Q(U_{z'})$ conflict, we need one more claim. Let t' be defined such that $\mathrm{dom}(t') = \mathrm{dom}(t_s) - \{(i, j)\}$ and $t'(i', j') = t_s(i', j')$ for $(i', j') \in \mathrm{dom}(t')$. Then u and u_z for $z \in \Sigma^n$ are t'-valid by Claim 8.

▷ Claim 14. Let $t \in \{t', t_s\}$ and $z, z' \in \Sigma^n$ such that $Q(U_z)$ and $Q(U_{z'})$ do not conflict. For each t-valid oracle $v \sqsupsetneq u$ that is defined for exactly the words of length $\leq n$ and that satisfies $v(q) = u_z(q)$ for all $|v| > q \in Q(U_z)$ and $v(q) = u_{z'}(q)$ for all $|v| > q \in Q(U_z)$, there exists a t-valid oracle $v' \sqsupsetneq v$ with $|v'| = |u_z|$, $v'(q) = u_z(q)$ for all $|v'| > q \in Q(U_z)$, and $v'(q) = u_{z'}(q)$ for all $|v'| > q \in Q(U_z)$.

Proof. Let $w \sqsupseteq u$ with $|w| < |u_z|$, $w(q) = u_z(q)$ for all $|w| > q \in Q(U_z)$, and $w(q) = u_{z'}(q)$ for all $|w| > q \in Q(U_{z'})$. Moreover, let $\alpha = |w|$. It suffices to show the following:

- If $\alpha = 0^{p'^\kappa}$ for some $p' \in \mathbb{P}_3$ with $-p' \in \mathrm{ran}(t)$ and $\kappa > 0$, then there exists a t-valid $w' \sqsupsetneq w$ that is defined for the words of length p'^κ, undefined for all words of greater length, and that satisfies $w'(q) = u_z(q)$ for all $|w'| > q \in Q(U_z)$ and $w'(q) = u_{z'}(q)$ for all $|w'| > q \in Q(U_{z'})$.
  Note that in this case $|w'| \leq |u_z|$ as u_z is defined for exactly the words of length $\leq \gamma(n)$.
- If α is not of length p'^κ for all $p' \in \mathbb{P}_3$ with $-p' \in \mathrm{ran}(t)$ and all $\kappa > 0$, then there exists $b \in \{0, 1\}$ such that wb is t-valid, $wb(q) = u_z(q)$ for all $|wb| > q \in Q(U_z)$ and $wb(q) = u_{z'}(q)$ for all $|wb| > q \in Q(U_{z'})$.

We study three cases.

1. Assume $\alpha = 0^{p'^\kappa}$ for some $p' \in \mathbb{P}_3$ with $-p' \in \mathrm{ran}(t)$ and $\kappa > 0$. Then we let $w' \sqsupsetneq w$ be the minimal oracle that is defined for all words of length p'^κ and contains $1^{p'^\kappa}$, i.e., $w' = w \cup \{1^{p'^\kappa}\}$ when interpreting the oracles as sets. As $u_z \cap \Sigma^{p'^\kappa} = u_{z'} \cap \Sigma^{p'^\kappa} = \{1^{p'^\kappa}\}$ by Claim 11.2, we obtain $w'(q) = u_z(q)$ for all $|w'| > q \in Q(U_z)$ and $w'(q) = u_{z'}(q)$ for all $|w'| > q \in Q(U_{z'})$. Moreover, by Claim 10.2b and Claim 10.2c, the oracle w' is t-valid.
2. Now assume that $\alpha = c(i', x, y)$ for $i' \in \mathbb{N}^+$ and $x, y \in \mathbb{N}$ with $0 < t(i') \leq \alpha$ such that one of the conditions $F_{i'}^{u_z}(x) = y$ and $F_{i'}^{u_{z'}}(x) = y$ holds. By Claim 10.1, then even one of the two conditions $F_{i'}^{u_z}(x) = y \in K^{u_z}$ and $F_{i'}^{u_{z'}}(x) = y \in K^{u_{z'}}$ holds. By symmetry, if suffices to argue for the case $F_{i'}^{u_z}(x) = y \in K^{u_z}$. Recall that the oracles u_z and $u_{z'}$ are t-valid. Hence, by V5 and $0 < t(i') \leq \alpha < |u_z|$, it holds $\alpha \in u_z$. We consider two cases depending on whether $F_{i'}^{w}(x)$ returns y. In any case, if $\alpha \in Q(U_z)$ (resp., $\alpha \in Q(U_{z'})$), then $F_{i'}^{w}(x) = F_{i'}^{u_z}(x)$ (resp., $F_{i'}^{w}(x) = F_{i'}^{u_{z'}}(x)$), since for all queries q of $F_{i'}^{u_z}(x)$ (resp., $F_{i'}^{u_{z'}}(x)$), it holds $q \in Q(U_z)$ (resp., $q \in Q(U_{z'})$), $|q| \leq |x|^{i'} + i' < |\alpha|$, and by assumption, $w(q) = u_z(q)$ (resp., $w(q) = u_{z'}(q)$).
   - (i) Assume $F_{i'}^{w}(x) = y$. Choose $b = 1$. As w is t-valid, $0 < t(i') \leq \alpha$, and $F_{i'}^{w}(x) = y$, Claim 10.2e yields that $w1$ is t-valid. As $\alpha \in u_z$, we have $w1(q) = u_z(q)$ for all $|w1| > q \in Q(U_z)$. It remains to show that $w1(q) = u_{z'}(q)$ for all $|w1| > q \in Q(U_{z'})$. If $\alpha \notin Q(U_{z'})$, this trivially holds. If $\alpha \in Q(U_{z'})$, then as observed above, $F_{i}^{u_{z'}}(x) = F_{i}^{w}(x) = y$. Hence, as $u_{z'}$ is t-valid and $0 < t'(i') \leq \alpha < |u_{z'}|$, it holds $\alpha \in u_{z'}$ by V5. Thus, $w1(q) = u_{z'}(q)$ for all $|w1| > q \in Q(U_{z'})$.
   - (ii) Assume $F_{i'}^{w}(x) \neq y$. Choose $b = 0$. Then Claim 10.2f states that wb is t-valid. It holds $\alpha \notin Q(U_z)$, since otherwise, as observed above, $F_{i'}^{w}(x) = F_{i'}^{u_z}(x) = y$, which would yield a contradiction. Thus, $wb(q) = u_z(q)$ for all $|wb| > q \in Q(U_z)$. It remains to show $wb(q) = u_{z'}(q)$ for all $|wb| > q \in Q(U_{z'})$. If $\alpha \notin Q(U_{z'})$, this trivially holds and otherwise, it also holds, since as observed above, we have $F_{i'}^{u_{z'}}(x) = F_{i'}^{w}(x) \neq y$, which implies $\alpha \notin u_{z'}$ (by V1, $\alpha \in u_{z'}$ would imply $F_{i'}^{u_{z'}}(x) = y$).
3. We now consider the remaining cases, i.e., we may assume: (i) α is not of length p'^κ for all $p' \in \mathbb{P}_3$ and $\kappa > 0$ with $-p' \in \mathrm{ran}(t)$ and (ii) if $\alpha = c(i', x, y)$ for some $i' \in \mathbb{N}^+$ and $x, y \in \mathbb{N}$ with $0 < t(i') \leq \alpha$, then none of the conditions $F_{i'}^{u_z}(x) = y$ and $F_{i'}^{u_{z'}}(x) = y$ holds.

In this case, it holds $\alpha \notin u_z \cup u_{z'}$ by Claim 11.1. We choose $b = 0$ and obtain that $wb(q) = u_z(q)$ for all $|wb| > q \in Q(U_z)$ and $wb(q) = u_{z'}(q)$ for all $|wb| > q \in Q(U_{z'})$. Moreover, by Claim 10.2, wb is t-valid. This shows Claim 14. ◀

▷ Claim 15. For all odd $z \in \Sigma^n$ and all even $z' \in \Sigma^n$, $Q(U_z)$ and $Q(U_{z'})$ conflict.

Proof. Let us first show that $\alpha \in Q(U_\alpha)$ for all $\alpha \in \Sigma^n$. For a contradiction, assume $\alpha \notin Q(U_\alpha)$ for some $\alpha \in \Sigma^n$. We study the cases $i = j$ and $i \neq j$ separately.

First assume $i = j$. In this case $p \in \mathbb{P}_1$. Let u' be the oracle that is defined for exactly the words of length $\leq n$ and satisfies $u' = u$ when the oracles are considered as sets. Then u' is t_s-valid by Claim 10.2d and u' and u_α agree on all words in $\Sigma^n \cap Q(U_\alpha)$ as $u_\alpha \cap \Sigma^n = \{\alpha\}$ and $\alpha \notin Q(U_\alpha)$. Thus, we can apply Claim 14 to the oracle u' for $z = z' = \alpha$. Hence, there exists a t_s-valid oracle v satisfying $|v| = |u_z|$, $v \cap \Sigma^n = \emptyset$, and $v(q) = u_\alpha(q)$ for all $q \in Q(U_\alpha)$. By the latter property and the fact that $U_\alpha \subseteq Q(U_\alpha)$ contains all queries asked by the least accepting path of $M_i^{u_\alpha}(F_r^{u_\alpha}(0^n))$, this path is also an accepting path of the computation $M_i^v(F_r^v(0^n))$. As v is defined for all words of length $\leq \gamma(n)$, the computation $M_i^v(F_r^v(0^n))$ is defined. Thus, $0^n \notin C_q^{v'}$ for all $v' \sqsupseteq v$ and $M_i^v(F_r^v(0^n))$ definitely accepts, in contradiction to the assumption that step s of the construction fails.

Now let us consider the case $i \neq j$. Here $p \in \mathbb{P}_3$. By symmetry, it suffices to consider the case that α is odd. Let α' be the minimal even element of Σ^n that is not in $Q(U_\alpha)$. Such α' exists as it holds $2^{n-1} > 4(\gamma(n)) > 2\gamma(n)$ by (1), $\ell(Q(U_\alpha)) \leq 2\gamma(n)$ by Claim 12, and hence, $\ell(Q(U_\alpha)) \leq 2\gamma(n) < 2^{n-1} = |\{\alpha'' \in \Sigma^n \mid \alpha'' \text{ even}\}|$. Now choose u' to be the oracle that is defined for exactly the words of length $\leq n$ and that satisfies $u' = u \cup \{\alpha'\}$ when the oracles are considered as sets. Then u' is t_s-valid by Claim 10.2b and Claim 10.2f. Moreover, as $\alpha, \alpha' \notin Q(U_\alpha)$, the oracles u' and u_α agree on all words in $\Sigma^n \cap Q(U_\alpha)$. Thus, we can apply Claim 14 to the oracle u' for $z = z' = \alpha$ and obtain a t_s-valid oracle v that is defined for all words of length $\leq \gamma(n)$ and satisfies both $v \cap \Sigma^n = \{\alpha'\}$ and $v(q) = u_\alpha(q)$ for all $q \in Q(U_\alpha)$. The latter property and the fact that $U_\alpha \subseteq Q(U_\alpha)$ contains all queries asked by the least accepting path of $M_i^{u_\alpha}(F_r^{u_\alpha}(0^n))$ yield that this path is also an accepting path of the computation $M_i^v(F_r^v(0^n))$. As v is defined for all words of length $\leq \gamma(n)$, the computation $M_i^v(F_r^v(0^n))$ definitely accepts. Let us study two cases depending on whether $M_j^v(F_r^v(0^n))$ definitely accepts or definitely rejects (note that this computation is defined as v is defined for all words of length $\leq \gamma(n)$):

- Assume that $M_j^v(F_r^v(0^n))$ definitely accept. Let s' be the step that treats the task (i, j). Hence, $s' < s$ since $t_s(i, j)$ is defined. By Claim 8, the oracle v is $t_{s'-1}$-valid. Now, as both $M_i^v(F_r^v(0^n))$ and $M_j^v(F_r^v(0^n))$ definitely accept, v is even t''-valid for $t'' = t_{s'-1} \cup \{(i, j) \mapsto 0\}$. But then the construction would have chosen $t_{s'} = t''$ and a suitable oracle $w_{s'}$ (e.g., $w_{s'} = v$), a contradiction.
- Assume that $M_j^v(F_r^v(0^n))$ definitely rejects. As $v \cap \Sigma^n = \{\alpha'\}$, it holds $0^n \in B_p^{v'}$ for all $v' \sqsupseteq v$. This is a contradiction to the assumption that step s of the construction fails.

Hence, from now on we may assume that $\alpha \in Q(U_\alpha)$ for all $\alpha \in \Sigma^n$. Moreover, assume there are z odd and z' even such that $Q(U_z)$ and $Q(U_{z'})$ do not conflict. Then let $u' \sqsupsetneq u$ be the minimal oracle that is defined for all words of length $\leq n$ and contains z and z', i.e., interpreting oracles as sets it holds $u' = u \cup \{z, z'\}$. Since $-p \notin \mathrm{ran}(t')$, the oracle u' is t'-valid by Claim 10.2a. If Claim 14 cannot be applied to the oracle u' for z and z', then $z \in Q(U_{z'})$ or $z' \in Q(U_z)$. Since we observed above that $z \in Q(U_z)$ and $z' \in Q(U_{z'})$ and moreover, $u_z \cap \Sigma^n = \{z\}$ and $u_{z'} \cap \Sigma^n = \{z'\}$, in this case $Q(U_z)$ and $Q(U_{z'})$ conflict. Hence, it remains to consider the case that Claim 14 can be applied to the oracle u' for z and z'.

Applying Claim 14, we obtain a t'-valid $v \sqsupseteq u'$ that is defined for all words of length $\leq \gamma(n)$ and that satisfies $v(q) = u_z(q)$ for all $q \in Q(U_z)$ and $v(q) = u_{z'}(q)$ for all $q \in Q(U_{z'})$.

Let s' be the step in which (i,j) is treated. As $t_s(i,j)$ is defined, it holds $s' < s$. Hence, t' is an extension of $t_{s'-1}$ and by Claim 8, v is $t_{s'-1}$-valid. We claim that v is t''-valid for $t'' = t_{s'-1} \cup \{(i,j) \mapsto 0\}$. Once this is proven, we obtain a contradiction as then the construction would have chosen $t_{s'} = t''$ and an appropriate $w_{s'}$ (e.g. $w_{s'} = v$). Then our assumption is wrong and for all odd $z \in \Sigma^n$ and all even $z' \in \Sigma^n$, $Q(U_z)$ and $Q(U_{z'})$ conflict.

It remains to prove that v is t''-valid for $t'' = t_{s'-1} \cup \{(i,j) \mapsto 0\}$. We study two cases.

Case 1: first we assume that $i \neq j$, i.e., it suffices to prove that $M_i^v(F_r^v(0^n))$ and $M_j^v(F_r^v(0^n))$ definitely accept. Recall that $M_i^{u_z}(F_r^{u_z}(0^n))$ and $M_j^{u_{z'}}(F_r^{u_{z'}}(0^n))$ definitely accept. Moreover, $v(q) = u_z(q)$ for all $q \in Q(U_z)$ and $v(q) = u_{z'}(q)$ for all $q \in Q(U_{z'})$ and in particular, v is defined for all words in $Q(U_z) \cup Q(U_{z'})$. This implies that the least accepting paths of $M_i^{u_z}(F_r^{u_z}(0^n))$ and $M_i^{u_z}(F_r^{u_z}(0^n))$ are also accepting paths of the computations $M_i^v(F_r^v(0^n))$ and $M_j^v(F_r^v(0^n))$.

Case 2: assume that $i = j$, i.e., we have to prove that on some input x the computation $M_i^v(x)$ has two accepting paths. As observed above, $z \in Q(U_z)$ and $z' \in Q(U_{z'})$. As $Q(U_z)$ and $Q(U_{z'})$ do not conflict, it holds $z \notin Q(U_{z'})$, which implies $Q(U_z) \neq Q(U_{z'})$. Let $\kappa \in \mathbb{N}$ be minimal such that $Q_\kappa(U_z) \neq Q_\kappa(U_{z'})$ and for a contradiction, assume $\kappa > 0$. Let $\alpha \in Q_\kappa(U_z) \triangle Q_\kappa(U_{z'})$. Without loss of generality, we assume $\alpha \in Q_\kappa(U_z) - Q_\kappa(U_{z'})$. Then there exist $i', x, y \in \mathbb{N}$ with $i' > 0$ such that $c(i', x, y) \in Q_{\kappa-1}(U_z)$ and $F_{i'}^{u_z}(x)$ asks the query α. By the choice of κ, it holds $Q_{\kappa-1}(U_{z'}) = Q_{\kappa-1}(U_z)$ and thus, $c(i', x, y) \in Q_{\kappa-1}(U_{z'})$. Consequently, all queries of $F_{i'}^{u_{z'}}(x)$ are in $Q_\kappa(U_{z'})$. However, $\alpha \notin Q_\kappa(U_{z'})$ and therefore, α cannot be asked by $F_{i'}^{u_{z'}}(x)$. This shows that there is a word $\beta \in u_z \triangle u_{z'}$ asked by both $F_{i'}^{u_z}(x)$ and $F_{i'}^{u_{z'}}(x)$ (otherwise, the two computations would ask the same queries). But then $\beta \in Q_\kappa(U_z) \cap Q_\kappa(U_{z'})$, which implies that $Q(U_z)$ and $Q(U_{z'})$ conflict, a contradiction. Hence, we obtain $\kappa = 0$ and $U_z = Q_0(U_z) \neq Q_0(U_{z'}) = U_{z'}$. Recall that U_z (resp., $U_{z'}$) is the set consisting of all oracle queries of the least accepting path P (resp., P') of the computation $M_i^{u_z}(F_r^{u_z}(0^n))$ (resp., $M_i^{u_{z'}}(F_r^{u_{z'}}(0^n))$). As $u_z(q) = v(q)$ for all $q \in Q(U_z) \supseteq U_z$ and $u_{z'}(q) = v(q)$ for all $q \in Q(U_{z'}) \supseteq U_{z'}$, the paths P and P' are accepting paths of the computation $M_i^v(F_r^v(0^n))$. Finally, P and P' are distinct paths since $U_z \neq U_{z'}$. This finishes the proof that v is t''-valid. Hence, the proof of Claim 15 is complete. ◀

The remainder of the proof that the construction is possible is based on an idea by Hartmanis and Hemachandra [8]. Consider the set

$$\begin{aligned} E &= \{\{z, z'\} \mid z, z' \in \Sigma^n, z \text{ odd} \Leftrightarrow z' \text{ even}, (z \in Q(U_{z'}) \vee z' \in Q(U_z))\} \\ &= \bigcup_{z \in \Sigma^n} \{\{z, z'\} \mid z' \in \Sigma^n, z \text{ odd} \Leftrightarrow z' \text{ even}, z' \in Q(U_z)\}. \end{aligned} \tag{2}$$

Let $z, z' \in \Sigma^n$ such that (z odd $\Leftrightarrow z'$ even). By Claim 15 and Claim 13, $Q(U_z)$ and $Q(U_{z'})$ conflict in a word of length n. Then, as observed above, they conflict in z or z', i.e., $z \in Q(U_{z'})$ or $z' \in Q(U_z)$. This shows $E = \{\{z, z'\} \mid z, z' \in \Sigma^n, z \text{ odd} \Leftrightarrow z' \text{ even}\}$ and thus, $|E| = 2^{2n-2}$. By Claim 12, for each $z \in \Sigma^n$ it holds $|Q(U_z)| \leq \ell(Q(U_z)) \leq 2\gamma(n)$. Consequently,

$$|E| \overset{(2)}{\leq} \sum_{z \in \Sigma^n} |Q(U_z)| \leq 2^n \cdot 2\gamma(n) = 2^{n+1} \cdot \gamma(n) \overset{(1)}{<} 2^{2n-2} = |E|,$$

a contradiction to the assumption that the construction fails in step s treating the task (i, j, r). This shows that the construction is possible and O is well-defined. It remains to show that DisjNP^O contains no pair $\leq_{\mathrm{m}}^{\mathrm{p},O}$-hard for $\mathrm{NP}^O \cap \mathrm{coNP}^O$, each problem in NP^O has P^O-optimal proof systems, and UP^O has no $\leq_{\mathrm{m}}^{\mathrm{p},O}$-complete problem. As the corresponding proofs are rather straightforward, we omit them. This completes the proof of Theorem 6. ◀

References

1 O. Beyersdorff, J. Köbler, and J. Messner. Nondeterministic functions and the existence of optimal proof systems. *Theor. Comput. Sci.*, 410(38-40):3839–3855, 2009. doi:10.1016/j.tcs.2009.05.021.

2 S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979.

3 T. Dose. Complete Disjoint coNP-Pairs but no Complete Total Polynomial Search Problems Relative to an Oracle. *arXiv e-prints*, pages 1–13, March 2019. arXiv:1903.11860.

4 T. Dose. P-Optimal Proof Systems for Each Set in NP but no Complete Disjoint NP-pairs Relative to an Oracle. *arXiv e-prints*, pages 1–19, April 2019. arXiv:1904.06175.

5 T. Dose and C. Glaßer. NP-completeness, proof systems, and disjoint NP-pairs. Technical Report 19-050, Electronic Colloquium on Computational Complexity (ECCC), 2019.

6 C. Glaßer, A. L. Selman, S. Sengupta, and L. Zhang. Disjoint NP-Pairs. *SIAM Journal on Computing*, 33(6):1369–1416, 2004.

7 J. Grollmann and A. L. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988.

8 J. Hartmanis and L. A. Hemachandra. Complexity Classes without Machines: On Complete Languages for UP. *Theor. Comput. Sci.*, 58:129–142, 1988. doi:10.1016/0304-3975(88)90022-9.

9 E. Khaniki. New relations and separations of conjectures about incompleteness in the finite domain. *arXiv e-prints*, pages 1–25, April 2019. arXiv:1904.01362.

10 J. Köbler and J. Messner. Is the Standard Proof System for SAT P-Optimal? In S. Kapoor and S. Prasad, editors, *FSTTCS 2000: Foundations of Software Technology and Theoretical Computer Science*, pages 361–372, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

11 J. Köbler, J. Messner, and J. Torán. Optimal proof systems imply complete sets for promise classes. *Information and Computation*, 184(1):71–92, 2003.

12 J. Krajícek and P. Pudlák. Propositional Proof Systems, the Consistency of First Order Theories and the Complexity of Computations. *Journal of Symbolic Logic*, 54:1063–1079, 1989.

13 M. Ogiwara and L. Hemachandra. A complexity theory of feasible closure properties. *Journal of Computer and System Sciences*, 46:295–325, 1993.

14 C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

15 P. Pudlák. On the lengths of proofs of consistency. In *Collegium Logicum*, pages 65–86. Springer Vienna, 1996.

16 P. Pudlák. *Logical Foundations of Mathematics and Computational Complexity - A Gentle Introduction*. Springer monographs in mathematics. Springer, 2013. doi:10.1007/978-3-319-00119-7.

17 P. Pudlák. Incompleteness in the Finite Domain. *The Bulletin of Symbolic Logic*, 23(4):405–441, 2017.

18 A. A. Razborov. On provably disjoint NP-pairs. *Electronic Colloquium on Computational Complexity (ECCC)*, 1(6), 1994.

Semicomputable Points in Euclidean Spaces

Mathieu Hoyrup
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
mathieu.hoyrup@inria.fr

Donald M. Stull
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
donald.stull@inria.fr

Abstract

We introduce the notion of a semicomputable point in $\mathbb{R}^n$, defined as a point having left-c.e. projections. We study the range of such a point, which is the set of directions on which its projections are left-c.e., and is a convex cone. We provide a thorough study of these notions, proving along the way new results on the computability of convex sets. We prove realization results, by identifying computability properties of convex cones that make them ranges of semicomputable points. We give two applications of the theory. The first one provides a better understanding of the Solovay derivatives. The second one is the investigation of left-c.e. quadratic polynomials. We show that this is, in fact, a particular case of the general theory of semicomputable points.

2012 ACM Subject Classification Theory of computation → Computability

Keywords and phrases Semicomputable point, Left-c.e. real, Convex cone, Solovay reducibility, Genericity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.48

Related Version A full version of the paper is available at `https://hal.inria.fr/hal-02154825v1`.

Funding *Donald M. Stull*: Supported by Inria Nancy Grand-Est.

1 Introduction

The general goal of this paper is to improve our understanding of weak notions of computability in computable analysis. Usually these notions are more difficult to understand than plain computability, and have a rich theory. For instance we mention the notions of computably enumerable (c.e.) subsets of $\mathbb{N}$, left-c.e. reals numbers, left-c.e. real functions, c.e. closed subsets of $\mathbb{R}^n$, co-c.e. closed sets, etc.

A closed subset of $\mathbb{R}^n$ is co-c.e. if its complement is a computable union of rational balls. When a closed set can be described by a few parameters, such as a simple geometrical figure, what properties must these parameters satisfy to make it a co-c.e. closed set? The case of filled triangles has been studied in [5], but the case of disks is more challenging.

A function $f : \mathbb{R} \to \mathbb{R}$ is left-c.e. if there is a program that takes x as input and outputs better and better approximations of $f(x)$ from the left (with no assumption on the speed of convergence to $f(x)$). When a function is described by a few parameters, such as a polynomial, what properties must these parameters satisfy to make it a left-c.e. function?

The cases of co-c.e. disks and left-c.e. polynomials are surprisingly two instances of a common framework that we investigate in this paper. In both cases, the object can be identified with a point in some Euclidean space (for instance, a polynomial is a vector of coefficients) and the computability notion can be expressed as the point having uniformly left-c.e. projections in some set of directions (the directions $(1, X, X^2)$ in the case of quadratic polynomials). This observation leads us to introduce and study the notion of a *semicomputable* point in Euclidean spaces. It is an extension to several dimensions of a notion introduced in [5] in the plane. In particular we define the semicomputability range of a point as the set of directions in which it is left-c.e., and investigate the possible sets that can be obtained as ranges of semicomputable points.

The extension from the plane to higher-dimensional Euclidean spaces is not a straightforward generalization because many subtleties appear in $\mathbb{R}^3$. For instance, the range of a point is a convex cone, so it is determined by two angles in $\mathbb{R}^2$ but can have many different shapes in $\mathbb{R}^3$. Another example is that the operation of taking the conical hull of two convex cones, while simple in $\mathbb{R}^2$, is not as simple in $\mathbb{R}^3$ in terms of computability.

The main results of the paper are realizations results: given a convex cone in $\mathbb{R}^n$ with some computability property, there exists a semicomputable point in $\mathbb{R}^n$ whose range is exactly that cone:

- Theorem 4.6: every Σ^0_2 cone is the range of some semicomputable point. If its closure is not Π^0_2 then the point is *non-uniformly* left-c.e. in the directions of the cone.
- Theorem 4.8: every salient Π^0_2 convex cone is the range of some semicomputable point. Moreover, that point is *uniformly* left-c.e. in the directions of the cone.

In Section 2.4 we give results about computability of convex sets and convex cones. In Section 3 we define semicomputable points of $\mathbb{R}^n$ and develop a thorough study of this notion. In particular we define the range of a semicommputable point, which is the set of directions in which its projections are left-c.e. In Section 4 we prove the main results of the paper, in which we identify classes of convex cones that can be realized as ranges of semicomputable points. In Section 5 we use these results to study Solovay derivatives and precisely identify the possible shapes of the functions $\overline{S}(aX+b,c)$ and $\underline{S}(aX+b,c)$ when a,b,c are fixed and X varies over the computable reals. In Section 6 we investigate the left-c.e. quadratic polynomials, which can be identified with semicomputable points with a certain range.

2 Background

2.1 Computability in Euclidean spaces

We endow $\mathbb{R}^n$ with the inner product $\langle x,y\rangle=\sum_{i=1}^n x_iy_i$, where $x=(x_1,\dots,x_n)$ and $y=(y_1,\dots,y_n)$, the associated norm $\|x\|=\sqrt{\langle x,x\rangle}$ and the distance $d(x,y)=\|x-y\|$. An open set $U\subseteq\mathbb{R}^n$ is ***effectively open*** if it is the union of a computable sequence of rational open balls (centered at rational points with rational radii). Let $A\subseteq\mathbb{R}^n$ be a closed set. A is ***c.e. closed*** if A contains a dense computable sequence, or equivalently the function $x\mapsto d(x,A)=\min_{y\in A}d(x,y)$ is right-c.e. A is ***co-c.e. closed*** if the complement of A is effectively open, or equivalently the function $x\mapsto d(x,A)$ is left-c.e. A closed set is ***computably closed*** if it is both c.e. closed and co-c.e. closed. A compact set K is ***effectively compact*** if the set of finite lists of rational balls covering K is c.e. A compact set is effectively compact if and only if it is co-c.e. closed. More details on these notions can be found in [3].

A real number is ***left-c.e.*** if it is the limit of a computable increasing sequence of rational numbers. A real number x is ***right-c.e.*** if $-x$ is left-c.e. It is computable if it is both left-c.e. and right-c.e. If $D\subseteq\mathbb{R}^n$ then a function $f:D\to[-\infty,+\infty]$ is left-c.e. if there exist uniformly effective open sets $(U_q)_{q\in\mathbb{Q}}$ such that for all $q\in\mathbb{Q}$, $f^{-1}(q,+\infty)=D\cap U_q$. f is right-c.e. if $-f$ is left-c.e.

Let $f:\mathbb{R}^m\times\mathbb{R}^n$ be left-c.e.. If $K\subseteq\mathbb{R}^n$ is effectively compact then $f_{\min}:\mathbb{R}^m\to\mathbb{R}$ defined by $f_{\min}(x)=\min_{y\in K}f(x,y)$ is left-c.e. If $A\subseteq\mathbb{R}^n$ is c.e. closed then $f_{\sup}:\mathbb{R}^m\to\mathbb{R}$ defined by $f_{\sup}(x)=\sup_{y\in A}f(x,y)$ is left-c.e.

Each of these computability notions can be relativized to any oracle. We will be particularly interested in their relativization to the halting set, denoted by $\emptyset'$. For instance, a real is $\emptyset'$-left-c.e. if it is left-c.e. relative to $\emptyset'$.

2.2 Solovay derivatives

The quantitative study of Solovay reducibility was initiated in [1] and continued in [7] and [5]. We briefly recall that if a, b are real numbers such that b is left-c.e., then we define

$$\overline{S}(a,b) = \inf\{q \in \mathbb{Q} : qb - a \text{ is left-c.e.}\},$$
$$\underline{S}(a,b) = \sup\{q \in \mathbb{Q} : qb - a \text{ is right-c.e.}\}.$$

We say that a is ***Solovay reducible*** to b if $\overline{S}(a,b) < +\infty$ and $\underline{S}(a,b) > -\infty$. Intuitively, it means that a is easier to approximate than b in the following sense: if $\overline{S}(a,b) < q$ and $\underline{S}(a,b) > r$, then there exist computable sequences a_i, b_i converging to a, b such that $r \leq \frac{a-a_i}{b-b_i} \leq q$.

Some left-c.e. real numbers are ***Solovay complete***, meaning that each left-c.e. number is reducible to them, and it is proved in [1] that if b is Solovay complete, then $\overline{S}(a,b) = \underline{S}(a,b)$.

2.3 Background on convex cones

We give the minimal amount of background on convex analysis and refer the reader to [2] for more details. A ***cone*** is a set $C \subseteq \mathbb{R}^n$ that is closed under multiplication by a nonnegative scalar. A ***convex cone*** is a cone that is convex, i.e. a set that is closed under addition and multiplication by a nonnegative scalar. The ***dual*** of a set C is the closed convex cone $C^* = \{x \in \mathbb{R}^n : \forall y \in C, \langle x, y\rangle \geq 0\}$. $(C^*)^*$ is the smallest closed convex cone containing C. In particular if C is a closed convex cone then $(C^*)^* = C$.

For $x \neq 0$, let $H_x = \{z : \langle x, z\rangle \geq 0\}$ be the half-space delimited by the hyperplane orthogonal to x, in the direction of x. One has $C^* = \bigcap_{x\in C} H_x$. As a result, $d(z, C^*) \geq \sup_{x\in C} d(z, H_x)$ and we show that equality holds. Observe that $d(z, H_x) = \max(-\frac{\langle z,x\rangle}{\|x\|}, 0)$.

▶ **Lemma 2.1.** *Let C be a convex set. One has $d(z, C^*) = \sup_{x\in C} d(z, H_x)$.*

For a convex cone C, let C_1 be the intersection of C with the unit sphere. In the previous lemma, one has $d(z, C^*) = \sup_{x\in C_1} d(z, H_x)$ if C is a convex cone.

A convex cone is ***flat*** if it contains some $x \neq 0$ and its opposite $-x$. It is called ***salient*** if it is not flat. C is salient if and only if C^* is full-dimensional if and only if there exist $\epsilon > 0$ and y such that $\langle x, y\rangle > \epsilon$ for all $x \in C_1$.

If $A \subseteq \mathbb{R}^n$ is a full-dimensional convex set, then $A \subseteq \overline{\mathrm{int}(A)}$ and $\mathrm{int}(\overline{A}) \subseteq A$. In particular, among the class of full-dimensional convex sets, every closed set is regular closed and every open set is regular open.

2.4 Computability of convex sets and cones

Computability of convex sets has been investigated in [6] and [9]. Here we present new results that are used to prove the results of the paper and are of independent interest.

▶ **Proposition 2.2.** *Let $C \subseteq \mathbb{R}^n$ be a closed convex cone.*

- *C is co-c.e. closed $\iff$ C^* is c.e. closed,*
- *C is c.e. closed $\iff$ C^* is co-c.e. closed,*
- *C is computably closed $\iff$ C^* is computably closed.*

Proof. If C is c.e. closed then let $(x_i)_{i\in\mathbb{N}}$ be a dense computable sequence in C. One has $C^* = \bigcap_i H_{x_i}$ which is therefore co-c.e. closed.

If C is co-c.e. then the intersection C_1 of C with the unit sphere is effectively compact. By Lemma 2.1 one has $d(z, C^*) = \max_{x\in C} d(z, H_x) = \max_{x\in C_1} d(z, H_x)$ which is a right-c.e. function of z, so C^* is c.e. closed.

The other implications can be obtained by observing that $(C^*)^* = C$. ◀

Observe that these results relativize to any oracle. The first equivalence in the next result was proved by Ziegler [9].

▶ **Proposition 2.3.** *Let $C \subseteq \mathbb{R}^n$ be a full-dimensional closed convex set.*

- *C is co-c.e. closed $\iff$ the set of rational points outside C is c.e.,*
- *C is c.e. closed $\iff$ its interior is effectively open.*

Proof. If C is co-c.e. closed then the set of rational points outside C is obviously c.e. Conversely, assume that the set of rational points outside C is c.e. Let $C_0 \subseteq C$ be any fixed full-dimensional rational polytope. Given $z \in \mathbb{R}^n$, one can compute the convex hull of $C_0 \cup \{z\}$ and in particular enumerate its interior U_z. As U_z is dense in that convex hull, one has $z \notin C \iff U_z$ contains a rational point outside C. It gives a procedure that given z, halts exactly when $z \notin C$, showing that C is co-c.e.

If the interior of C is effectively open then one can enumerate the rational points in the interior, which are dense in C. Conversely, if C is c.e. closed then let $(x_i)_{i\in\mathbb{N}}$ be a dense computable sequence in C. A point z belongs to the interior of C iff there exist $n+1$ points in the sequence such that z belongs to the interior of their convex hull, which gives a procedure that halts exactly when $z \in \text{int}(C)$. ◀

The assumption that C is full-dimensional is needed. For the first item, if C contains no rational point then no information about C can be obtained from an enumeration of the rationals ouside C (i.e., all the rational points). For the second item, C needs to have a non-empty interior.

▶ **Lemma 2.4.** *Let $A, B \subseteq \mathbb{R}^n$ be c.e. closed convex sets.*

- *If $A \cap B$ has non-empty interior then $A \cap B$ is c.e. closed.*
- *$A \cap B$ is $\emptyset'$-co-c.e. closed. There exist $A, B \subseteq \mathbb{R}^3$ such that $A \cap B$ is not $\emptyset'$-c.e. closed.*

Proof. The interiors of A and B are effectively open and dense in A and B respectively. Their intersection is effectively open and dense in $A \cap B$, which is then c.e. closed.

In general, if A, B are c.e. closed then they are $\emptyset'$-computable and in particular $\emptyset'$-co-c.e. closed and so is their intersection.

There exists a right-c.e. convex function $f : \mathbb{R} \to \mathbb{R}$ such that $f^{-1}(0)$ is not $\emptyset'$-c.e. closed. Let $\alpha > 0$ be $\emptyset'$-right-c.e. but not $\emptyset'$-left-c.e. There exists a sequence of uniformly left-c.e. reals $\alpha_i > 0$ such that $\alpha = \inf_i \alpha_i$. Let $f_i(x) = \max(0, x - \alpha_i)$ and $f = \sum_i 2^{-i} f_i$. The functions f_i are uniformly right-c.e. so f is right-c.e., and $f^{-1}(0) = [0, \alpha]$ is not $\emptyset'$-c.e. closed because α is not $\emptyset'$-left-c.e.

Let $A = \{(x, y) : y \geq f(x)\}$ be the epigraph of f and $B = \{(x, y) : y \leq 0\}$. A and B are c.e. closed but $A \cap B = \{(x, 0) : f(x) = 0\}$ is not $\emptyset'$-c.e. closed. ◀

▶ **Proposition 2.5.** *Let $C \subseteq \mathbb{R}^n$ be a full-dimensional closed convex set.*

- *C is $\emptyset'$-co-c.e. closed $\iff$ the set of rational points outside C is $\emptyset'$-c.e. $\iff$ C is Π^0_2,*
- *C is $\emptyset'$-c.e. closed $\iff$ its interior is $\emptyset'$-effectively open $\iff$ C contains a dense Σ^0_2-set.*

Proof. Several equivalences are obtained by relativizing Proposition 2.3, we prove the others. Any set that is $\emptyset'$-co-c.e. is Π^0_2, and if C is Π^0_2 then the set of rational points ouside C is obviously $\emptyset'$-c.e.

Any $\emptyset'$-effectively open set is a Σ^0_2-set, and $\mathrm{int}(C)$ is dense in C. If C contains a dense Σ^0_2-set, then, with $\emptyset'$ as oracle, one can compute a dense sequence in that set, so C is $\emptyset'$-c.e. closed. ◀

Again the full dimension assumption is needed. For the first item, there exists a Π^0_2-singleton whose unique element is not computable relative to $\emptyset'$ (even relative to any $\emptyset^{(n)}$, $n \in \mathbb{N}$, see Proposition 1.8.62 in [8]). For the second item, if x is $\emptyset'$-computable but not computable, then $\{x\}$ is $\emptyset'$-c.e. closed convex but does not contain any non-empty Σ^0_2-set.

3 Semicomputable point

The notions of left-c.e. and right-c.e. real number can be extended to higher dimensions. A first extension to points of the plane has been introduced in [5]. We pursue this extension to $\mathbb{R}^n$ for any $n \geq 1$. Although the definition extends immediately to this more general setting, the results are more involved because higher dimensions allow richer behaviors. For instance, a convex cone in $\mathbb{R}^2$ is delimited by two directions only, while a convex cone in $\mathbb{R}^3$ is a delimited by an uncountable set of directions.

▶ **Definition 3.1.** *A point $x \in \mathbb{R}^n$ is* **semicomputable** *if there exist n linearly independent rational vectors $v_1, \dots, v_n$ such that each $\langle v_i, x\rangle$ is left-c.e., $1 \leq i \leq n$.*

▶ **Definition 3.2.** *Let $D \subseteq \mathbb{R}^n$ be a closed convex cone. We say that $x \in \mathbb{R}^n$ is D***-c.e.** *if the mapping $d \in D \mapsto \langle d, x\rangle$ is left-c.e.*

Observe that this notion really makes sense when D is full-dimensional (or full-dimensional in some computable subspace), otherwise x could be D-c.e. only because the elements of D encode information about x. For instance, if $\|x\|$ is left-c.e. and $D = \{\lambda x : \lambda \geq 0\}$ then the mapping $d \in D \mapsto \langle d, x\rangle = \|d\| \cdot \|x\|$ is left-c.e., which should not be interpreted as "x is left-c.e. in some direction".

The closedness condition on D is justified by the next observation: D can always be assumed to be closed.

▶ **Proposition 3.3.** *Let $x \in \mathbb{R}^n$ and $D \subseteq \mathbb{R}^n$ be a full-dimensional convex cone. If the mapping $d \in D \mapsto \langle d, x\rangle$ is left-c.e. then x is $\overline{D}$-c.e.*

Proof. Let $d_0 \in D$ be a rational vector in the interior of D. Let $d \in \overline{D}$ be given as oracle. The vectors $d_n = (1 - 2^{-n})d + 2^{-n}d_0$ are uniformly computable in d and belong to D. The number $\langle d, x\rangle$ is the effective limit of $\langle d_n, x\rangle$, which is left-c.e. uniformly in n and d, so $\langle d, x\rangle$ is left-c.e. uniformly in d. ◀

Being C^*-c.e. can be dually expressed in terms of C.

▶ **Proposition 3.4.** *Let $x \in \mathbb{R}^n$ and $C \subseteq \mathbb{R}^n$ be a closed convex cone.*

- *When C is co-c.e. closed, x is C^*-c.e. $\iff$ $x + C$ is co-c.e. closed,*
- *When C is c.e. closed and full-dimensional, x is C^*-c.e. $\iff$ $x - C$ is c.e. closed.*

Proof.

- If x is C^*-c.e. then the complement of $x + C$ is effectively open. Indeed, $y \notin x + C \iff y - x \notin C \iff \inf_{d \in C^*} \langle d, y - x\rangle < 0$ which is a c.e. condition as C^* is c.e. closed and $\langle d, y - x\rangle$ is right c.e. in d, y.

If $x + C$ is co-c.e. then let $K \in \mathbb{N}$ be an upper bound on $\|x\|$ and $A = (x + C) \cap \overline{B}(0, K)$. A is effectively compact, contains x and for $d \in C^*$, $\langle d, x\rangle = \min_{z\in A} \langle d, z\rangle$ which is a left-c.e. function of d.

- If x is C^*-c.e. then $\mathrm{int}(x - C)$ is effectively open. Indeed $y \in \mathrm{int}(x - C) \iff x - y \in \mathrm{int}(C) \iff \min_{d\in C_1^*} \langle d, x - y\rangle > 0$ which is a c.e. condition as C_1^* is effectively compact and $\langle d, x - y\rangle$ is left-c.e. in d, y.

  If $x - C$ is c.e. closed and $(x_i)_{i\in\mathbb{N}}$ is a dense computable sequence in $x - C$, then for $d \in C^*$ one has $\langle d, x\rangle = \sup_i \langle d, x_i\rangle$ which is left-c.e. uniformly in d. ◀

▶ **Proposition 3.5.** *Let $C \subseteq \mathbb{R}^n$ be a closed convex cone.*

- *If $x + C$ is co-c.e. closed for some $x \in \mathbb{R}^n$, then C is co-c.e. closed.*
- *If $x + C$ is c.e. closed for some $x \in \mathbb{R}^n$, then C is c.e. closed.*

Proof. Let $E \subseteq \mathbb{R}^n$ be a set such that 0 belongs to the convex hull of E. One has $C^* = \bigcup_{e\in E}(C + e)^*$. Indeed, if $y \in C^*$ then there exists $e \in E$ such that $\langle e, y\rangle \geq 0$, so $y \in (C + e)^*$. Conversely, if $y \in (C + e)^*$ and $c \in C$ then $\langle y, c\rangle = \lim_{n\to\infty} \frac{1}{n}\langle y, e + nc\rangle \geq 0$ so $y \in C^*$.

Given x, there exists a finite set E of rational points such that the convex hull of $x + E$ contains 0. As a result, $C^* = \bigcup_{e\in E}(C+x+e)^*$. If $C+x$ is co-c.e. closed then each $(C+x+e)^*$ is c.e. closed so C^* is c.e. closed, hence C is co-c.e. closed. If $C + x$ is c.e. closed then each $(C + x + e)^*$ is co-c.e. closed so C^* is co-c.e. closed, hence C is c.e. closed. ◀

It was proved in [7] and [5] that if f is computable and differentiable at c then $\underline{S}(f(c), c) = \overline{S}(f(c), c) = f'(c)$. If $x = (c, f(c))$ and $v = (1, f'(c))$ then it means that $\langle d, x\rangle$ is left-c.e. for all rational directions d such that $\langle d, v\rangle > 0$. We now investigate when this is uniform in d, i.e. when x is $\{v\}^*$-c.e.

▶ **Proposition 3.6** (Positive case)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ be computable and convex or concave. If $c \in \mathbb{R}$ is left-c.e. and $x = (c, f(c))$ and $v = (1, f'_-(c))$ where $f'_-(c)$ is the left-derivative of f at c, then x is $\{v\}^*$-c.e.*

Proof. Assume that f is convex, the other case is obtained by considering $-f$. Let c_i be a computable increasing sequence converging to x. Let q, r be rational numbers such that $r < f'_-(c) < q$. Compute i such that $\frac{f(c_{i+1})-f(c_i)}{c_{i+1}-c_i} > r$. For $j \geq i$ one has $r < \frac{f(c)-f(c_j)}{c-c_j} \leq f'(c) < q$, so $rc - f(c) = \inf_{j\geq i} rc_j - f(c_j)$ and $qc - f(c) = \sup_{j\geq i} qc_j - f(c_j)$ are respectively right-c.e. and left-c.e., uniformly in r and q. ◀

▶ **Proposition 3.7** (Negative case)**.** *Let $c \in \mathbb{R}$ be left-c.e. and $f : \mathbb{R} \to \mathbb{R}$ be computable and such that $f'(c) = 0$ and $f(c)$ is not right-c.e. Let $x = (c, f(c))$ and $v = (1, f'(c)) = (1, 0)$. x is not $\{v\}^*$-c.e.*

Proof. Simply take $d = (0, -1) \in \{v\}^*$. d is computable but $\langle d, x\rangle = -f(c)$ is not left-c.e. ◀

Said differently, in that case $qc - f(c)$ is non-uniformly left-c.e. for rationals $q > 0$.

3.1 Converging sequences

We may equivalently define semicomputable points to be those points which are the limit of a computable sequence converging in some restricted region of the space, namely a salient convex cone. There is a precise relation between the cones where such sequences can live and the cones of directions in which the point is left-c.e.

The first observation is straightforward.

▶ **Proposition 3.8.** *Let $x \in \mathbb{R}^n$ and $C \subseteq \mathbb{R}^n$ be a convex cone. If there exists a computable sequence x_i converging to x in $x - C$, then x is C^*-c.e.*

Proof. If $d \in C^*$ then $\langle d, x - x_i \rangle \geq 0$ so $\langle d, x \rangle = \sup_i \langle d, x_i \rangle$ is left-c.e. uniformly in d. ◀

In general it is not an equivalence. However when C is c.e. closed, or equivalently when C^* is co-c.e. closed, the equivalence holds.

▶ **Proposition 3.9.** *Let $x \in \mathbb{R}^n$ and $C \subseteq \mathbb{R}^n$ be a salient c.e. closed convex cone. x is C^*-c.e. $\iff$ there exists a computable sequence converging to x in $x - C$.*

Proof. Assume that x is C^*-c.e. The interior of $x - C$ is an effective open set. Indeed, y belongs to that set iff $\min_{d \in C_1^*} \langle d, x - y \rangle > 0$, which is a c.e. condition as C_1^* is effectively compact. As a result, there is a computable enumeration $(y_i)_{i \in \mathbb{N}}$ of the rational vectors in that set. Define a computable sequence $(x_i)_{i \in \mathbb{N}}$ as follows: take $x_{i+1} \in \int(x - C)$ such that $y_0, \dots, y_i \prec x_{i+1}$.

As C is salient, the growing sequence x_i converges to a point in $x - C$. As it eventually exceeds each y_i, the limit must be x. ◀

3.2 Taking unions of convex cones

In $\mathbb{R}^2$, let P, Q be full-dimensional closed convex cones and R be the conical hull of $P \cup Q$. If $x \in \mathbb{R}^2$ is P-c.e. and Q-c.e., then x is R-c.e. However we will see below (Theorem 3.12) that this property fails in higher dimensions. We first show that it can be recovered under computability assumptions on P, Q.

▶ **Proposition 3.10.** *Let $P, Q \subseteq \mathbb{R}^n$ be closed convex cones, R be the conical hull of $P \cup Q$ and $x \in \mathbb{R}^n$ be P-c.e. and Q-c.e.*

- *If P and Q are c.e. closed then R is c.e. closed and x is R-c.e.,*
- *If P and Q are co-c.e. closed and R is salient, then R is co-c.e. closed and x is R-c.e.*

In the second statement, the condition that R is salient is needed otherwise the complexity of R can increase, as we now show.

▶ **Proposition 3.11.** *If $P, Q \subseteq \mathbb{R}^n$ are co-c.e. closed convex cones and R is the convex cone induced by $P \cup Q$, then R is $\emptyset'$-c.e. closed. In dimension $n \geq 3$ one can take P, Q so that R is not $\emptyset'$-co-c.e. closed.*

The proof essentially uses Lemma 2.4. Indeed, P^* and Q^* are c.e. closed and $R^* = P^* \cap Q^*$. One can embed the convex sets A, B from Lemma 2.4 in $\mathbb{R}^3$ and take their conical hulls.

We will see that the second item fails when R is not salient (Corollary 4.7). We now build a simpler example without the co-c.e. assumption about P and Q.

▶ **Theorem 3.12.** *In dimension $n \geq 3$, there exist closed convex cones $P, Q \subseteq \mathbb{R}^n$ and a point $x \in \mathbb{R}^n$ that is P-c.e. and Q-c.e. but not R-c.e., where R is the conical hull of $P \cup Q$.*

The idea of the proof is to build a, b, c such that $qc - a$ and $rc - b$ are *uniformly* left-c.e. for $q > \overline{S}(a, c)$ and $r > \overline{S}(b, c)$, but $sc - (a + b)$ is *non-uniformly* left-c.e. for $q > \overline{S}(a + b, c)$. To do this, we take $a = f(c)$ ang $b = g(c)$ where f, g satisfy the conditions of Proposition 3.6 but $f + g$ satisfies the conditions of Proposition 3.7.

3.3 Semicomputability range of a point

▶ **Definition 3.13.** *If $x \in \mathbb{R}^n$ then its* **semicomputability range***, or simply* **range***, is the set of computable $d \in \mathbb{R}^n$ such that $\langle d, x\rangle$ is left-c.e., and is denoted by* $\text{range}(x)$*.*

One of the main goals of the paper is to investigate the following problem.

▶ **Problem 1.** *What sets can be realized as* $\text{range}(x)$ *for some x?*

Let $\mathbb{R}_c$ be the field of computable real numbers. From the definition we see that $\text{range}(x)$ contains computable points from $\mathbb{R}^n_c$ only. By abuse of language, when we write $A \subseteq \text{range}(x)$ we mean $A \cap \mathbb{R}^n_c \subseteq \text{range}(x)$, and similarly, $\text{range}(x) = A$ means $\text{range}(x) = A \cap \mathbb{R}^n_c$. The interior of $\text{range}(x)$ is meant to be the interior of $\text{range}(x)$ in the subspace topology on $\mathbb{R}^n_c$.

▶ **Proposition 3.14.** *Let $x \in \mathbb{R}^n$:*

1. $\text{range}(x)$ *is a convex cone over the field* $\mathbb{R}_c$*.*
2. *If $D \subseteq \mathbb{R}^n$ is a closed polygonal convex cone with computable coordinates, then x is D-c.e. $\iff$ $D \subseteq \text{range}(x)$.*
3. *If $D \subseteq \mathbb{R}^n$ be a closed convex cone contained in the interior of* $\text{range}(x)$*, then x is D-c.e.*

Proof.

1. Straightforward.
2. x is D-c.e. $\iff$ x is d-c.e. for each extreme direction $d \in D$ $\iff$ each such direction belongs to $\text{range}(x)$.
3. There exists a rational polygonal convex cone E containing D and contained in $\text{range}(x)$. By 2., x is E-c.e. hence D-c.e. ◀

We will see that $\text{range}(x)$ is not necessarily closed (in the subspace $\mathbb{R}^n_c$), and that the third item sometimes fails when D is just contained in $\text{range}(x)$ (Theorem 4.6).

▶ **Proposition 3.15.** *Let $x \in \mathbb{R}^n$ be a semicomputable point. Let $D \subseteq \mathbb{R}^n$ be a closed convex cone such that x is D-c.e. and* $\text{range}(x) = D$*. Then D is $\emptyset'$-co-c.e. closed.*

Proof. Let M be a machine that given a rational point $d \in D$ approximates $\langle d, x\rangle$ from the left. With $\emptyset'$ as oracle, given a rational point d, one can compute x, $\langle d, x\rangle$ and $M(d)$ and eventually see whether $M(d) \neq \langle d, x\rangle$, which means that $d \notin D$. As a result, the set of rational points outside D is c.e. relative to $\emptyset'$ so D is Π^0_2 by Proposition 2.5. ◀

We will see that this is tight: every $\emptyset'$-co-c.e. closed convex cone can be obtained (Theorem 4.8).

3.4 Solovay complete coordinates

When one of the coordinates of $x \in \mathbb{R}^n$ is Solovay complete, the range of x is easily described.

▶ **Proposition 3.16.** *Let $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ where x_1 is Solovay complete. Let $v = (1, S(x_2, x_1), \dots, S(x_n, x_1))$. One has $\overline{\text{range}(x)} = \{v\}^*$.*

For a closed convex cone C,

$$v \in \text{int}(C) \implies x \text{ is } C^*\text{-c.e.} \implies v \in C.$$

Proof. A computable sequence x_i converging to x must asymptotically converge along the direction v, for each rational d such that $\langle d, v\rangle > 0$, one eventually has $\langle d, x - x_i\rangle > 0$, so $\langle d, x\rangle$ is left-c.e. The set of such vectors d is dense in $\{v\}^*$.

If v belongs to the interior of C then C^* is contained in the interior of $\{v\}^* = \overline{\mathrm{range}(x)}$ so x is C^*-c.e. by Proposition 3.14 item 3. If x is C^*-c.e. then $C^* \subseteq \mathrm{range}(x) \subseteq \{v\}^*$, i.e. $v \in C$. ◀

In particular, if d is a computable vector such that $\langle d, v\rangle > 0$, then $\langle d, x\rangle$ is left-c.e.

4 Realizing convex cones

In this section we investigate the possible ranges of semicomputable points. In order to realize a given convex cone D, we build a point that is left-c.e. along each computable direction in D, and no more. To do so, we make the point *generic* in some sense. Let us briefly recall from [4] the notion of genericity that we need.

4.1 Genericity

▶ **Definition 4.1.** *Let $A \subseteq \mathbb{R}^n$. A point $x \in A$ is* **generic inside** *A if for every effective open set $U \subseteq \mathbb{R}^n$, either $x \in U$ or there exists a neighborhood B of x such that $B \cap U \cap A = \emptyset$.*

▶ **Example 4.2.**

- Taking $A = X$, being generic inside X amounts to being 1-generic,
- Every x is obviously generic inside $\{x\}$,
- In the space of real numbers with the Euclidean topology, a real number $x \in (0, 1)$ is said to be right-generic if x is generic inside $[x, 1]$,

The last example is a particular instance of the following general situation.

If τ' is a weaker topology on X then we define $S(x)$ as the intersection of the τ'-open sets containing x. Equivalently, $S(x) = \{y \in X : x \leq_{\tau'} y\}$ where $\leq_{\tau'}$ is the specialization pre-order defined by $x \leq y$ iff every τ'-neighborhood of x contains y.

Let τ be the Euclidean topology on $\mathbb{R}^n$.

▶ **Theorem 4.3** (Theorem 4.1.1 in [4]). *Let τ' a topology that is effectively weaker than τ, such that emptiness of finite intersections of basic open sets in τ and τ' is decidable. There exists a point x that is computable in $(\mathbb{R}^n, \tau')$ and generic inside $S(x)$.*

For instance, the topology τ' generated by the semi-lines $(q, +\infty)$ is effectively weaker than τ, and its specialization pre-order is the natural ordering $\leq$ on $\mathbb{R}$. Theorem 4.3 implies the existence of right-generic left-c.e. reals.

▶ **Proposition 4.4.** *Let $C \subseteq \mathbb{R}^n$ be a closed convex cone. If x is generic inside $x + C$ then* $\mathrm{range}(x) \subseteq C^*$*.*

Proof. Let $d \notin C^*$ be computable and assume that $\alpha := \langle d, x\rangle$ is left-c.e. The set $U = \{y : \langle d, y\rangle < \alpha\}$ is effectively open and $x \notin U$. As $d \notin C^*$ there exists $c \in C$ such that $\langle d, c\rangle < 0$. For $\epsilon > 0$, $\langle d, x + \epsilon c\rangle < \langle d, x\rangle = \alpha$ so $x + \epsilon c \in U \cap (x + C)$. As a result, x belongs to the closure of $U \cap (x + C)$ so x is not generic inside $x + C$. ◀

In particular, if x is C^*-c.e. and generic inside $x + C$ then $\mathrm{range}(x) = C^*$.

4.2 Realizing convex cones

Theorem 4.3 can now be applied to obtain a first class of cones realized as ranges of points.

▶ **Theorem 4.5** (C.e. closed cones). *Let $C \subseteq \mathbb{R}^n$ be a co-c.e. closed convex cone. There exists x that is C^*-c.e. and generic inside $x + C$, hence* $\mathrm{range}(x) = C^*$.

Proof. C^* is c.e. closed, llet $d_i \in C^*$ be a computable dense sequence. Consider the topology τ' generated by the basic open sets $U_{i,j} = \{x : \langle d_i, x\rangle > q_j\}$, where $(q_j)_{j\in\mathbb{N}}$ is some computable enumeration of the positive rational numbers. One easily checks that emptiness of finite intersections of basic open sets in τ, τ' is decidable, so we can apply Theorem 4.3. We obtain a point x that is computable in $(\mathbb{R}^n, \tau')$, i.e. the numbers $\langle d_i, x\rangle$ are uniformly left-c.e. hence x is C^*-c.e. and $C^* \subseteq \mathrm{range}(x)$. Moreover x is generic inside $S(x) = x + C$, hence $\mathrm{range}(x) \subseteq C^*$ by Proposition 4.4. ◀

Therefore, any c.e. closed convex cone can be realized as the range of a point. We can extend the result to other classes of closed convex cones. To do so, we need to refine the construction techniques.

▶ **Theorem 4.6** (Σ^0_2 cones). *Let $(D_k)_{k\in\mathbb{N}}$ be a growing sequence of uniformly co-c.e. closed convex cones in $\mathbb{R}^n$. There exists x such that for any co-c.e. closed convex cone K, x is K-c.e. $\iff$ K is contained in some D_k. In particular,* $\mathrm{range}(x) = \bigcup_k D_k$.

In particular, any $\emptyset'$-effectively open convex cone is the range of a point.

We can use this result to give a counter-example to Proposition 3.10 when the cone is not salient.

▶ **Corollary 4.7.** *There exists co-c.e. closed convex cones $P, Q \subseteq \mathbb{R}^3$ and a point x that is P-c.e. and Q-c.e. but not R-c.e., where R is the convex cone induced by $P \cup Q$.*

Proof. Take P, Q from Proposition 3.11. The induced cone R is $\emptyset'$-c.e. closed but not $\emptyset'$-co-c.e. closed. By Proposition 2.3, R contains a dense Σ^0_2-set R', and we can assume that R' contains P and Q (otherwise replace R' with $R' \cup P \cup Q$). By Theorem 4.6 there exists x such that $\mathrm{range}(x) = R'$, x is P-c.e. and Q-c.e. But x is not R-c.e., otherwise R would be $\emptyset'$-co-c.e. closed by Proposition 3.15. ◀

▶ **Theorem 4.8** (Π^0_2 cones). *Let $D \subseteq \mathbb{R}^n$ be a salient Π^0_2 convex cone. There exists x that is D-c.e. and such that* $\mathrm{range}(x) = D$.

4.3 Beyond linear maps

If x is C^*-c.e. and generic inside $x + C$ then we know for which computable linear maps $f : \mathbb{R}^n \to \mathbb{R}$ the number $f(x)$ is left-c.e.: exactly when $f \in C^*$ (f can be identified with the vector v such that $f(x) = \langle v, x\rangle$).

Genericity has also consequences on functions $f : \mathbb{R}^n \to \mathbb{R}$ that are not linear but totally differentiable. We recall that if f is ***totally differentiable*** at x then there exists a vector $\mathrm{grad}f(x)$ such that $f(x + h) = f(x) + \langle \mathrm{grad}f(x), h\rangle + o(h)$.

▶ **Proposition 4.9.** *Let $C \subseteq \mathbb{R}^n$ be a closed convex cone. Let $x \in \mathbb{R}^n$ be C^*-c.e. and generic inside $x + C$. Let $f : \mathbb{R}^n \to \mathbb{R}$ be computable and totally differentiable at x.*

- *If* $\mathrm{grad}f(x) \in \mathrm{int}(C^*)$ *then $f(x)$ is left-c.e.*
- *If* $\mathrm{grad}f(x) \notin C^*$ *then $f(x)$ is not left-c.e.*

Proof. Let $D^* \subseteq \mathrm{int}(C^*)$ be a computable polygonal convex salient cone containing $\mathrm{grad} f(x)$ in its interior. There exists $\delta > 0$ such that $\langle \mathrm{grad} f(x), d\rangle > \delta$ for all $d \in D_1$. As x is D^*-c.e., there exists a computable sequence $x_i \in x - D$ converging to x by Proposition 3.9. Therefore one has $f(x_i) = f(x) - \langle \mathrm{grad} f(x), (x - x_i)\rangle + o(x - x_i) < f(x) - \|x - x_i\|\, \delta + o(x - x_i) < f(x)$ for i larger than some i_0, so $f(x) = \sup_{i \geq i_0} f(x_i)$ is left-c.e.

Assume that $\mathrm{grad} f(x) \notin C^*$ and that $\alpha := f(x)$ is left-c.e. The set $U = \{y : f(y) < \alpha\}$ is effectively open. As $\mathrm{grad} f(x) \notin C^*$, there exists $c \in C$ such that $\langle \mathrm{grad} f(x), c\rangle < 0$. One has $f(x + \epsilon c) = f(x) + \epsilon\langle \mathrm{grad} f(x), c\rangle + o(\epsilon) < f(x)$ for sufficiently small ϵ, so $x + \epsilon c \in U \cap (x + C)$. Therefore $x \notin U$ and belongs to the closure of $(x + C) \cap U$, contradicting the assumption that x is generic inside $x + C$. ◀

5 Application to the Solovay derivatives

We pursue the study of the Solovay derivatives $\overline{S}(a, b)$ and $\underline{S}(a, b)$ started in [5] in the general case, i.e. without assuming that b is Solovay complete. The general goal is to find ways to calculate $\underline{S}(a, b)$ and $\overline{S}(a, b)$ for given a, b. Although formulae are available in some cases, we investigate one of the simplest situations in which no general formula exists:

▶ **Problem 2.** *If $a, b, c \in \mathbb{R}$ are fixed, what can be the shapes of the functions $\overline{S}(aX + b, c)$ and $\underline{S}(aX + b, c)$, where X varies among the computable real numbers?*

When c is Solovay complete, one has $S(x, c) := \overline{S}(x, c) = \underline{S}(x, c)$ for any d-c.e. x and

$$S(aX + b, c) = S(a, c)X + S(b, c).$$

However in general only inequalities can be derived (see [5]):

If $X \geq 0$,

$$\overline{S}(aX + b, c) \leq \overline{S}(a, c)X + \overline{S}(b, c)$$
$$\underline{S}(aX + b, c) \geq \underline{S}(a, c)X + \underline{S}(b, c).$$

If $X \leq 0$,

$$\overline{S}(aX + b, c) \leq \underline{S}(a, c)X + \overline{S}(b, c)$$
$$\underline{S}(aX + b, c)) \geq \overline{S}(a, c)X + \underline{S}(b, c).$$

It seems at first that these two functions of X should be very rigid because a, b, c are fixed, so their local shape should not depend too much on X. However, we will see that, up to some geometrical contraints, they can have a wide variety of possible shapes. Fortunately, we can use the notions and results of this paper to precisely identify the class of possible shapes of these two functions. The idea is geometrical: these functions can be read in some way from the convex cone $\mathrm{range}(x)$, where $x = (a, b, c)$. Therefore their shapes are precisely the shapes that can be obtained from arbitrary convex cones. Let $\overline{\mathbb{R}} = [-\infty, +\infty]$.

▶ **Definition 5.1.** *Let $\mathcal{F}$ be the family of pairs of functions (f, g) from $\mathbb{R}$ to $\overline{\mathbb{R}}$ such that:*

- *$f \geq g$,*
- *f is convex and g is concave (i.e., the epigraphs of f and $-g$ are convex sets),*
- *Every line segment joining the graph of f to the graph of g lies below the graph of f and above the graph of g.*

The third condition implies that $\lim_{x\to-\infty} f'(x) = \lim_{x\to+\infty} g'(x)$ and $\lim_{x\to+\infty} f'(x) = \lim_{x\to-\infty} g'(x)$. Examples of such pairs are: $f(X) = -g(X) = \sqrt{1 + X^2}$, or $f(X) = X^2$ and $g(X) = -\infty$.

The main result of this section is that $\mathcal{F}$ captures essentially the possible shapes of $(\overline{S}(aX + b, c), \underline{S}(aX + b, c))$, up to computability conditions.

▶ **Theorem 5.2.** *Let $a, b, c \in \mathbb{R}$ with c left-c.e. and non-computable. One has $(\overline{S}(aX + b, c), \underline{S}(aX + b, c)) \in \mathcal{F}$. Conversely,*

- *Any pair $(f, g) \in \mathcal{F}$ where f is $\emptyset'$-left-c.e. and g is $\emptyset'$-right-c.e. can be realized,*
- *Any pair $(f, g) \in \mathcal{F}$ where f is $\emptyset'$-right-c.e. and g is $\emptyset'$-left-c.e. can be realized,*

To prove this result we show that the pairs $(f, g) \in \mathcal{F}$ are exactly the functions that can be read on convex cones in $\mathbb{R}^3$ in the following way: given a cone C in $\mathbb{R}^3$, the intersection of C with the planes $y = \pm 1$ convex sets, and the curves delimiting them are exactly the pairs $(f, g) \in \mathcal{F}$.

Now, if $x = (a, b, c) \in \mathbb{R}^*$ then the pair $(\overline{S}(aX + b, c), \underline{S}(aX + b, c))$ is obtained in this way from the cone $C = \text{range}(x)$, so it belongs to $\mathcal{F}$. A pair $(f, g) \in \mathcal{F}$ can be realized by building a point whose range induces (f, g), which can be done by imposing computability conditions on f and g and applying the results from Section 4.

6 Left-c.e. quadratic polynomials

In this section, we briefly investigate the quadratic real polynomials $P_{a,b,c}(X) = aX^2 + bX + c$ that are left-c.e. functions of X. Our main problem is the following:

▶ **Problem 3.** *For which triples (a, b, c) is the polynomial $P_{a,b,c}$ left-c.e.?*

The key observation is that $P_{a,b,c}(X)$ is linear in (a, b, c), which allows to think of a left-c.e. polynomial as a semicomputable point $(a, b, c) \in \mathbb{R}^3$. More precisely, the ordering $(a, b, c) \preceq (a', b', c')$ defined by $P_{a,b,c} \leq P_{a',b',c'}$ is a vector space ordering. Hence its positive cone is a convex cone $C = \{(a, b, c) \in \mathbb{R}^3 : P_{a,b,c} \geq 0\} = \{(a, b, c) \in \mathbb{R}^3 : a, c \geq 0 \text{ and } b^2 \leq 4ac\}$. Its dual is $C^* = \{(a, b, c) \in \mathbb{R}^3 : a, c \geq 0 \text{ and } b^2 \leq ac\}$ and is the closure of the conical hull of the vectors $(X^2, X, 1)$, with $X \in \mathbb{R}$.

Thus $P_{a,b,c}$ is left-c.e. if and only if (a, b, c) is C^*-c.e. This reformulation allows us to think geometrically about left-c.e. polynomials, and to apply the results of this paper to these objects. Let us list a few properties of left-c.e. polynomials, some of them being derived from the analysis developed in the paper:

1. There is a symmetry between a and c and between b and $-b$. More precisely, $P_{a,b,c}$ is left-c.e. $\iff$ $P_{a,b,c}, P_{c,b,a}, P_{a,-b,c}$ and $P_{c,-b,a}$ are left-c.e. for $X \geq 1$.
2. If $P_{a,b,c}$ is left-c.e. then a, c are left-c.e. and b is d-c.e. (b is a difference of left-c.e. numbers).
3. If a is Solovay complete left-c.e. then (Proposition 3.16)
$$S(b, a)^2 < 4S(c, a) \implies P_{a,b,c} \text{ is left-c.e.} \implies S(b, a)^2 \leq 4S(c, a).$$
4. Let $P_{a,b,c}$ be left-c.e. For computable $X > 0$,
$$-\frac{1}{\sqrt{X}} \leq \underline{S}(b, aX + c) \quad \text{and} \quad \overline{S}(b, aX + c) \leq \frac{1}{\sqrt{X}}.$$
Indeed, $aX^2 + bX + c$ is left-c.e. for all computable $X \in \mathbb{R} \iff \frac{1}{\sqrt{Y}}(aY + c) \pm b$ is left-c.e. for all computable $Y > 0$ (take $Y = X^2$).
5. Let $x = (a, b, c)$ be C^*-c.e. and generic inside $x + C$ (it exists as C^* is computable, see Theorem 4.3). $P_{a,b,c}$ is left-c.e. and for computable $X > 0$,
$$-\frac{1}{\sqrt{X}} = \underline{S}(b, aX + c) \quad \text{and} \quad \overline{S}(b, aX + c) = \frac{1}{\sqrt{X}}.$$
The second equality is obtained as follows: for a rational $q < \frac{1}{\sqrt{X}}$, $(qX, -1, q) \notin C^* = \text{range}(x)$ so $q(aX + c) - b$ is not left-c.e., hence $\overline{S}(b, aX + c) = \frac{1}{\sqrt{X}}$.

Although b is Solovay reducible to $aX + c$ for each computable $X > 0$, b is not reducible to neither a nor c and $\underline{S}(b, a) = \underline{S}(b, c) = -\infty$ and $\overline{S}(b, a) = \overline{S}(b, c) = +\infty$. Indeed, for $q \in \mathbb{Q}$, both $(q, \pm 1, 0)$ and $(0, \pm 1, q)$ are outside C^*.

6. The condition that $P_{a,b,c}$ is left-c.e. cannot be reduced to a finite number of linear combination of a, b, c being left-c.e. Indeed, such a condition would express that the point (a, b, c) is D-c.e. for some polygonal convex cone D, but the convex cone C^* is not polygonal (it is determined by infinitely many directions).
7. The condition that $P_{a,b,c}$ is left-c.e. cannot be characterized by simply considering the values of $\underline{S}(b, c)$, $\overline{S}(b, c)$, $\underline{S}(a, c)$, $\overline{S}(a, c)$, $\underline{S}(b, a)$, $\overline{S}(b, a)$. Indeed, these values only reflect the intersections of range(a, b, c) with the three planes $z = 0$, $x = 0$ and $y = 0$, which do not determine completely range(a, b, c).

We do not know if it is possible to better understand Problem 3, i.e. whether it is possible to reduce this property to more fundamental properties of a, b, c. The results presented above suggest a negative answer to that question.

We mention that a similar analysis can be made of co-c.e. disks in the plane. The disk centered at (a, b) with radius c is co-c.e. if and only if the point $(a, b, c) \in \mathbb{R}^3$ is C^*-c.e., where $C = C^* = \{(x, y, z) : z \leq 0, x^2 + y^2 \leq z^2\}$. Again C^* is not polygonal so one cannot reduce the condition that the disk is co-c.e. to a finite number of conditions.

References

1 George Barmpalias and Andrew Lewis-Pye. Differences of halting probabilities. *J. Comput. Syst. Sci.*, 89:349–360, 2017.

2 Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

3 Vasco Brattka and Gero Presser. Computability on subsets of metric spaces. *Theoretical Computer Science*, 305(1-3):43–76, 2003.

4 Mathieu Hoyrup. Genericity of Weakly Computable Objects. *Theory Comput. Syst.*, 60(3):396–420, 2017.

5 Mathieu Hoyrup, Diego Nava Saucedo, and Don M. Stull. Semicomputable Geometry. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 129:1–129:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

6 Martin Kummer and Marcus Schäfer. Computability of convex sets. In Ernst W. Mayr and Claude Puech, editors, *STACS 95*, pages 550–561, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

7 Joseph S. Miller. On Work of Barmpalias and Lewis-Pye: A Derivation on the D.C.E. Reals. In Adam R. Day, Michael R. Fellows, Noam Greenberg, Bakhadyr Khoussainov, Alexander G. Melnikov, and Frances A. Rosamond, editors, *Computability and Complexity - Essays Dedicated to Rodney G. Downey on the Occasion of His 60th Birthday*, volume 10010 of *Lecture Notes in Computer Science*, pages 644–659. Springer, 2017.

8 André Nies. *Computability and randomness.* Oxford logic guides. Oxford University Press, 2009.

9 Martin Ziegler. Computability on Regular Subsets of Euclidean Space. *Mathematical Logic Quarterly*, 48(S1):157–181, 2002.

Bounded-Depth Frege Complexity of Tseitin Formulas for All Graphs

Nicola Galesi
Dipartimento di Informatica, *Sapienza Università di Roma,* Via Salaria 113, Rome, Italy
nicola.galesi@uniroma1.it

Dmitry Itsykson
St. Petersburg Department of V.A. Steklov Institute of Mathematics of the
Russian Academy of Sciences, Fontanka 27, St. Petersburg, Russia
dmitrits@pdmi.ras.ru

Artur Riazanov
St. Petersburg Department of V.A. Steklov Institute of Mathematics of the
Russian Academy of Sciences, Fontanka 27, St. Petersburg, Russia
aariazanov@gmail.com

Anastasia Sofronova
St. Petersburg Department of V.A. Steklov Institute of Mathematics of the
Russian Academy of Sciences, Fontanka 27, St. Petersburg, Russia
St. Petersburg State University, 7-9 Universitetskaya Emb., St. Petersburg, Russia
ana.a.sofronova@gmail.com

Abstract

We prove that there is a constant K such that *Tseitin* formulas for an undirected graph G requires proofs of size $2^{\mathrm{tw}(G)^{\Omega(1/d)}}$ in depth-d Frege systems for $d < \frac{K \log n}{\log \log n}$, where $\mathrm{tw}(G)$ is the treewidth of G. This extends Håstad recent lower bound for the grid graph to any graph. Furthermore, we prove tightness of our bound up to a multiplicative constant in the top exponent. Namely, we show that if a Tseitin formula for a graph G has size s, then for all large enough d, it has a depth-d Frege proof of size $2^{\mathrm{tw}(G)^{\mathcal{O}(1/d)}}\mathrm{poly}(s)$. Through this result we settle the question posed by M. Alekhnovich and A. Razborov of showing that the class of Tseitin formulas is quasi-automatizable for resolution.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Tseitin formula, treewidth, AC_0-Frege

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.49

Related Version A full version is available as ECCC report TR19-069 `https://eccc.weizmann.ac.il/report/2019/069/`.

Funding The research is supported by Russian Science Foundation (project 16-11-10123).

Acknowledgements The authors thank Navid Talebanfard for discussions on the lower bound. Nicola also thanks Paul Wollan for introducing him to the tree-cutwidth. Dmitry is a Young Russian Mathematics award winner and would like to thank sponsors and jury of the contest.

1 Introduction

Propositional proof complexity is motivated by the result of Cook and Reckhow [12] saying that if there is a propositional proof system in which any unsatisfiable formula F has a short proof of unsatisfiability (of size polynomial in the size of F), then NP = coNP. In the last 30 years the complexity of proofs was investigated for several proof systems with the aim of finding concrete evidence, and eventually a proof, that for all proof systems there is a propositional formula which is not efficiently provable, i.e. requires super-polynomial proof

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 49; pp. 49:1–49:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

size. The approach followed to prove such lower bounds was essentially borrowed from circuit complexity. Lines in a proof are Boolean formulas and we can define different proof system according to the circuit complexity of such formulas. For example resolution, a well-known refutational system for CNFs, corresponds to a system where formulas are of depth 1. In circuit complexity we keep on trying to strength lower bounds to computationally more powerful class of circuits. In proof complexity we follow the analogous approach: to strength lower bounds to systems working on formulas computationally more powerful. The hope is that techniques used to prove lower bounds for classes of Boolean circuits could be lifted to work with proof systems operating with formulas in the same circuit class. At present however we are far from such ideal situation and in fact, in terms of circuit classes, lower bounds for proof systems are well below those for Boolean circuits.

The complexity of proofs in resolution is largely studied. The first lower bound for (a restriction of) resolution was given by Tseitin in [35]. To obtain his result Tseitin introduced a class of formulas (nowadays known as *Tseitin formulas*) encoding a generalisation of the principle that the sum of the degrees of all vertices in a graph is an even number. A Tseitin formula $\mathsf{T}(G, f)$ is defined for every undirected graph $G(V, E)$ and a charging function $f : V \to \{0, 1\}$. We introduce a propositional variable for every edge of G so that $\mathsf{T}(G, f)$ is a CNF representation of a linear system over the field GF(2) that for every vertex $v \in V$ states that the sum of all edges incident to v equals $f(v)$. Tseitin formulas, usually defined on graphs with good expansion properties, are among the main examples we could prove lower bounds for in different proof systems. For unrestricted resolution it was Urquhart in [36] and later Ben-Sasson and Wigderson [8] who proved exponential lower bounds for Tseitin formulas over constant-degree expander graphs. Another example of an important principle largely studied in proof complexity is the Pigeonhole principle, PHP_n. Haken [19], Beame and Pitassi [4] and Ben-Sasson Wigderson [8] proved exponential resolution lower bounds for CNF encoding of the negation of PHP_n, which were later generalized and improved in several other works [13, 30, 32, 33, 9, 24].

Bounded-depth Frege extends resolution since the formulas in the line of proofs are computable by AC^0 circuits, i.e. constant-depth circuits with unbounded fan-in gates. The importance of understanding the complexity of proofs in bounded-depth-Frege systems was due at least to two reasons: (1) for general Frege systems, where formulas have no restrictions, i.e. are of depth $\mathcal{O}(\log n)$, Buss in [10] proved that the Pigeonhole principle can be proved in polynomial size, hence obtaining an exponential separation with resolution. (2) Lower bounds for AC^0-circuits were known [21, 15] and hence we could hope for applying lower bound techniques for AC^0 to lower bounds to bounded-depth Frege. Studying the complexity of proofs in bounded-depth Frege is of the utmost importance since it is a frontier proof system, i.e. one of the strongest propositional proof systems with known significant lower bounds at the moment. Any advance is then a step towards proving lower bounds for $\mathrm{AC}^0[2]$-Frege, i.e. a bounded-depth Frege admitting also formulas with parity gates, which are unknown at the moment, though we know since a long time exponential lower bounds for $\mathrm{AC}^0[2]$ circuits [34, 31, 25]. In this work we contribute to the complexity of proofs in bounded-depth Frege proving new lower bounds for Tseitin formulas.

Ajtai in [1] was the first to prove a lower bound in bounded-depth Frege. He showed that a proof of PHP_n must have a super-polynomial size. His result was later followed by several results simplifying his technique [5] and improving the lower bound [27, 26] showing that any polynomial-size Frege proof of PHP_n must have depth $\Omega(\log \log n)$. The proof complexity of Tseitin formulas in bounded-depth Frege was first considered by Urquhart and Fu in [37], a work where they simplified and adapted the lower bound for the PHP_n to the case of Tseitin formulas over a complete graph. Ben-Sasson in [7], proved exponential lower bounds for the Tseitin formulas over constant-degree expander graphs using a new reduction from the pigeonhole principle [37]. All these lower bounds are adaptation of the technique of [27, 26], hence vanish when the depth of formulas in the proof is more than $\log\log n$. In a very recent major breakthrough [28] showed that Tseitin formulas over a 3-expander graph of n nodes requires super-polynomial bounded-depth Frege proofs at depth $\mathcal{O}(\sqrt{\log n})$. Their result was later improved to depth up to $\frac{C\log n}{\log\log n}$ by Håstad in [22] but for Tseitin formulas defined only on the 2-dimensional grid, where C is a positive constant.

Proofs of $\mathsf{T}(G, f)$ were studied in terms of the treewidth of G, $\mathrm{tw}(G)$, for resolution [2, 17] and for OBDD proof systems [18]. We use Håstad result to prove tight bounds on the complexity of proofs in bounded-depth Frege of $\mathsf{T}(G, f)$ *over any graph* G in terms of the treewidth of G. Our main result is the following theorem:

▶ **Theorem 1.** *There is a constant K such that for any graph G over n nodes and for all $d \leq K\frac{\log n}{\log\log n}$, every depth-$d$ Frege proof of $\neg\mathsf{T}(G, f)$ has size at least $2^{\mathrm{tw}(G)^{\Omega(1/d)}}$. Furthermore, for all large enough d there exist depth-d Frege proofs of $\neg\mathsf{T}(G, f)$ of size $2^{\mathrm{tw}(G)^{\mathcal{O}(1/d)}}\mathrm{poly}(|\mathsf{T}(G, f)|)$.*

A class of unsatisfiable CNFs F_n is (quasi-)automatizable in a proof systems S, if there exists a deterministic algorithm that, given F in F_n returns a proof in S in time which is (quasi-)polynomial in $|F| + |\tau_F|$, where $|\tau_F|$ is the size of shortest proof of F in S. Theorem 1, together with the results from [17, 20, 2, 3] implies that *for any graph* G, the class of Tseitin formulas is *quasi-automatizable* in all systems between treelike resolution and constant-depth Frege. This answers the problem of [2] of extending to all graphs the quasi-automazibablity of $\mathsf{T}(G, f)$ in resolution, known only for graphs with bounded cyclicity [2].

Using a result in [3, 23] we can also prove that the size of proofs of $\mathsf{T}(G, f)$ in proof systems between tree-like resolution and bounded-depth Frege are quasi-polynomially correlated, i.e. *if* $\mathsf{T}(G, f)$ *has a proof of size* S *in bounded-depth Frege, then it has a proof of size at most* $2^{\mathrm{poly}(\log(S+|\mathsf{T}(G,f)|))}$ *in treelike resolution and vice versa.* This result provides evidence to the conjecture of Urquhart that the shortest resolution proofs of $\mathsf{T}(G, f)$ are regular. Finally other consequences of Theorem 1 are: (1) It gives polynomial size Frege proofs of $\mathsf{T}(G, f)$ of depth $\log(\mathrm{tw}(G))$. (2) It improves the lower bounds of [7, 28] since expanders have treewidth $\Omega(n)$ and on such graphs our lower bound is $2^{n^{\Omega(1/d)}}$, which works for larger d than [7, 28]; (3) Induces a strict depth-hierarchy for the proof complexities of Tseitin formulas over an infinite sequence of graphs G_n.

Overview of the proof technique

In Theorem 17 we prove the lower bound from Theorem 1. The proof is based on the improvement of the Excluded Grid Theorem by Robertson and Seymour recently obtained by Chuzhoy [11]: an arbitrary graph G contains as a minor a $r \times r$ grid, where $r = \Omega\left(\mathrm{tw}(G)^{1/37}\right)$. More precisely we use the corollary of this result (see Corollary 8) stating that any graph G has a *wall* of size r as a *topological minor* (i.e. can be obtained from G by several removing of vertices, edges and *suppressions*, see Fig. 1 and Fig. 3). Our proof consists of two parts: at first, we show that if H is a topological minor of G, then any bounded-depth Frege proof of a Tseitin formula $\mathsf{T}(G, f)$ can be transformed to a proof of a $\mathsf{T}(H, f')$, with constant increase in depth and polynomial increase in size. And then we prove a lower bound on the size of depth-d Frege proof of Tseitin formulas based on walls. In this proof we use the lower bound for grid graphs proved by Håstad [22].

In Theorem 18 we prove the upper bound from Theorem 1. We consider the *compact representation* of linear functions $\mathbb{F}_2^n \to \mathbb{F}_2$ on variables $x_1, x_2, \ldots, x_n$ by propositional formulas of depth d and of size $2^{n^{\mathcal{O}(1/d)}}$. We show that for linear functions f and g if the equations $f(x) = a$ and $g(x) = b$ are given in our representation, then there is a derivation of $(h+g)(x) = a + b$ of depth d and of size $2^{n^{\mathcal{O}(1/d)}}$. We also show that if a linear equation is represented in CNF, then it is possible to infer its compact representation with depth d and size $2^{n^{\mathcal{O}(1/d)}}$. Since a Tseitin formula is an unsatisfiable system of linear equations written in CNF, hence it is possible to prove a Tseitin formula in size $2^{m^{\mathcal{O}(1/d)}}$ and depth d, where m is the number of edges in G. However we wish to have the treewidth of G instead of m. We consider a tree-partition of a graph G, the vertices of G are split into bags and there exists a tree such that bags are nodes of this tree and if two vertices of G are connected, then they are either in one bag or in adjacent bags. It is known that there is a tree partition where the size of bags are at most $\mathcal{O}(\mathrm{tw}(G)\Delta(G))$ [38]. Since the number of edges touching a given bag is $\mathcal{O}(\mathrm{tw}(G)\Delta(G)^2)$ we can use the compact representation to take care of the equations involving the parity of sum of adjacent bags with proofs growing in terms of the treewidth of G.

Organization

The paper is divided into four sections. After the Preliminary section, we have Section 3 for the lower bound (Theorem 17), Section 4 for the upper bound (Theorem 18). Proofs omitted due to space constraints may be found in [16].

2 Preliminaries

2.1 Formulas and restrictions

We consider propositional formulas over binary $\vee$ and $\wedge$, unary $\neg$ and Boolean constants $\mathbf{0}, \mathbf{1}$. We represent formulas as rooted trees such that internal vertices are labeled with connectives and leaves are labeled with propositional variables or Boolean constants. The *depth* of a formula is the maximal number of alternations of types of connectives over all the paths from the root to a leaf plus one.

We assume that disjunctions with unbounded fanin are represented via binary disjunctions. By default, we mean that $\bigvee_{i=1}^{n} x_i$ is right-associative; i.e., denotes $(\ldots(x_1 \vee x_2) \vee \ldots) \vee x_{n-1}) \vee x_n$; we also assume the same for $\bigwedge$.

We denote by $\mathsf{vars}(F)$ the set of variables of a formula F. A *partial assignment* α for a formula F is mapping from $\mathsf{vars}(F) \to \{0, 1, *\}$, where $\alpha(x) = *$ if x is unassigned. We denote by $\mathsf{dom}(\alpha) = \alpha^{-1}(\{0, 1\})$ the set of variables in F which α assigns a Boolean value.

2.2 Pudlák-Buss games

We use the game interpretation of Frege proofs introduced by Pudlák and Buss [29]. Let us define a game with two players Pavel and Sam. The game starts with initial conditions of the form $\varphi_1 = a_1, \ldots, \varphi_k = a_k$, where $\varphi_1, \varphi_2, \ldots, \varphi_k$ are propositional formulas and $a_1, a_2, \ldots, a_k \in \{0, 1\}$ such that $\bigwedge_{i=1}^{k}(\varphi_i = a_i)$ is identically false. Sam claims that he knows an assignment of variables that satisfies $\bigwedge_{i=1}^{k}(\varphi_i = a_i)$, the goal of Pavel is to convict Sam. At each his move Pavel asks Sam the value of a propositional formula and Sam gives an answer. The game stops when Pavel convicts Sam, namely Pavel finds an immediate contradiction among initial conditions and Sam's answers. An immediate contradiction with a Boolean connective $\circ$ of arity t is a set of $(t+1)$ formulas $\alpha_1, \ldots, \alpha_t$ and $\circ(\alpha_1, \ldots, \alpha_t)$ with claimed values $a_1, \ldots, a_t$ and b such that $\circ(a_1, \ldots, a_k) \neq b$. In particular, $\mathbf{0}$ with claimed value 1 is an immediate contradiction.

A strategy of Pavel is a function that maps initial conditions and the history of a game to a propositional formula (request). A winning strategy is a strategy that allows Pavel to convict Sam for any behaviour of Sam. A winning strategy of Pavel can be represented as a binary tree whose nodes are labeled with Pavel's requests and edges correspond to Sam's answers. A leaf of the tree corresponds to an immediate contradiction among initial conditions and equalities corresponding to the path from the root to this leaf.

A Pudlák-Buss game derivation of a formula ψ from formulas $\varphi_1, \varphi_2, \ldots, \varphi_s$ is a tree of a Pavel's winning strategy in a game with initial conditions $\varphi_1 = 1, \varphi_2 = 1, \ldots, \varphi_s = 1, \psi = 0$. In that follows by derivations we always mean Pudlák-Buss game derivations. We are interested in the two complexity parameters of derivations: 1) the size of a derivation S that equals the total size of formula ψ and all formulas that are used as labels of nodes; 2) the depth of a derivation d is the maximum depth of ψ and formulas that are used as labels of nodes. We use the notation $\varphi_1, \ldots, \varphi_s \vdash_d \psi$ for a derivation of ψ from $\varphi_1, \varphi_2, \ldots, \varphi_s$ of depth at most d. A derivation of φ is a derivation of φ from the empty set of formulas.

▶ **Lemma 2.** *Assume that there is a derivation $\varphi_1, \ldots, \varphi_k \vdash_{d_1} \psi_1$ of size S_1 and also there is a derivation $\varphi_1, \ldots, \varphi_k, \psi_1 \vdash_{d_2} \psi_2$ of size S_2, then there is a derivation $\varphi_1, \ldots, \varphi_k \vdash_{\max\{d_1, d_2\}} \psi_2$ of size $S_1 + S_2$.*

Proof. Let us create the new tree with the root labelled with ψ_1 such that edge form the root labelled with 0 goes to the root of the first derivation and edge labelled with 1 goes to the root of the second derivation. ◀

▶ **Lemma 3.** *1. If a formula φ has a Frege derivation of size S and depth d, then φ has a Pudlák-Buss game derivation of size $\mathcal{O}(S^2)$ and depth d. 2. If φ has a Pudlák-Buss game derivation of size S and depth d, then φ has a Frege derivation of size $\mathcal{O}(S^3)$ and depth $d + \mathcal{O}(1)$.*

▶ **Lemma 4.** *Let ψ_1 and ψ_2 be two formulas of depth at most d such that $|\mathsf{vars}(\psi_1) \cup \mathsf{vars}(\psi_2)| = k$ and ψ_1 semantically implies ψ_2. Then there exists a derivation $\psi_1 \vdash_d \psi_2$ of size at most $2^k \left(|\psi_1|^2 + |\psi_2|^2\right)$.*

A shortcut contradiction for the disjunction is a situation where Pavel asks Sam formulas $\bigvee_{i=1}^k \alpha_i$ and α_j for $j \in [k]$ and gets the answers 0 and 1 respectively. Similarly a shortcut contradiction for the conjunction is a situation where Pavel asks Sam formulas $\bigwedge_{i=1}^k \alpha_i$ and α_j for $j \in [k]$ and gets the answers 1 and 0. *An ordinary derivation* is a derivation which does not use shortcut contradictions.

▶ **Lemma 5.** *Consider a derivation of size S and of depth d that uses shortcut contradictions in leaves. Then there is an ordinary derivation of size at most S^3 and of depth d.*

Tseitin Formulas. Let $G(V, E)$ be an undirected graph and $v \in V$. We denote by $E(v)$ the set of edges in E incident with v and by $N(v)$ the set of neighbours $u \in V$ of v, i.e. the u such that $(u, v) \in E(v)$.

A vertex-charging for $G(V, E)$ is a mapping $f : V \longrightarrow \{0, 1\}$. We say that f is an odd-charging of G if $\sum_{v \in V} f(v) \equiv 1 \mod 2$. The *Tseitin formulas* defined on G using variables $x_e, e \in E$ are the formulas: $\mathsf{T}(G, f) := \bigwedge_{v \in V} \mathsf{Par}(v)$, where $\mathsf{Par}(v)$ is a CNF formula representing $\bigoplus_{e \in E(v)} x_e = f(v)$.

▶ **Lemma 6** ([36]). *$\mathsf{T}(G,f)$ is unsatisfiable if and only if there is a connected component U of G such that the restriction of f on U is odd-charging.*

In this work we will work with the tautological form of Tseitin formulas in the form of $\neg\,\mathsf{T}(G,f)$.

2.3 Grids, Walls, Minors, Topological Minors and Treewidth

We consider 4 structural operations on undirected graphs $G = (V,E)$ possibly with parallel edges, but without loops. We follow [6, 14].

- *edge removal* of $e \in E$. It produces the graph $[G\backslash e] = (V, E\backslash\{e\})$.
- *vertex removal* of $v \in V$. It produces the graph $[G\backslash v] = (V\backslash\{v\}, E\backslash E(v))$, where $E(v)$ is the set of edges in E incident with $v \in V$.
- *edge contraction* of $e = (uv) \in E$. Is the replacement of u and v with a single vertex such that edges incident to the new vertex are the edges other than e that were incident with u or v. The resulting graph $G\star e$ has one edge less than G.
- *vertex suppression* of a vertex v in G of degree 2. Let u and w be v's neighbours in G. The suppression of v is obtained by deleting v along with two edges (uv) and (wv) and adding a new edge (wu) (possibly parallel to an existing one). The resulting graph $[G\backslash_s v]$ has one vertex less than G. See Figure 1.

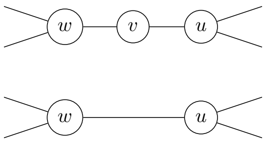


Figure 1 Suppression of v from G.

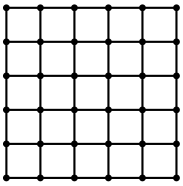


Figure 2 The grid $\mathcal{H}_{5,5}$.

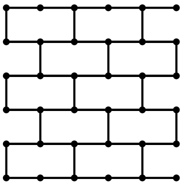


Figure 3 The wall $\mathcal{W}_5$.

A graph H is a *minor* of G if H can be obtained from G by a sequences of edge and vertex removals and edge contractions. A graph H is a *topological minor* of G if H can be obtained from G by a sequence of edge removals, vertex removals and by vertex suppressions [6, 14].

The grid $\mathcal{H}_{m,n}$ is the graph of the cellular rectangle $m \times n$; it has $(m+1)(n+1)$ vertices and $n(m+1) + m(n+1)$ edges, among them $n(m+1)$ horizontal and $m(n+1)$ vertical edges. See fig. 2.

The wall $\mathcal{W}_n$ is a subgraph of $\mathcal{H}_{n,n}$ that is obtained by the removing of several vertical edges. Vertical edges of $\mathcal{H}_{n,n}$ are in n rows and we enumerate them in every row from the left to the right. In the odd rows we remove all vertical edges with even numbers and in even rows we remove all vertical edges with odd numbers. See fig. 3.

A tree decomposition of an undirected graph $G(V,E)$ is a tree $T = (V_T, E_T)$ such that every vertex $u \in V_T$ corresponds to a set $X_u \subseteq V$ and it satisfies the following properties: 1. The union of X_u for $u \in V_T$ equals V. 2. For every edge $(a,b) \in E$ there exists $u \in V_T$ such that $a, b \in X_u$. 3. If a vertex $a \in V$ is in the sets X_u and X_v for some $u, v \in V_T$, then it is also in X_w for all w on the path between u and v in T.

The width of a tree decomposition is the maximum $|X_u|$ for $u \in V_T$ minus one. *A treewidth* of a graph G is the minimal value of the width among all tree decompositions of the graph G.

Recall the following Theorem proved in [11].

▶ **Theorem 7** ([11]). *If G has a treewidth t, then it has the grid $\mathcal{H}_{r,r}$ as a minor, where $r = \Omega(t^{1/37})$.*

The following Corollary was mentioned in [6].

▶ **Corollary 8** ([6]). *If G a has treewidth t, then it has the wall $\mathcal{W}_r$ as a topological minor, where $r = \Omega(t^{1/37})$.*

3 The Lower Bound

3.1 Topological Minors and Tseitin Formulas

Le φ be a formula and let α be a partial assignment to variables of φ. Define $\varphi[\alpha]$ to be the formula obtained from φ substituting each variable x in the domain of α, with the constant assigned to x by α. Notice that φ and $\varphi[\alpha]$ have the same size and depth.

▶ **Lemma 9.** *Let Φ_a and Φ'_a for $a \in A$ be propositional formulas of depth at most d such that $|\mathsf{vars}(\Phi_a) \cup \mathsf{vars}(\Phi'_a)| \leq k$. Assume that for all $a \in A$, Φ_a is semantically equivalent to Φ'_a. Then $\neg \bigwedge_{a\in A} \Phi'_a \vdash_{d+\mathcal{O}(1)} \neg \bigwedge_{a\in A'} \Phi_a$ of size at most $2^k \mathrm{poly}\left(\sum_{a\in A}(|\Phi_a| + |\Phi'_a|)\right)$, where $A' = \{a \in A \mid \Phi_a$ is not identically true$\}$.*

▶ **Lemma 10** ([18]). *Let $G(V, E)$ be a connected graph and $H(V', E')$ be a connected subgraph of G with $E' \neq \emptyset$ that is obtained from G by the deletion of some vertices and edges. For every unsatisfiable Tseitin formula $\mathsf{T}(G, f)$ there exists a partial assignment α to variables x_e for $e \in E \setminus E'$ such that α does not falsify any clause of $\mathsf{T}(G, f)$.*

▶ **Lemma 11.** *Let $G(V, E)$ be a connected graph and $H(V', E')$ be a connected subgraph of G. Assume that there is a derivation $\vdash_d \neg\mathsf{T}(G, f)$ of size S. Then for some f' there is a derivation $\vdash_{d+\mathcal{O}(1)} \neg\mathsf{T}(H, f')$ of size $S + \mathrm{poly}(|\mathsf{T}(G, f)|)$.*

Proof. Let T be the game tree of $\vdash_d \neg\mathsf{T}(G, f)$. Let α be given by Lemma 10 that is defined on all variables x_e for $e \in E \setminus E'$ and does not falsify any clause of $\mathsf{T}(G, f)$. $T[\alpha]$ be the tree obtained form T applying the substitution α to all the queried formulas. Size and depth do not change, hence $T[\alpha]$ defines a derivation $\vdash_d\neg\mathsf{T}(G, f)[\alpha]$ of size S. $\neg\mathsf{T}(G, f)$ has the form $\neg \bigwedge_{v\in V} \mathsf{Par}(v)$, where $\mathsf{Par}(v)$ is a parity condition of the vertex v. Hence, $\neg\mathsf{T}(G, f)[\alpha]$ is of the form $\neg \bigwedge_i \mathsf{Par}(v)[\alpha]$. If $v \notin V'$, then α assigns values to all variables from $\mathsf{Par}(v)$, since α does not falsify $\mathsf{Par}(v)$, α satisfies $\mathsf{Par}(v)$, hence $\mathsf{Par}(v)[\alpha]$ is identically true. If $v \in V'$, then $\mathsf{Par}(v)[\alpha]$ is a parity statement depending on variables x_e, where $e \in E'$ is incident to v. Hence, for $v \in V'$, $\mathsf{Par}(v)[\alpha]$ is semantically equivalent to a parity condition of a Tseitin formula $\mathsf{T}(H, \varphi')$ for some charging φ'. Let Δ be the maximal degree of G. Then every parity condition of $\mathsf{T}(H, \varphi')$ or $\mathsf{T}(G, \varphi)$ depends on at most Δ variables. Notice that since we represent parities in CNF, $|\mathsf{T}(G, f)| \geq 2^\Delta$. By Lemma 9, there is a derivation $\neg\mathsf{T}(G, f)[\alpha] \vdash_{\mathcal{O}(1)} \neg\mathsf{T}(H, f')$ of size $\mathrm{poly}(|\mathsf{T}(G, f)|)$. The claim follows using the size S, depth d derivation of $\neg\mathsf{T}(G, f)[\alpha]$ together with Lemma 2. ◀

A 1-*substitution* for a formula φ is a partial function mapping variables of φ into its literals. After applying a 1-substitution σ to φ, the depth of the new formula $\varphi[\sigma]$ can increase by one. However 1-substitutions are closed under composition: if σ_1 maps $[y \mapsto \neg z]$ and σ_2 maps $[x \mapsto \neg y]$, then $\sigma = \sigma_1 \circ \sigma_2$ is the 1-substitution $[x \mapsto z, y \mapsto \neg z]$. We use 1-substitutions to handle in $\mathsf{T}(G, f)$ the operation of *vertex suppression* on the graph G. Let $G = (V, E)$ be a graph and $v \in V$ be a node and let $\mathsf{T}(G, f)$ be a Tseitin formula on G. Let v

be a degree-2 vertex v in G with neighbours u and w. Consider the following 1-substitution σ_v and the charge function f_v for $[G\backslash_s v]$:

$$\sigma_v = \begin{cases} [x_{vw} \mapsto x_{wu}, x_{vu} \mapsto x_{wu}] & \text{if } f(v) = 0 \\ [x_{vw} \mapsto x_{wu}, x_{vu} \mapsto \neg x_{wu}] & \text{if } f(v) = 1 \end{cases} \qquad f_v(z) = \begin{cases} f(z) & \text{if } z \in V \setminus \{u, v\} \\ f(u) + f(v) & \text{if } z = u \end{cases}$$

Let $G(V,E)$ be a graph and $f : V \to \{0,1\}$ be a charging. Let A be a finite set. We say that a formula Ψ is a *pseudo Tseitin formula based on G and f with fake vertices in A*, and we write Ψ is $\mathsf{T}^*_A(G,f)$, if Ψ has the form $\bigwedge_{v\in V\cup A} \psi_v$, where

1. for all $v \in V$, ψ_v is a propositional formula depending on variables x_e for all edges e incident to v. And ψ_v is semantically equivalent to the parity condition $\mathsf{Par}(v)$ of $\mathsf{T}(G,f)$.
2. for all $v \in A$, ψ_v is a tautology.

▶ **Lemma 12.** *Let $G(V,E)$ be a connected constant-degree graph over n vertices. Let $[G\backslash_s v]$ be the graph obtained after the suppression of a degree-2 vertex v in G. If Ψ is $\mathsf{T}^*_A(G,f)$, then $\Psi[\sigma_v]$ is $\mathsf{T}^*_{A\cup\{v\}}([G\backslash_s v], f_v)$.*

Proof. Assume that v is linked to two vertices w and u in G. Let A be the set of fake vertices of Ψ so Ψ has the form $\bigwedge_{x\in V\cup A} \psi_x$, hence $\Psi[\sigma_v]$ is $\bigwedge_{x\in V\cup A} \psi_x[\sigma_v]$. For $x \in A$, $\psi(x)$ is a tautology, hence $\psi_x[\sigma_v]$ is also a tautology. By the definition of σ_v, $\psi_v[\sigma_v]$ is a tautology. It is not hard to verify that for $x \in V \setminus \{v\}$, $\psi_x[\sigma_v]$ is equivalent to parity condition of $\mathsf{T}([G\backslash_s v], f_v)$. Hence, $\Psi[\sigma_v]$ is $\mathsf{T}^*_{A\cup\{v\}}([G\backslash_s v], f_v)$ ◀

▶ **Lemma 13.** *Let $G(V,E)$ be a graph and $f : V \to \{0,1\}$ and $W = \{v_1, \ldots, v_k\}$ be degree 2 nodes in V suppressed in that order from G and $[G\backslash_s W]$ be the resulting graph. Let σ_i be the corresponding 1-substitutions and let $\sigma = \sigma_k \circ \ldots \circ \sigma_1$. There is a charging f_k of G such that if Ψ is $\mathsf{T}^*_A(G,f)$, then $\Psi[\sigma]$ is $\mathsf{T}^*_{A\cup W}([G\backslash_s W], f_k)$.*

▶ **Lemma 14.** *Let G be a connected graph on n vertices and with the maximal degree at most 3. Let H be obtained from G by several suppressions. Assume that there is a derivation of $\neg\,\mathsf{T}(G,f)$ of size S and depth d. Then for some charging f_k there is a derivation of $\neg\,\mathsf{T}(H,f_k)$ of size $\mathcal{O}(S) + \mathrm{poly}(n)$ and depth $d + \mathcal{O}(1)$.*

Proof. Assume that, in order, to get H from G we have to apply suppressions for vertices $W = \{v_1, \ldots, v_k\}$. Let σ_i be the 1-substitutions corresponding to the suppression of v_i, and let $\sigma = \sigma_k \circ \cdots \circ \sigma_1$. $\mathsf{T}(G,f)$ is $\mathsf{T}^*_\emptyset(G,f)$. Let f_k be the charging given by Lemma 13 applied to $\mathsf{T}(G,f)$ and $[G\backslash_s W] = H$. Then $\mathsf{T}(G,f)[\sigma]$ is $\mathsf{T}^*_W(H,f_k)$. We apply the 1-substitution σ to the given derivation of $\neg\,\mathsf{T}(G,f)$ and we get a derivation of $\neg\,\mathsf{T}(G,f)[\sigma]$ of size $\mathcal{O}(S)$ and depth at most $d+1$. By Lemma 9, applied on $\mathsf{T}(G,f)[\sigma]$ and $\mathsf{T}(H,f_k)$, there is a derivation $\neg\,\mathsf{T}(G,f)[\sigma] \vdash_{d+\mathcal{O}(1)} \neg\,\mathsf{T}(H,f_k)$ of size $\mathrm{poly}(n)$. Combining the two derivations together by Lemma 2 we obtain a derivation $\vdash_{d+\mathcal{O}(1)} \mathsf{T}(H,f_k)$ of size $\mathcal{O}(s) + \mathrm{poly}(n)$. ◀

3.2 From Walls To Grids

▶ **Lemma 15.** *If there exists a derivation $\vdash_d \neg\,\mathsf{T}(\mathcal{W}_n, f)$ of size S, then there exists a derivation $\vdash_{d+\mathcal{O}(1)} \neg\,\mathsf{T}(M_n, f')$ of size $\mathcal{O}(S) + \mathrm{poly}(n)$, where M_n is a connected constant-degree graph that contains $\mathcal{H}_{n,\lfloor\frac{n-1}{2}\rfloor}$ as a subgraph.*

Proof. Consider a set I of all the horizontal edges of $\mathcal{W}_n$ that belong to odd columns (on fig. 4 and 5 edges from I are red). I is a matching, i.e. no two edges from I are incident to the same vertex. If we contract all edges from I, we get the graph M_n that for odd n

coincides with $\mathcal{H}_{n,\frac{n-1}{2}}$ and for even n coincides with a graph that is obtained from $\mathcal{H}_{n,\lfloor\frac{n}{2}\rfloor}$ by the removal of several edges from the last vertical (see fig. 4 and 5). For every $e \in I$ we denote its left vertex by u_e and the right vertex by v_e. Let E_{u_e} be the set of edges of $\mathcal{W}_n$ incident to u_e except e. Let τ_e denote a CNF formula encoding $\bigoplus_{f \in E_{u_e}} x_f = f(u_e)$.

Consider a game tree T for the derivation of the Tseitin tautology $\neg\,\mathsf{T}(\mathcal{W}_n, f)$ of size S and depth d. To every formula used in this tree we apply the substitution that replaces every occurrence of x_e with τ_e. We denote the resulting tree by T'.

Notice that T' is a correct game tree of a derivation $\vdash_{d+\mathcal{O}(1)} \neg F$, where F is obtained from $\mathsf{T}(\mathcal{W}_n, f)$ by the same substitution. The depth of this derivation is increased by at most a constant since in several leaves we hang a formula of constant depth; here we also use that I is a matching and thus we do not add new occurrences of variables corresponding edges from I. The size of τ_e is $\mathcal{O}(1)$, hence any formula from the derivation is increased in at most a constant factor, thus the size of the derivation defined by the tree T' is $\mathcal{O}(S)$.

We define a function f' on vertices of M_n as follows. If a vertex w of the graph M_n is obtained by merging the vertices w', w'' of the graph $\mathcal{W}_n$, then $f'(w) = (f(w') + f(w'')) \bmod 2$. If the vertex w of $\mathcal{H}_{n,\lfloor n/2\rfloor}$ is obtained from the vertex w of $\mathcal{W}_n$, then $f'(w) = f(w)$.

Now we show how to derive $\neg\,\mathsf{T}(M_n, f')$ from $\neg F$. $\mathsf{T}(\mathcal{W}_n, f)$ is a Tseitin formula and it has the following structure: $\bigwedge_{v \in V} \psi_v$, where V is the set of vertices of $\mathcal{W}_n$ and ψ_v is a CNF formula encoding a parity condition for the vertex v. F differs from $\mathsf{T}(\mathcal{W}_n, f)$ only in conditions corresponding to vertices that are incident to an edge from I (if n is even, then there are vertices in $\mathcal{W}_n$ that are not incident to any edge from I). Notice that F has the form $\bigwedge_{v \in V} \psi'_v$ where ψ'_v is obtained by substitution from ψ_v. Let $w = u_e$ for some $e \in I$, then the formula ψ'_w is identically true. If $w = v_e$, then the condition ψ'_w is equivalent to the parity condition of the merged vertex $\{u_e, v_e\}$ in the Tseitin formula $\mathsf{T}(M_n, f')$, but ψ'_w is not written in canonical form.

Since all degrees in M_n are at most 4, then by Lemma 9 there exists a derivation $\neg F \vdash_{d+\mathcal{O}(1)} \neg\,\mathsf{T}(M_n, f')$ of size $\mathrm{poly}(n)$. The claim follows by Lemma 2. ◀

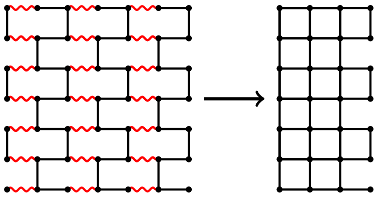


Figure 4 $\mathcal{W}_6$ is contracted to M_6.

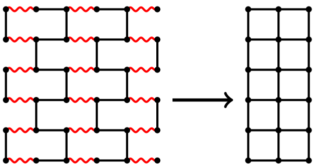


Figure 5 $\mathcal{W}_5$ is contracted to M_5.

3.3 Putting it all together

We use Håstad's Theorem from [22].

▶ **Theorem 16** ([22]). *There is a constant $K > 0$ such that for $d \leq \frac{K \log n}{\log\log n}$ any depth d derivation of $\neg\,\mathsf{T}(\mathcal{H}_{n,n}, f))$ has size at least $2^{n^{\Omega(1/d)}}$.*

▶ **Theorem 17.** *There exist constants $K > 0$ and $C > 0$ such that for every connected graph G of treewidth t and every $d \leq \frac{K \log n}{\log\log n} - C$, any depth d derivation of $\neg\,\mathsf{T}(G, f))$ has size at least $2^{t^{\Omega(1/d)}}$.*

Proof. Suppose that $\neg\,\mathsf{T}(G,f)$ have a derivation of size S and depth d. By Corollary 8 we know that G contains the wall $\mathcal{W}_r$ as a topological minor, where $r = \Omega(t^{1/37})$. Consider a sequence of operations (edge/vertex removals and suppressions) that transform G to $\mathcal{W}_r$. Assume that removals do not follow suppressions. And let G' be a subgraph of G that is obtained from G by application of all removals (hence, $\mathcal{W}_r$ can be obtained from G' by application of several suppressions).

By Lemma 11, for some f' there is a derivation of $\neg\,\mathsf{T}(G',f')$ of size $\mathrm{poly}(|\,\mathsf{T}(G,f)|) + S$ and depth $d + \mathcal{O}(1)$. Since $\mathcal{W}_r$ can be obtained from G' by application of several suppressions, G' is connected. Suppressions can not increase the degrees, hence all degrees in G' are at most 3. By Lemma 14, for some f'' there is a derivation of $\neg\,\mathsf{T}(\mathcal{W}_r, f'')$ of size $\mathrm{poly}(|\,\mathsf{T}(G,f)|) + S$ and depth $d + \mathcal{O}(1)$. By Lemma 15, for some f''' there is a derivation of $\neg\,\mathsf{T}(M_r, f''')$ of size $\mathrm{poly}(|\,\mathsf{T}(G,f)|) + \mathcal{O}(S)$ and depth $d + \mathcal{O}(1)$, where M_r is connected constant-degree graph containing $\mathcal{H}_{\lfloor (r-1)/2 \rfloor}$ as a subgraph. And finally by Lemma 11, for some f'''' there is a Frege derivation of $\mathsf{T}(\mathcal{H}_{\lfloor (r-1)/2 \rfloor}, f'''')$ of size $\mathrm{poly}(|\,\mathsf{T}(G,f)|) + \mathcal{O}(S)$ and depth $d + \mathcal{O}(1)$. Notice that S is the size of a derivation of $\neg\,\mathsf{T}(G,f)$, hence $S \geq |\,\mathsf{T}(G,f)|$. Thus, for some constants C and c there is a derivation of $\neg\,\mathsf{T}(\mathcal{H}_{\lfloor (r-1)/2 \rfloor}, f'''')$ of size S^c and depth $d + C$.

By Theorem 16, there is a constant K such that if $d + C \leq \frac{K \log n}{\log \log n}$, then $S^c \geq 2^{\lfloor (r-1)/2 \rfloor^{\Omega(1/(d+C))}}$. Hence $S \geq 2^{r^{\Omega(1/d)}}$ and, thus, $S \geq 2^{t^{\Omega(1/d)}}$. ◀

4 The Upper Bound

In this section we prove the following Theorem:

▶ **Theorem 18.** *Let $G(V,E)$ be a connected undirected graph and $\mathsf{T}(G,f)$ be an unsatisfiable Tseitin formula. Then for all large enough d the formula $\neg\,\mathsf{T}(G,f)$ has a derivation of depth d and size $2^{\mathrm{tw}(G)^{\mathcal{O}(1/d)}}\mathrm{poly}(|\,\mathsf{T}(G,f)|)$.*

In order to prove Theorem 18 we define a *compact representation* of parity by depth-d formulas, then we show that we can efficiently derive the sum of $\mathbb{F}_2$-linear equations using the compact representation of parities. And then we prove Theorem 18 using a tree-partition of the graph G.

4.1 A compact representation of parity

Let $t_1, t_2, \ldots, t_d$ be natural numbers, where d is a non-negative integer. Let $U_0, U_1, \ldots, U_d$ be partitions of a finite set F. We say that a list of partitions $U = (U_0, U_1, \ldots, U_d)$ is a $(t_1, \ldots, t_d)$*-refinement* of F if the following conditions hold:

1. U_0 consists of the only element $U_{0,1} = F$.
2. For every i, U_{i+1} is a subpartition of U_i such that every element of U_i is split into t_{i+1} parts. Hence, U_i split F into m_i parts: $U_{i,1}, U_{i,2}, \ldots, U_{i,m_i}$, where $m_i = \prod_{j=1}^{i} t_j$.
3. All elements of U_d have cardinality at most 1.

Let U be a $(t_1, \ldots, t_d)$-refinement of a set F and let $U_{i,j}$ be one of the blocks of this refinement. Then U induces on each of the blocks U_{ij} a $(t_{i+1}, \ldots, t_d)$-refinement U' which is obtained by restricting $U_i, \ldots, U_d$ to the set U_{ij}. U' is called a *sub-refinement* of U_{ij} in U.

▶ **Lemma 19.** *Let F be a set of size n and $d \geq 0$ be an integer. Let $t_1, \ldots, t_d$ be integers such that $t_1 \cdot t_2 \cdot \ldots \cdot t_d \geq n$. Then there exists a $(t_1, \ldots, t_d)$-refinement U of F.*

For $a \in \{0,1\}$ and natural number n we define a Boolean function $\mathrm{PARITY}_n^a : \{0,1\}^n \to \{0,1\}^n$ such that $\mathrm{PARITY}_n^a(x_1, \ldots, x_n) = 1$ iff $\bigoplus_{i=1}^{n} x_i = a$ for all $x_1, \ldots, x_n \in \{0,1\}$.

▶ **Lemma 20.** *Let n and d be positive integers and U be a $(t_1, t_2, \dots, t_d)$-refinement of $[n]$. Then there exists a formula representing PARITY_n^b of depth at most $3d + 1$ and of size $\prod_{i=1}^{d} 2^{t_i+1} t_i$.*

Proof. Let us prove by backward induction on i from d to 0 that for every $j \in [\prod_{k=1}^{i} t_k]$, there is a formula representing $\bigoplus_{k \in U_{i,j}} x_k$ of depth $3(d-i)$ and of size $\prod_{q=i+1}^{d} 2^{t_q+1} t_q$. If $i = d$, then $|U_{d,j}| \leq 1$, hence $\bigoplus_{k \in U_{i,j}} x_k$ is either 0 or a variable x_k and thus has size 1 and depth 0.

Assume that $i < d$. Let $\ell_1, \ell_2, \dots, \ell_{t_{i+1}}$ be such that $U_{i,j} = U_{i+1,\ell_1} \sqcup U_{i+1,\ell_2} \sqcup \dots \sqcup U_{i+1,\ell_{t_{i+1}}}$. Let for $r \in [t_{i_1}]$, β_r be a representation of $\bigoplus_{k \in U_{i+1,\ell_r}} x_k$ of size $\prod_{q=i+2}^{d} 2^{t_q+1} t_q$ and depth $3(d-i-1)$ that exists by the induction hypothesis. Consider a CNF-representation of $\beta_1 \oplus \dots \oplus \beta_{t_{i+1}}$: $\bigoplus_{k \in U_{i,j}} x_k = \bigwedge_{\substack{S \subseteq \{1,\dots,t_{i+1}\} \\ |S| \bmod 2 = 0}} \left(\bigvee_{s \in S} \neg \beta_s \vee \bigvee_{s \notin S} \beta_s \right)$. After the substitution of the representations of $\beta_1, \dots, \beta_{t_{i+1}}$ we obtain a formula of size at most $2^{t_{i+1}} t_{i+1} \cdot \prod_{q=i+2}^{d} 2^{t_q+1} t_q + 2^{t_{i+1}} t_{i+1} \leq \prod_{q=i+1}^{q} 2^{t_q+1} t_q$ and of depth $3(d-i-1) + 3 = 3(d-i)$.

Therefore we have constructed a representation of PARITY_n^1 of the needed size and depth. The representation of PARITY_n^0 could be constructed as $\neg\varphi$ where φ is the obtained representation of PARITY_n^1. ◀

We call the representation of PARITY_n^a obtained by Lemma 20 *the compact representation* of PARITY_n^a with respect to a $(t_1, \dots, t_d)$-refinement U.

Let us define for $S \subseteq [n]$ and for $a \in \{0, 1\}$, $\mathrm{PARITY}_{n,S}^a(x_1, \dots, x_n) = (\neg a) \oplus \bigoplus_{i \in S} x_i$. We define a compact representation of $\mathrm{PARITY}_{n,S}^a$ with respect to a $(t_1, \dots, t_d)$-refinement U as the result of substitutions $x_j := \mathbf{0}$ for all $j \notin S$ to the compact representation of PARITY_n^a with respect to U. We denote the compact representation of $\mathrm{PARITY}_{n,S}^a(x_1, x_2, \dots, x_n)$ w.r.t. U by $\Phi^a(S, U)$.

▶ **Lemma 21.** *Let U be a $(t_1, \dots, t_d)$-refinement of $[n]$ and U' be a sub-refinement of U_{ij} in U. Then for every $S \subseteq U_{ij}$ there exists a derivation $\Phi^a(S, U') \vdash_{3d + \mathcal{O}(1)} \Phi^a(S, U)$ of size at most $4|\Phi^a(S, U)|^3$.*

4.2 Summation of linear equations

Let $S \triangle T$ be the symmetric difference of sets S and T i.e. $S \triangle T = (S \cup T) \setminus (S \cap T)$.

▶ **Lemma 22.** *Let U be a $(t_1, \dots, t_d)$-refinement of $[n]$. Let $S_1, S_2, \dots, S_k \subseteq [n]$ and $a_1, \dots, a_k \in \{0, 1\}$. Then there exists a constant c such that:*

1. *There exists a derivation $\Phi^{a_1}(S_1, U), \Phi^{a_2}(S_2, U), \dots, \Phi^{a_k}(S_k, U) \vdash_{3d + \mathcal{O}(1)} \Phi^{a_1 \oplus \dots \oplus a_k}(S_1 \triangle \dots \triangle S_k, U)$ of size at most $c \cdot k \cdot |\Phi^1(\varnothing, U)|^6$.*
2. *If $\bigwedge_{i \in [k]} \left(\bigoplus_{j \in S_i} x_j = a_i \right)$ is unsatisfiable then there exists a derivation $\Phi^{a_1}(S_1, U), \Phi^{a_2}(S_2, U), \dots, \Phi^{a_k}(S_k, U) \vdash_{3d + \mathcal{O}(1)} \mathbf{0}$ of size at most $c \cdot k \cdot |\Phi^1(\varnothing, U)|^6$.*

4.3 Tree-partition width

Let $G(V, E)$ be an undirected graph and $S_1, \dots, S_m$ be a partition of V. $S_1, \dots, S_m$ is a tree-partition of G if there exists a tree $T([m], E_T)$ such that every edge e of G connects either two vertices from the same part S_i or connects a vertex from S_i and a vertex from S_j, where i and j are adjacent in T, i.e. $(i, j) \in E_T$. A *width* of a tree-partition $S_1, S_2, \dots, S_m$ is the size of the largest set S_i for $i \in [m]$. A *tree-partition width* of a graph G is the smallest width among all tree-partitions of G. We denote the tree-partition width of G by $\mathrm{tpw}(G)$.

If we add a new vertex in the middle of every edge (i, j) of the tree T and put the set $S_i \cup S_j$ on it, we will get a tree decomposition of G, hence $\mathrm{tw}(G) \leq 2\mathrm{tpw}(G) - 1$.

The following theorem shows an inequality in the other direction.

▶ **Theorem 23** ([38]). *If* $\mathrm{tw}(G) \geq 1$*, then* $\mathrm{tpw}(G) \leq 10\Delta(G)\mathrm{tw}(G)$*, where* $\Delta(G)$ *is the maximum degree of* G*.*

So, $\mathrm{tw}(G)$ and $\mathrm{tpw}(G)$ coincide up to a multiplicative constants for constant degree graphs.

▶ **Theorem 24.** *Let* $G(V, E)$ *be a connected graph and let a Tseitin formula* $\mathsf{T}(G, f)$ *be unsatisfiable. Then there exists a derivation* $\vdash_{3d+\mathcal{O}(1)} \mathsf{T}(G, f)$ *of size at most* $\mathrm{poly}(|\mathsf{T}(G, f)|) \cdot 2^{(\mathrm{tpw}(G)\Delta(G))^{\mathcal{O}(1/d)}}$*, where* $\Delta(G)$ *is the maximum degree of* G*.*

Proof. Let $S_1, \ldots, S_m$ be a tree-partition of G with width $\mathrm{tpw}(G)$ and let $T([m], E_T)$ be the corresponding tree. W.l.o.g. we assume that T is a rooted tree with the root m; for all $i \in [m-1]$, $p(i)$ denotes its parent and for all $i \in [m]$, $\mathrm{s}(i)$ denotes the set of direct successors of i . W.l.o.g. we assume that $p(i) > i$ for all $i \in [m-1]$.

Since $\mathsf{T}(G, f)$ is unsatisfiable and G is connected, $\bigoplus_{v \in V} f(v) = \bigoplus_{i \in [m]} \bigoplus_{v \in S_i} f(v) = 1$. We consider the sum $\bigoplus_{i \in [m]} \bigoplus_{e \in E(S_i, V \setminus S_i)} x_e$. Since each x_e occurs in the sum exactly twice, the sum (modulo 2) is 0 for all values of x_e. Then for each assignment to $\{x_e\}_{e \in E}$ there exists i_0 such that $\bigoplus_{v \in S_{i_0}} f(v) \neq \bigoplus_{e \in E(S_{i_0}, V \setminus S_{i_0})} x_e$. The first part of Pavel's strategy is to find such i_0.

Pavel will request parity of the sum of all edges between S_i and S_j for all $(i, j) \in E_T$. In order to represent these formulas in a compact way we now define m different $(t_1, \ldots, t_d)$-refinements $W^1, \ldots, W^m$; for every i, W^i is a refinement of the set $E\left(S_i, \bigcup_{j \in \mathrm{s}(i)} S_j\right)$ of all edges connecting a vertex from S_i with a vertex from $\bigcup_{j \in s(i)} S_j$. We construct appropriate refinements W^i later.

Pavel asks Sam the values of $\bigoplus_{e \in E(S_i, S_{p(i)})} x_e$ represented as $\Phi^1\left(E\left(S_i, S_{p(i)}\right), W^{p(i)}\right)$ for $i \in [m-1]$ in the increasing order until he finds i_0 such that $\bigoplus_{e \in E(S_{i_0}, V \setminus S_{i_0})} x_e \neq \bigoplus_{v \in S_{i_0}} f(v)$.

At the moment when Sam has answered the value of $\Phi^1\left(E\left(S_i, S_{p(i)}\right), W^{p(i)}\right)$ the values of $\bigoplus_{e \in E(S_i, S_j)} x_e$ for each j such that $(i, j) \in E_T$ are all determined, thus, the value of $\bigoplus_{e \in E(S_i, V \setminus S_i)} x_e$ is determined. If $\bigoplus_{e \in E(S_i, V \setminus S_i)} x_e \neq \bigoplus_{v \in S_i} f(v)$ Pavel proceeds to the next part of his strategy. Otherwise he continues to ask Sam similar questions corresponding to the vertices with larger indices.

Now we describe the strategy of Pavel in case if he finds i_0. We are going to describe this case in terms of derivation using Lemma 2 multiple times. Consider a linear system that consists of the equation $\bigoplus_{e \in E(S_{i_0}, V \setminus S_{i_0})} x_e = 1 \oplus \bigoplus_{v \in S_{i_0}} f(v)$ and all parity conditions of $\mathsf{T}(G, f)$ of the vertices from S_{i_0}. This linear system is unsatisfiable. We are going to use Lemma 22. In order to do it we need to derive the representations of these linear equations w.r.t. some refinement Q of a superset of $E(S_{i_0}, V)$.

Let for $i \in [m]$, U^i be a $(t_1, t_2, \ldots, t_d)$-refinement of the set $E(S_i)$ of all edges connecting two vertices from S_i (we construct these refinements in the end of the proof together with the refinements W^i). Let us define a $(3, t_1, \ldots, t_d)$-refinement Q as a union of $(t_1, \ldots, t_d)$-refinements $W^{i_0}, W^{p(i_0)}$ and U^{i_0} such that $Q_1 = \{E(S_{i_0}, \bigcup_{j \in \mathrm{s}(i_0)} S_j), E(S_{p(i_0)}, \bigcup_{j \in \mathrm{s}(p(i_0))} S_j), E(S_{i_0})\}$ and for every $j \in \{2, 3, \ldots, d+1\}$, Q_j is the union of $W^{i_0}_{j-1}, W^{p(i_0)}_{j-1}$ and $U^{i_0}_{j-1}$.

Let a_j be Sam's answer to the question $\bigoplus_{e \in E(S_{i_0}, S_j)} x_e$ for each j that is a neighbour of i_0 in T, hence we may assume that $\Phi^{a_j}(E(S_{i_0}, S_j), W^{i_0})$ for $j \in \mathrm{s}(i_0)$ and $\Phi^{a_{p(i_0)}}(E(S_{i_0}, E_{p(i_0)}),$

$W^{p(i_0)}$) are already derived. By Lemma 21, we derive $\Phi^{a_j}(E(S_{i_0}, S_j), Q)$ from $\Phi^{a_j}(E(S_{i_0}, S_j), W^{i_0})$ for $j \in \mathrm{s}(i_0)$ and $\Phi^{a_{p(i_0)}}(E(S_{i_0}, S_{p(i_0)}), Q)$ from $\Phi^{a_{p(i_0)}}(E(S_{i_0}, S_{p(i_0)}), W^{p(i_0)})$, where a_j are Sam's answers to the corresponding questions.

By the first part of Lemma 22 we derive $\Phi^{1\oplus\left(\bigoplus_{v\in S_{i_0}} f(v)\right)}(E(S_{i_0}, V \setminus S_{i_0}), Q)$ from the set of formulas $\{\Phi^{a_j}(E(S_{i_0}, S_j), Q) \mid (i_0, j) \in E_T\}$. We assume that the parity conditions of the vertices of G in $\mathsf{T}(G, f)$ represented as CNF are asked at the beginning of the game i.e. for each $v \in V$ we know that the CNF representation of $\bigoplus_{u:(u,v)\in E} x_e$ is true (if any clause of $\mathsf{T}(G, f)$ is false Pavel queries all subformulas of $\mathsf{T}(G, f)$ except subformulas of the clauses and gets an immediate contradiction, if any of the parity conditions is false it yields an immediate contradiction with the corresponding subset of clauses). Thus, by Lemma 4 we derive the representations of parity conditions of the vertices from S_{i_0} w.r.t. Q. Since the corresponding linear system is unsatisfiable, using the second part of Lemma 22 we get a contradiction.

▷ Claim 25. The size of the described game tree is at most $m \cdot 2^{3\Delta(G)}\Delta^2(G)\mathrm{tpw}(G)2^{\mathcal{O}\left(\sum_{i=1}^{d} t_i\right)}$.

Let us choose $t_i = (\Delta(G)\mathrm{tpw}(G))^{2/d}$ for all $i \in [d]$. Since $|S_i| \le \mathrm{tpw}(G)$, $|E(S_i)| + \left|E\left(S_i, \bigcup_{j\in \mathrm{s}(i)} S_j\right)\right| \le \Delta(G)\mathrm{tpw}(G)$. Hence, the condition $\prod_{i=1}^{d} t_i \ge \Delta(G)\mathrm{tpw}(G) \ge |E(S_i)| + \left|E\left(S_i, \bigcup_{j\in \mathrm{s}(i)} S_j\right)\right|$ holds and, thus, for all $i \in [m]$ the refinements U^i, W^i exist by Lemma 19. If we substitute choosen values in the bound from Clam 25, we get the upper bound $m \cdot 2^{\mathcal{O}\left(3\Delta(G)+d(\Delta(G)\mathrm{tpw}(G))^{2/d}\right)} = \mathrm{poly}(|\mathsf{T}(G, f)|) \cdot 2^{(\Delta(G)\mathrm{tpw}(G))^{\mathcal{O}(1/d)}}$. ◀

Now we are ready to prove Theorem 18.

Proof of Theorem 18. Theorem 24 and Theorem 23 imply that there exists a constant c and a derivation $\vdash_{3d+\mathcal{O}(1)} \neg\mathsf{T}(G, f)$ of size at most $\mathrm{poly}(|\mathsf{T}(G, f)|)2^{(10\Delta^2(G)\mathrm{tw}(G))^{c/d}}$. If $\mathrm{tw}(G) > \Delta(G)$ then we can rewrite our upper bound on the size as $\mathrm{poly}(|\mathsf{T}(G, f)|)2^{(10\mathrm{tw}(G))^{3c/d}}$. If $\mathrm{tw}(G) > 1$ then it is $\mathrm{poly}(|\mathsf{T}(G, f)|)2^{(\mathrm{tw}(G))^{\mathcal{O}(1/d)}}$. If $\mathrm{tw}(G) = 1$ then it is simply $\mathrm{poly}(|\mathsf{T}(G, f)|)$. Otherwise if $\mathrm{tw}(G) \le \Delta(G)$ we can rewrite the upper bound as $\mathrm{poly}(|\mathsf{T}(G, f)|)\cdot\ 2^{(10\Delta(G))^{3c/d}} = \mathrm{poly}(|\mathsf{T}(G, f)|)$ if $3c/d \le 1$. Thus, for $d \ge 3c$ the upper bound is $\mathrm{poly}(|\mathsf{T}(G, f)|)\cdot\ 2^{\mathrm{tw}(G)^{3c/d}}$. Therefore, for the both cases we have the needed upper bound. ◀

References

1 Miklós Ajtai. The Complexity of the Pigeonhole Principle. *Combinatorica*, 14(4):417–433, 1994. doi:10.1007/BF01302964.

2 Michael Alekhnovich and Alexander A. Razborov. Satisfiability, Branch-Width and Tseitin tautologies. *Computational Complexity*, 20(4):649–678, 2011. doi:10.1007/s00037-011-0033-1.

3 Paul Beame, Chris Beck, and Russell Impagliazzo. Time-Space Trade-offs in Resolution: Superpolynomial Lower Bounds for Superlinear Space. *SIAM J. Comput.*, 45(4):1612–1645, 2016. doi:10.1137/130914085.

4 Paul Beame and Toniann Pitassi. Simplified and Improved Resolution Lower Bounds. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 274–282. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548486.

5 Stephen Bellantoni, Toniann Pitassi, and Alasdair Urquhart. Approximation and Small-Depth Frege Proofs. *SIAM J. Comput.*, 21(6):1161–1179, 1992. doi:10.1137/0221068.

6 Rémy Belmonte, Archontia Giannopoulou, Daniel Lokshtanov, and Dimitrios M. Thilikos. The Structure of of W_4-Immersion-Free Graphs. *ICGT 2014*, Arxiv: 1602.02002, February 2016. arXiv:1602.02002.

7 Eli Ben-Sasson. Hard examples for the bounded depth Frege proof system. *Computational Complexity*, 11(3-4):109–136, 2002. doi:10.1007/s00037-002-0172-5.

8 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *J. ACM*, 48(2):149–169, 2001. doi:10.1145/375827.375835.

9 Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. A lower bound for the pigeonhole principle in tree-like Resolution by asymmetric Prover-Delayer games. *Inf. Process. Lett.*, 110(23):1074–1077, 2010. doi:10.1016/j.ipl.2010.09.007.

10 Samuel R. Buss. Polynomial Size Proofs of the Propositional Pigeonhole Principle. *J. Symb. Log.*, 52(4):916–927, 1987. doi:10.2307/2273826.

11 Julia Chuzhoy. Excluded Grid Theorem: Improved and Simplified. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 645–654. ACM, 2015. doi:10.1145/2746539.2746551.

12 Stephen A. Cook and Robert A. Reckhow. The Relative Efficiency of Propositional Proof Systems. *J. Symb. Log.*, 44(1):36–50, 1979. doi:10.2307/2273702.

13 Stefan S. Dantchev and Søren Riis. Tree Resolution Proofs of the Weak Pigeon-Hole Principle. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity, Chicago, Illinois, USA, June 18-21, 2001*, pages 69–75. IEEE Computer Society, 2001. doi:10.1109/CCC.2001.933873.

14 Zdenek Dvorák and Paul Wollan. A Structure Theorem for Strong Immersions. *Journal of Graph Theory*, 83(2):152–163, 2016. doi:10.1002/jgt.21990.

15 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.

16 Nicola Galesi, Dmitry Itsykson, Artur Riazanov, and Anastasia Sofronova. Bounded-depth Frege complexity of Tseitin formulas for all graphs. Technical Report 19-069, Electronic Colloquium on Computational Complexity (ECCC), 2019. URL: https://eccc.weizmann.ac.il/report/2019/069.

17 Nicola Galesi, Navid Talebanfard, and Jacobo Torán. Cops-Robber Games and the Resolution of Tseitin Formulas. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 311–326. Springer, 2018. doi:10.1007/978-3-319-94144-8_19.

18 Ludmila Glinskih and Dmitry Itsykson. On Tseitin formulas, read-once branching programs and treewidth. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:20, 2019. URL: https://eccc.weizmann.ac.il/report/2019/020.

19 Armin Haken. The Intractability of Resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

20 Daniel J. Harvey and David R. Wood. The treewidth of line graphs. *Journal of Combinatorial Theory, Series B*, 132:157–179, 2018. doi:10.1016/j.jctb.2018.03.007.

21 J. Hastad. *Computational Limitations of Small-Depth Circuits.* MIT press, 1987.

22 Johan Håstad. On Small-Depth Frege Proofs for Tseitin for Grids. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 97–108. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.18.

23 Dmitry Itsykson and Vsevolod Oparin. Graph Expansion, Tseitin Formulas and Resolution Proofs for CSP. In Andrei A. Bulatov and Arseny M. Shur, editors, *Computer Science - Theory and Applications - 8th International Computer Science Symposium in Russia, CSR*

2013, Ekaterinburg, Russia, June 25-29, 2013. Proceedings, volume 7913 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2013. doi:10.1007/978-3-642-38536-0_14.

24 Dmitry Itsykson, Vsevolod Oparin, Mikhail Slabodkin, and Dmitry Sokolov. Tight Lower Bounds on the Resolution Complexity of Perfect Matching Principles. *Fundam. Inform.*, 145(3):229–242, 2016. doi:10.3233/FI-2016-1358.

25 S. Jukna. *Boolean Function Complexity: Advances and Frontiers*. Springer, 2012.

26 Jan Krajícek, Pavel Pudlák, and Alan R. Woods. An Exponenetioal Lower Bound to the Size of Bounded Depth Frege Proofs of the Pigeonhole Principle. *Random Struct. Algorithms*, 7(1):15–40, 1995. doi:10.1002/rsa.3240070103.

27 Toniann Pitassi, Paul Beame, and Russell Impagliazzo. Exponential Lower Bounds for the Pigeonhole Principle. *Computational Complexity*, 3:97–140, 1993. doi:10.1007/BF01200117.

28 Toniann Pitassi, Benjamin Rossman, Rocco A. Servedio, and Li-Yang Tan. Poly-logarithmic Frege depth lower bounds via an expander switching lemma. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 644–657. ACM, 2016. doi:10.1145/2897518.2897637.

29 Pavel Pudlák and Samuel R. Buss. How to lie without being (easily) convicted and the lengths of proofs in propositional calculus. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic*, pages 151–162, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

30 Ran Raz. Resolution lower bounds for the weak pigeonhole principle. *J. ACM*, 51(2):115–138, 2004. doi:10.1145/972639.972640.

31 A. Razborov. Lower Bounds on the Size of Bounded Depth Networks Over a Complete Basis with Logical Addition. *Matematicheskie Zametki*, 41:598–607, 1987.

32 Alexander A. Razborov. Resolution lower bounds for the weak functional pigeonhole principle. *Theor. Comput. Sci.*, 303(1):233–243, 2003. doi:10.1016/S0304-3975(02)00453-X.

33 Alexander A. Razborov. Resolution lower bounds for perfect matching principles. *J. Comput. Syst. Sci.*, 69(1):3–27, 2004. doi:10.1016/j.jcss.2004.01.004.

34 Roman Smolensky. Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 77–82. ACM, 1987. doi:10.1145/28395.28404.

35 G.S. Tseitin. On the complexity of derivation in the propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic Part II*. A. O. Slisenko, editor, a968.

36 Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987. doi:10.1145/7531.8928.

37 Alasdair Urquhart and Xudong Fu. Simplified Lower Bounds for Propositional Proofs. *Notre Dame Journal of Formal Logic*, 37(4):523–544, 1996. doi:10.1305/ndjfl/1040046140.

38 David R. Wood. On tree-partition-width. *European Journal of Combinatorics*, 30(5):1245–1253, 2009. Part Special Issue on Metric Graph Theory. doi:10.1016/j.ejc.2008.11.010.

On the Expressivity of Linear Recursion Schemes

Pierre Clairambault
Univ Lyon, EnsL, UCBL, CNRS, LIP, F-69342, LYON Cedex 07, France

Andrzej S. Murawski
Department of Computer Science, University of Oxford, UK

Abstract

We investigate the expressive power of higher-order recursion schemes (HORS) restricted to linear types. Two formalisms are considered: multiplicative additive HORS (MAHORS), which feature both linear function types and products, and multiplicative HORS (MHORS), based on linear function types only.

For MAHORS, we establish an equi-expressivity result with a variant of tree-stack automata. Consequently, we can show that MAHORS are strictly more expressive than first-order HORS, that they are incomparable with second-order HORS, and that the associated branch languages lie at the third level of the collapsible pushdown hierarchy.

In the multiplicative case, we show that MHORS are equivalent to a special kind of pushdown automata. It follows that any MHORS can be translated to an equivalent first-order MHORS in polynomial time. Further, we show that MHORS generate regular trees and can be translated to equivalent order-0 HORS in exponential time. Consequently, MHORS turn out to have the same expressive power as 0-HORS but they can be exponentially more concise.

Our results are obtained through a combination of techniques from game semantics, the geometry of interaction and automata theory.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases higher-order recursion schemes, linear logic, game semantics, geometry of interaction

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.50

Funding *Pierre Clairambault*: Supported by ANR RAPIDO (ANR-14-CE25-0007) and Labex MiLyon (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007), operated by the French National Research Agency (ANR).
Andrzej S. Murawski: Supported by a Royal Society Leverhulme Trust Senior Research Fellowship (LT170023).

Acknowledgements We would like to thank Sylvain Salvati for consultations on formal languages.

1 Introduction

Higher-order recursion schemes (HORS) have recently emerged as a promising technique for model-checking higher-order programs [17]. Linear higher-order recursion schemes (LHORS) were introduced in [5] to facilitate a finer analysis of HORS by mixing intuitionistic and linear types. In this paper, we investigate the expressivity of their purely linear fragment.

First, we consider *multiplicative additive* HORS (MAHORS), which in addition to the linear function types ($\multimap$) feature product types (&), and thus allow for sharing but not re-use. We show that MAHORS are equivalent to a tree-generating variant of tree-stack automata (TSA), originally introduced to capture multiple context-free languages in the word language setting [7]. The translation from MAHORS to TSA amounts to representing the game semantics of MAHORS in the spirit of abstract machines derived from Girard's Geometry of Interaction (GoI) [11, 6]. The GoI view of computation makes it possible to interpret computation as a token machine that traverses a graph strongly related to the syntactic structure of the term. Somewhat suprisingly, so far this nearly automata-theoretic

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 50; pp. 50:1–50:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$\frac{}{\Gamma, x : \varphi \mid \Delta \vdash x : \varphi} \qquad \frac{}{\Gamma \mid \Delta, x : \varphi \vdash x : \varphi} \qquad \frac{\Gamma \mid \Delta \vdash t : \varphi_1 \mathbin{\&} \varphi_2}{\Gamma \mid \Delta \vdash \pi_i\, t : \varphi_i} \qquad \frac{}{\Gamma \mid \Delta \vdash \bot : \varphi}$$

$$\frac{\Gamma \mid \Delta_1 \vdash t : \varphi \multimap \psi \qquad \Gamma \mid \Delta_2 \vdash u : \varphi}{\Gamma \mid \Delta_1, \Delta_2 \vdash t\, u : \psi} \qquad \frac{\Gamma \mid \Delta, x : \kappa \vdash t : \varphi}{\Gamma \mid \Delta \vdash \lambda x^{\kappa}.t : \kappa \multimap \varphi} \qquad \frac{\Gamma \mid \Delta \vdash t_i : \varphi_i \qquad (i \in \{1,2\})}{\Gamma \mid \Delta \vdash \langle t_1, t_2 \rangle : \varphi_1 \mathbin{\&} \varphi_2}$$

Figure 1 Typing rules for the additive linear λ-calculus.

flavour of GoI has not been exploited to establish connections with automata models, and we believe we are the first to do so explicitly. As a consequence, we can conclude that the branch languages of trees generated by MAHORS are multiple context-free and, thus, that they belong to the third level of the collapsible pushdown hierarchy [12]. In addition, we show that MAHORS are strictly more expressive than first-order HORS[1], and that they are not comparable with second-order HORS.

Secondly, we consider *multiplicative* HORS (MHORS), featuring linear function types only. In this case, our earlier MAHORS-to-TSA translation specialises to a translation into a special kind of tree-generating pushdown automata (LPDA) in which reachable configurations must be reached in a unique run. We show that MHORS and LPDA are equi-expressive and, moreover, that any MHORS can be translated to an equivalent MHORS of order 1. Further, using reachability techniques for pushdown automata, we show that LPDA are equivalent to bounded pushdown automata that forget elements stored at the bottom of the stack after the stack height exceeds a certain depth. It follows that MHORS generate regular trees, though the MHORS representation may be exponentially more succinct than order-0 HORS.

2 Linear Recursion Schemes

In this section we introduce the object of study of this paper, MAHORS and MHORS.

The main ingredient of MAHORS is the *linear λ-calculus with products* – also called the *additive linear λ-calculus*, as the product is an additive connective in the sense of Linear Logic [10]. The following definitions follow [5], restricting type formers to linear connectives (note that [5] imposes some syntactic restrictions on the shape of types and terms that we can drop here to simplify presentation, as they play no role in the technical development).

Types are formed with the ground type o and the connectives $\multimap$ and &. We define the **typed terms** directly by the typing rules of Figure 1. Typing judgments have the form $\Gamma \mid \Delta \vdash t : \varphi$, where Γ and Δ are two lists of variable declarations. Intuitively, Δ is the main context containing variables that can be used at most once (such terms are often called *affine* but we opt for the name *linear* nonetheless). In contrast, Γ comprises *duplicable* variables that may be reused at will, as witnessed by the application rule. In M(A)HORS, Γ will be used only for terminal and non-terminal symbols. Linear λ-terms are equipped with standard reduction rules; we write $\triangleright_\beta$ for β-reduction for functions and products, whose definition can be found *e.g.* in [5]. Any term t has a normal form, written $\mathrm{BT}(t)$.

Trees arise as ground-type terms typable in replicable contexts representing a ranked alphabet. Recall that in HORS, a symbol b of arity n is represented as a constant $\mathsf{b} : o \to \cdots \to o \to o$ with n arguments. Here, a ranked alphabet Σ may be represented in two distinct

[1] Type order is defined by $ord(o) = 0$ and $ord(\theta \to \theta') = \max(ord(\theta) + 1, ord(\theta'))$. The order of a HORS is the highest order of (the types of) its non-terminals.

ways: *multiplicatively*, with $\mathsf{b} : o \multimap \ldots \multimap o \multimap o$, or *additively*, with $\mathsf{b} : \&_n o \multimap o$, where $\&_n o$ stands for $o \& \cdots \& o$ (n copies)[2]. The choice does not impact how finite trees are represented: in both cases a $\triangleright_\beta$-normal $\Sigma \mid \_ \vdash t : o$ (if not $\bot$) must start with a variable from Σ with some arity n, followed by n $\triangleright_\beta$-normal sub-trees; *i.e.* it represents a tree (with certain branches possibly leading to $\bot$). The multiplicative vs additive distinction matters in the definition of schemes, though: with additive typing, resources (variables) may be shared when calculating two sub-branches of an infinite tree, which is disallowed with multiplicative typing.

Linear recursion schemes consist of a system of recursive equations, where each clause is given by a λ-term with a restricted shape. A term $\Gamma \mid \_ \vdash t : \varphi$ is called **applicative** if it is $\triangleright_\beta$-normal, and has the form $\lambda x_1^{\varphi_1} \ldots \lambda x_n^{\varphi_n}.t'$ where t' has no abstraction.

▶ **Definition 1.** *A* ***Multiplicative Additive Recursion Scheme (MAHORS)*** *is a 4-tuple $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S\rangle$ where: (1) Σ is a ranked alphabet; (2) $\mathcal{N}$ is a finite set of typed* ***non-terminals****; we use upper-case letters $F, G, H, \ldots$ to range over them. We denote the type of F by $\mathcal{N}(F)$ and write $F : \mathcal{N}(F)$; (3) $S \in \mathcal{N}$ is a distinguished* ***start symbol*** *of type o; and (4) $\mathcal{R}$ is a function associating to each F in $\mathcal{N}$ an applicative term $\Sigma, \mathcal{N} \mid \_ \vdash \mathcal{R}(F) : \mathcal{N}(F)$, with Σ represented additively. A* ***MHORS*** *is defined as a MAHORS where Σ is represented multiplicatively and the typing of $\mathcal{N}$ does not involve products.*

If $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S\rangle$ is a MAHORS, then for each $F \in \mathcal{N}$ and $n \in \mathbb{N}$ there is $\Sigma \mid \_ \vdash \mathsf{unf}_n(F) : \mathcal{N}(F)$ defined by $\mathsf{unf}_0(F) = \bot$ and $\mathsf{unf}_{n+1}(F) = \mathcal{R}(F)[\mathsf{unf}_n(G)/G \mid G \in \mathcal{N}]$. The family $(\mathsf{unf}_i(F))_{i\in\mathbb{N}}$ forms a chain for $\leq$ defined as usual by $\bot \leq t$, closed by congruence. As evaluation is monotone, $(\mathrm{BT}(\mathsf{unf}_i(F))_{i\in\mathbb{N}}$ also forms a chain, hence it has a lub which may be defined as the ideal completion of finite normal terms $\Sigma \mid \_ \vdash t : o$ ordered by $\leq$. We may then define $\mathrm{BT}(\mathcal{G}) = \bigsqcup_{i\in\mathbb{N}} \mathrm{BT}(\mathsf{unf}_i(S))$, the **infinite tree generated by** $\mathcal{G}$.

Our schemes comprise an explicit divergence symbol $\bot$. This is unusual, but does not affect expressivity as it could always be defined with a new non-terminal with rule $\mathcal{R}(\Omega) = \Omega$. Finally, we identify silently trees and terms $\Sigma \mid \_ \vdash t : o$.

3 Finite Memory Game Semantics and Geometry of Interaction

Game semantics is a semantic technique to give a compositional interpretation of higher-order programs [14]. By presenting higher-order computation as a *game* between two players embodying the program and its execution environment (Player for the program, Opponent for the environment), it effectively reduces higher-order computation to an exchange of tokens between terms. At first forgetting recursion, we briefly review the interpretation of the linear λ-calculus with products in *simple games*, then introduce its refined interpretation as finite-memory strategies, which will inform the translation of M(A)HORS to TSA.

3.1 Games and strategies

A **game** is a tuple $A = \langle M_A, \lambda_A, P_A\rangle$ where M_A is a set of *moves*, $\lambda_A : M_A \to \{O, P\}$ is a *polarity function* (we write $M_A^O = \lambda_A^{-1}(\{O\})$ and $M_A^P = \lambda_A^{-1}(\{P\})$), and $P_A \subseteq M_A^*$ is a non-empty prefix-closed set of *valid plays*, whose elements are O-starting and alternating: if $s = s_1 \ldots s_n \in P_A$, then $\lambda_A(s_1) = O$ and $\lambda_A(s_i) \neq \lambda_A(s_{i+1})$. We write $\epsilon \in P_A$ for the empty play and $s \sqsubseteq s'$ for the prefix ordering.

[2] [5] considers also intermediate typings, but this does not contribute extra expressivity.

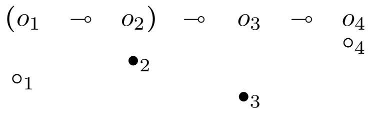


Figure 2 A play on $[\![(o \multimap o) \multimap o \multimap o]\!]$.

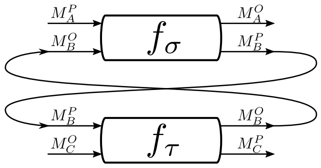


Figure 3 Composition of history-free skeletons.

Games represent *types*. Plays in a game for a type φ represent executions on φ following (for this paper) a call-by-name evaluation strategy. For instance, Figure 2 shows a play in the game for $(o \multimap o) \multimap o \multimap o$, read from top to bottom. We use indices on atom occurrences and moves for disambiguation, but the usual convention in game semantics is to signify the identity of moves simply by their position under the corresponding type component. After Opponent ($\circ$, the environment) starts computation by the initial move on the right, Player ($\bullet$, the program) responds by interrogating its function argument. Opponent, playing for this argument, calls its argument. Player terminates by calling its second argument. This play is, in fact, the maximal play of the interpretation of $\lambda f^{o \multimap o}.\, \lambda x^o.\, f\, x : (o \multimap o) \multimap o \multimap o$.

Each type φ may be interpreted as a game $[\![\varphi]\!]$. The game $[\![o]\!]$ has $M_{[\![o]\!]} = \{\circ\}$ with $\lambda(o) = O$, and $P_{[\![o]\!]} = \{\epsilon, \circ\}$. To match the type constructor $\multimap$, the **linear arrow game** $A \multimap B$ has as moves the tagged disjoint union $M_{A \multimap B} = M_A + M_B = \{1\} \times M_A \cup \{2\} \times M_B$ with polarity $\lambda_{A \multimap B}(1, a) = \overline{\lambda_A(a)}$ and $\lambda_{A \multimap B}(2, b) = \lambda_B(b)$, where $\overline{O} = P$ and $\overline{P} = O$. The plays $P_{A \multimap B}$ include all O-starting, alternating sequences $s \in M^*_{A \multimap B}$ such that the *restrictions* $s \upharpoonright A \in M^*_A$ and $s \upharpoonright B \in M^*_B$, defined in the obvious way, are in P_A and P_B respectively. Hence, $A \multimap B$ can be viewed as playing the two games A and B in parallel, with the polarity reversed in A, in such a way that any play must start in B and Player is able to switch between the components. With these definitions the reader can check that $[\![(o \multimap o) \multimap o \multimap o]\!] = ([\![o]\!] \multimap [\![o]\!]) \multimap ([\![o]\!] \multimap [\![o]\!])$ includes four moves corresponding to the four atom occurrences, and has only two maximal plays: the one in Figure 2, and $\circ_4 \bullet_3$.

The **tensor game** $A \otimes B$ has moves $M_{A \otimes B} = M_A + M_B$, polarity $\lambda_{A \otimes B}(1, a) = \lambda_A(a)$ and $\lambda_{A \otimes B}(2, b) = \lambda_B(b)$, and plays are those $s \in M^*_{A \otimes B}$ that are alternating, O-starting and such that $s \upharpoonright A \in P_A$ and $s \upharpoonright B \in P_B$. Dually to $\multimap$, it follows from the definition that here only O can change between components. The **product game** $A \,\&\, B$ has the same moves and polarity as $A \otimes B$, but only the plays where either $s \upharpoonright A$ or $s \upharpoonright B$ is empty. Hence, with their first move, Opponent fixes the component in which the rest of the game will be played.

A **strategy** σ on A, written $\sigma : A$, is $\sigma \subseteq P^{ev}_A$ (writing P^{ev}_A for the set of even-length plays) which is non-empty, closed under even-length prefix, and *deterministic*, in the sense that if $sab, sab' \in \sigma$, then $b = b'$. The interpretation of terms yields strategies; for instance

$$[\![\lambda f^{o \multimap o}.\, \lambda x^o.\, f\, x : (o \multimap o) \multimap o \multimap o]\!] = \{\epsilon,\ \circ_4 \bullet_2,\ \circ_4 \bullet_2 \circ_1 \bullet_3\}$$

is a strategy on $[\![(o \multimap o) \multimap o \multimap o]\!]$ with moves following the naming convention of Figure 2.

The interpretation of terms exploits a number of constructions on strategies. In particular, to compute the **composition** of $\sigma : A \multimap B$ and $\tau : B \multimap C$ we first let σ, τ interact by considering all sequences in $(M_A + M_B + M_C)^*$ whose restrictions to A, B and B, C are respectively in σ and τ; and then project those to $P_{A \multimap C}$ to obtain $\tau \circ \sigma : A \multimap C$. We omit the details [14]. Overall, the structure needed to interpret the linear λ-calculus with products is succinctly summarized by stating that games and strategies form a *symmetric monoidal closed category with products* [14] – to any $\_ \mid x_1 : \varphi_1, \ldots, x_n : \varphi_n \vdash t : \varphi$ this lets us associate $[\![t]\!] : \bigotimes_{1 \le i \le n} [\![\varphi_i]\!] \multimap [\![\varphi]\!]$ in such a way that this is invariant under reduction – note however that in this paper, we avoid the categorical language as much as possible.

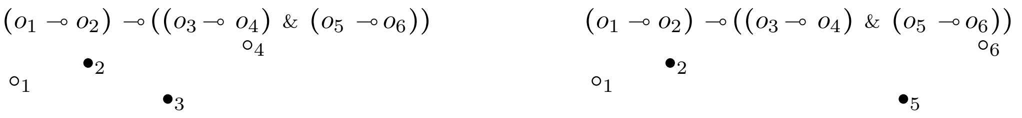

$(o_1 \multimap o_2) \multimap ((o_3 \multimap o_4) \,\&\, (o_5 \multimap o_6))$
$\circ_4$
$\bullet_2$
$\circ_1$
$\bullet_3$
$(o_1 \multimap o_2) \multimap ((o_3 \multimap o_4) \,\&\, (o_5 \multimap o_6))$
$\circ_6$
$\bullet_2$
$\circ_1$
$\bullet_5$

Figure 4 The two maximal plays of contraction on $[\![o \multimap o]\!]$.

3.2 History-free and finite memory strategies

A strategy $\sigma : A$ is **history-free** if its behaviour only depends on the last move, *i.e.* there is a partial function $f : M_A^O \rightharpoonup M_A^P$ such that for all $s \in \sigma$, for all $sa \in P_A$, we have $sab \in \sigma$ iff $f(a)$ is defined and $b = f(a)$. It is key in *AJM games* [1] that, without products, terms yield history-free strategies. If $\sigma : A$ is history-free, it is characterized by the corresponding partial function $f : M_A^O \rightharpoonup M_A^P$, known as its *history-free skeleton*. For instance, the strategy $[\![\lambda f^{o \multimap o}.\, \lambda x^o.\, f\, x]\!]$ with a unique maximal play in Figure 2, has history-free skeleton $\{\circ_4 \mapsto \bullet_2,\ \circ_1 \mapsto \bullet_3\}$.

One can also directly interpret terms as history-free skeletons: this is usually referred to as *Geometry of Interaction* [11], which has close ties with game semantics [3]. In particular, composition of history-free strategies can be performed directly on skeletons. If $\sigma : A \multimap B$ and $\tau : B \multimap C$ are history-free, their history-free skeletons, which have the types

$$f_\sigma : M_A^P + M_B^O \rightharpoonup M_A^O + M_B^P \qquad\qquad f_\tau : M_B^P + M_C^O \rightharpoonup M_B^O + M_B^P,$$

may be composed via *feedback* on B, pictured in Figure 3. For any Opponent move in $A \multimap C$, we apply the corresponding function f_σ or f_τ. As long as the response is in B, we keep applying f_σ and f_τ alternately. This process may stay in B forever (a *livelock*, in which case the composition $f_{\tau\circ\sigma}$ is undefined), but otherwise we eventually get a Player move in $A \multimap C$ as required; defining a partial function $f_{\tau\circ\sigma} : M_{A\multimap C}^O \rightharpoonup M_{A\multimap C}^P$. One may visualize a token entering on the left carrying an Opponent move, then bouncing in B until it eventually exits on the right. Other constructions used in the interpretation may be presented similarly, altogether giving (for the linear λ-calculus) a presentation of evaluation through a finite automaton called a *token machine*, where a token enters through an Opponent move, and bounces through the term until it eventually exits, giving the result of computation [18].

This is our starting point to represent evaluation of M(A)HORS via an automaton. However, there is an issue: strategies for linear λ-terms *with products* are not in general history-free. For instance, Figure 4 displays the two maximal plays of a *contraction/duplication* strategy $[\![\lambda f^{o\multimap o}.\, \langle f, f\rangle : (o \multimap o) \multimap ((o \multimap o) \,\&\, (o \multimap o))]\!]$. It reacts to $\circ_1$ differently depending on the history. To account for this, one may replace partial functions $f : M_A^O \rightharpoonup M_A^P$ with $f : M_A^O \times \mathcal{M} \rightharpoonup M_A^P \times \mathcal{M}$, *i.e.* *transducers*, where $\mathcal{M}$, the *memory*, is a finite set (see the *memoryful geometry of interaction* of [13] – however, we are not aware of this being used to define finite memory strategies). We give below a definition in this spirit, adapted to ease the translation to TSA and to deal with the branching in M(A)HORS due to *terminal symbols*.

We fix a ranked alphabet Σ (the multiplicative/additive distinction plays no role here).

▶ **Definition 2.** *A* ***transducer*** $\mathcal{T}$ *on a game A, written $\mathcal{T} : A$, is $\mathcal{T} = \langle \mathcal{M}_- \uplus \mathcal{M}_+, m_0, \delta_-, \delta_+\rangle$ where $\mathcal{M}_-$ is a finite set of* ***passive memory states*** *with a distinguished* ***initial memory state*** *$m_0 \in \mathcal{M}_-$, $\mathcal{M}_+$ is a finite set of* ***active memory states****, and* ***transition functions****:*

$$\begin{array}{lcrcl} \delta_- & : & \mathcal{M}_- \times M_A^O & \to & \mathcal{M}_+ \\ \delta_+ & : & \mathcal{M}_+ & \rightharpoonup & \mathcal{M}_+ \;+\; \mathcal{M}_- \times M_A^P \;+\; \{\mathsf{b}(m_1, \ldots, m_{|\mathsf{b}|}) \mid m_i \in \mathcal{M}_+,\ \mathsf{b} \in \Sigma\}\,. \end{array}$$

$$\delta_+^{\mathcal{T}\odot\mathcal{S}}((m_\mathcal{S}^-, m_\mathcal{T}^+)) = \begin{cases} (m_\mathcal{S}^-, m') & \text{if } \delta_+^\mathcal{T}(m_\mathcal{T}^+) = m' \\ \mathsf{b}((m_\mathcal{S}^-, m_1), \dots, (m_\mathcal{S}^-, m_{|\mathsf{b}|})) & \text{if } \delta_+^\mathcal{T}(m_\mathcal{T}^+) = \mathsf{b}(m_1, \dots, m_{|\mathsf{b}|}) \\ ((m_\mathcal{S}^-, m_\mathcal{T}^-), (2, c)) & \text{if } \delta_+^\mathcal{T}(m_\mathcal{T}^+) = (m_\mathcal{T}^-, (2, c)) \\ (\delta_-^\mathcal{S}(m_\mathcal{S}^-, (2, b)), m_\mathcal{T}^-) & \text{if } \delta_+^\mathcal{T}(m_\mathcal{T}^+) = (m_\mathcal{T}^-, (1, b)) \end{cases}$$

$$\delta_+^{\mathcal{T}\odot\mathcal{S}}((m_\mathcal{S}^+, m_\mathcal{T}^-)) = \begin{cases} (m', m_\mathcal{T}^-) & \text{if } \delta_+^\mathcal{S}(m_\mathcal{S}^+) = m' \\ \mathsf{b}((m_1, m_\mathcal{T}^-), \dots, (m_{|\mathsf{b}|}, m_\mathcal{T}^-)) & \text{if } \delta_+^\mathcal{S}(m_\mathcal{S}^+) = \mathsf{b}(m_1, \dots, m_{|\mathsf{b}|}) \\ ((m_\mathcal{S}^-, m_\mathcal{T}^-), (1, a)) & \text{if } \delta_+^\mathcal{S}(m_\mathcal{S}^+) = (m_\mathcal{S}^-, (1, a)) \\ (m_\mathcal{S}^-, \delta_-^\mathcal{T}(m_-^\mathcal{T}, (1, b))) & \text{if } \delta_+^\mathcal{S}(m_\mathcal{S}^+) = (m_\mathcal{S}^-, (2, b)) \end{cases}$$

Figure 5 Positive transitions of the composition of strategic transducers.

Any transducer $\mathcal{T}$ on $[\![o]\!]$ will be called **closed**. Apart from the forced initial $\delta_-(m_0, \circ)$, it is a finite tree-generating automaton, producing a tree $\mathsf{Tree}(\mathcal{T})$. But in general transducers may play on arbitrary games. In passive states, a transducer is waiting for an Opponent move, while in active states, it is performing internal computation that may result in a terminal symbol or in a Player move and the transition to a passive state. If $\delta_+(m) = \mathsf{b}(m_1, \dots, m_{|\mathsf{b}|})$, it produces the terminal symbol b; exploring the ith child results in continuing with m_i.

Like strategies, transducers can be *composed*.

▶ **Definition 3.** *Let $\mathcal{S} = (\mathcal{M}_-^\mathcal{S} \uplus \mathcal{M}_+^\mathcal{S}, m_0^\mathcal{S}, \delta_-^\mathcal{S}, \delta_+^\mathcal{S}) : A \multimap B$ and $\mathcal{T} = (\mathcal{M}_-^\mathcal{T} \uplus \mathcal{M}_+^\mathcal{T}, m_0^\mathcal{T}, \delta_-^\mathcal{T}, \delta_+^\mathcal{T}) : B \multimap C$ be transducers. The transducer $\mathcal{T} \odot \mathcal{S}$ on game $A \multimap C$ has $\mathcal{M}_-^{\mathcal{T}\odot\mathcal{S}} = \mathcal{M}_-^\mathcal{S} \times \mathcal{M}_-^\mathcal{T}$ and $\mathcal{M}_+^{\mathcal{T}\odot\mathcal{S}} = \mathcal{M}_+^\mathcal{S} \times \mathcal{M}_-^\mathcal{T} \uplus \mathcal{M}_-^\mathcal{S} \times \mathcal{M}_+^\mathcal{T}$, with initial state $(m_0^\mathcal{S}, m_0^\mathcal{T})$. The* transition function *is defined via $\delta_-^{\mathcal{T}\odot\mathcal{S}}((m_\mathcal{S}^-, m_\mathcal{T}^-), (2, c)) = (m_\mathcal{S}^-, \delta_-^\mathcal{T}(m_\mathcal{T}^-, (2, c)))$, $\delta_-^{\mathcal{T}\odot\mathcal{S}}((m_\mathcal{S}^-, m_\mathcal{T}^-), (1, a)) = (\delta_-^\mathcal{S}(m_\mathcal{S}^-, (1, a)), m_\mathcal{T}^-)$, and positive transitions given in Figure 5.*

Besides composition, all operations on strategies used in the interpretation of the linear λ-calculus with products have a counterpart on transducers. Altogether, for any $\Sigma \mid x_1 : \varphi_1, \dots, x_n : \varphi_n \vdash t : \varphi$, this yields a transducer $\langle t \rangle : \otimes_{1 \le i \le n} [\![\varphi_i]\!] \multimap [\![\varphi]\!]$. In particular, if $\Sigma \mid \_ \vdash t : o$, this yields a *closed* transducer $\langle t \rangle : [\![o]\!]$. It is obtained directly by induction on syntax following denotational semantics, and in particular in polynomial time. We can prove:

▶ **Proposition 4.** *For any $\Sigma \mid \_ \vdash t : o$, $\mathsf{Tree}(\langle t \rangle) = \mathrm{BT}(t)$.*

The proof works by linking transducers with game semantics. The simple game semantics presented above cannot directly deal with the presence of non-terminals replicable at will and the associated branching, so we must first extend it to "tree-generating game semantics". The details, though rather direct, are too lengthy for the paper, so we instead present the connection ignoring the terminal symbols.

Ignoring branching transitions, transducers generate strategies. Writing $m^- \overset{a}{\to} m^+$ when $\delta_-(m^-, a) = m^+$, $m_1^+ \to m_2^+$ when $\delta_+(m_1^+) = m_2^+$ and $m^+ \overset{b}{\to} m^-$ when $\delta_+(m^+) = (m^-, b)$; the set $\mathrm{Traces}(\mathcal{T})$ comprises all sequences $s_1 \dots s_{2n} \in M_A^*$ such that (with $m_0, \dots, m_n \in \mathcal{M}_-$)

$$m_0 \overset{s_1}{\to} \to^* \overset{s_2}{\to} m_1 \overset{s_3}{\to} \to^* \overset{s_4}{\to} m_2 \dots m_{n-1} \overset{s_{2n-1}}{\to} \to^* \overset{s_{2n}}{\to} m_n \,.$$

We say that $\mathcal{T}$ is a **strategic transducer** if for all $s \in \mathrm{Traces}(\mathcal{T}) \cap P_A$, if $sa \in P_A$ and $sab \in \mathrm{Traces}(\mathcal{T})$, then $sab \in P_A$. Then, $\mathrm{Traces}(\mathcal{T}) \cap P_A$ is a strategy written $\mathrm{Strat}(\mathcal{T})$. We say that $\sigma : A$ has **finite memory** if $\sigma = \mathrm{Strat}(\mathcal{T})$ for a strategic transducer $\mathcal{T}$. We also recover *history-free strategies* as those for which $\mathcal{M}_-$ is a singleton. For instance, the strategy in Figure 4 is generated using $\mathcal{M}_- = \{m_0, m_1\}$ and $\mathcal{M}_+ = \mathcal{M}_- \times M^O_{[\![o \multimap o]\!]}$, $\delta_-(m, a) = (m, a)$, $\delta_+(\_, \circ_4) = (m_0, \bullet_2)$, $\delta_+(\_, \circ_6) = (m_1, \bullet_2)$, $\delta_+(m_0, \circ_1) = (m_0, \bullet_3)$ and $\delta_+(m_1, \circ_1) = (m_1, \bullet_5)$.

Proposition 4 boils down to the fact that all constructions on transducers in the interpretation preserve strategic transducers, and match operations on strategies – for instance, $\mathrm{Strat}(\mathcal{T} \odot \mathcal{S}) = \mathrm{Strat}(\mathcal{T}) \circ \mathrm{Strat}(\mathcal{S})$. This entails that for all t, $\mathrm{Strat}(\langle t \rangle) = [\![t]\!]$. But for closed transducers $\langle t \rangle$ and tree-generating game semantics, $\mathsf{Tree}(\langle t \rangle) = \mathrm{Strat}(\langle t \rangle)$. Since game semantics is invariant under reduction, $[\![t]\!] = [\![\mathrm{BT}(t)]\!] = \mathrm{BT}(t)$, and Proposition 4 follows.

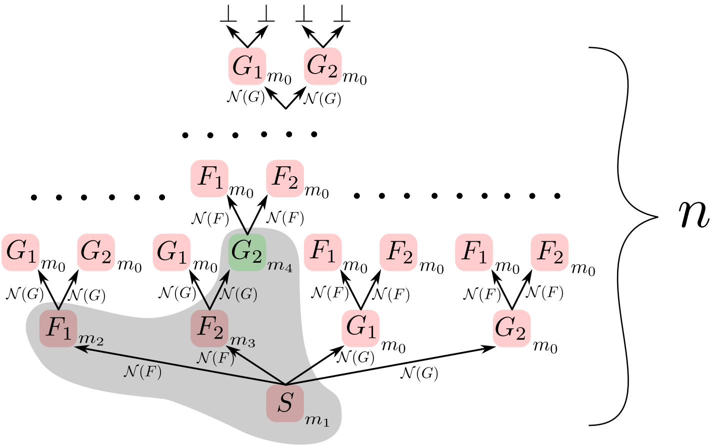


Figure 6 Illustration of a state of the n-th unfolding.

4 Game Semantics to TSA

The previous section lets us associate, to any $\Sigma \mid \_ \vdash t : o$, a finite tree-generating automaton. We extend this with recursion in two steps: first we evaluate finite unfoldings using finite automata, and then we build a single automaton with additional memory (a *Tree Stack Automaton*) whose runs amount to dynamically exploring these finite unfoldings.

4.1 Unfolding recursive calls

Let us fix a M(A)HORS $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$. By definition, for each $F \in \mathcal{N}$ we have $\Sigma, \mathcal{N} \mid \_ \vdash \mathcal{R}(F) : \mathcal{N}(F)$. Let $N \in \mathbb{N}$ be such that for all $F, G \in \mathcal{N}$, G appears at most N times in $\mathcal{R}(F)$. For all $F \in \mathcal{N}$, we choose a term $\Sigma \mid \mathcal{N}_1, \ldots, \mathcal{N}_N \vdash \mathcal{R}'(F) : \mathcal{N}(F)$ obtained by giving different names $G_1, \ldots, G_p$ ($p \leq N$) to all occurrences of $G \in \mathcal{N}$ in $\mathcal{R}(F)$. How these names are assigned does not matter. Although $\mathcal{R}'$ differs from $\mathcal{R}$, it can be equivalently used to define the finite approximations of $\mathrm{BT}(\mathcal{G})$. For each $F \in \mathcal{N}$ and $n \in \mathbb{N}$, we redefine $\Sigma \mid \_ \vdash \mathsf{unf}_n(F) : \mathcal{N}(F)$ by setting $\mathsf{unf}_0(F) = \bot$, and $\mathsf{unf}_{n+1}(F) = \mathcal{R}'(F)[\mathsf{unf}_n(G)/G_i \mid G \in \mathcal{N}_i, 1 \leq i \leq N]$. Although defined differently, this gives the same result as in Section 2.

But, unlike the original unfolding, this one can be replicated with strategic transducers. For each $F \in \mathcal{N}$, the interpretation of the previous section yields a strategic transducer:

$$\langle \mathcal{R}'(F) \rangle : \bigotimes_{1 \leq i \leq N} \bigotimes_{G \in \mathcal{N}} [\![\mathcal{N}(G)]\!] \multimap [\![\mathcal{N}(F)]\!] .$$

The unfolding above can then be replicated as follows.

▶ **Proposition 5.** *Setting $\mathcal{T}_F^0 = \bot$ with all positive transitions undefined, and $\mathcal{T}_F^{n+1} = \langle \mathcal{R}'(F) \rangle \odot (\bigotimes_{1 \leq i \leq N} \bigotimes_{G \in \mathcal{N}} \mathcal{T}_G^n) : [\![\mathcal{N}(F)]\!]$, for all $n \in \mathbb{N}$, we have $\mathsf{Tree}(\mathcal{T}_S^n) = \mathrm{BT}(\mathsf{unf}_n(S))$.*

Proof. By the substitution lemma for symmetric monoidal closed categories with products, syntactic substitution matches composition in the denotational model. It follows by induction that for all $F \in \mathcal{N}$, for all $n \in \mathbb{N}$, $\langle \mathsf{unf}_n(F) \rangle$ and $\mathcal{T}_F^n$ are transducers generating the same finite memory strategy. By Proposition 4, $\mathsf{Tree}(\mathcal{T}_S^n) = \mathsf{Tree}(\langle \mathsf{unf}_n(S) \rangle) = \mathrm{BT}(\mathsf{unf}_n(S))$. ◀

Figure 6 displays the structure of transducer compositions arriving at the finite tree automaton $\mathcal{T}_S^n$, for a M(A)HORS $\mathcal{G}$ where $\mathcal{R}(S)$ has two occurrences of F and two occurrences of G, $\mathcal{R}(F)$ has two occurrences of G, and $\mathcal{R}(G)$ has two occurrences of F. Each node stands for the matching strategic transducer (corresponding to a non-terminal), edges represent compositions. Running $\mathcal{T}_S^n$ passes control between the composed transducers, with always exactly one active after the initial transition. Figure 6 shows a possible state during a run: the grey area marks nodes that have already been explored. Outside of the grey area, the (local) transducer memory must be m_0. The green node is active, and all others passive. Following the transition function of $\langle\!\langle \mathcal{R}(G) \rangle\!\rangle$, we may next update the local memory m_4, produce a terminal and branch, or update to a passive state and send control up or down.

4.2 Tree Stack Automata

Now we give a single automaton with infinite memory whose bounded restrictions match the approximations above. It has a *stack* to deal with recursion, such that each state of the stack corresponds to a node in Figure 6. As these nodes stand for strategic transducers, they all have a finite memory. Accordingly, the automaton maintains a *store* associating, to each previously visited stack state/node, its *local memory*, accessed or updated only when visiting that node. We think of the store as a tree: the stack alphabet denotes *directions*, and stack values denote *positions* in the tree, *i.e. nodes* in (the infinite version of) Figure 6. *Pushes* and *pops* correspond to moving *up* and *down* the tree. Such an automata model is known as a *Tree Stack Automaton (TSA)* [7] – here, we introduce *tree-generating TSA*.

▶ **Definition 6.** *A tree-generating TSA $\mathcal{A}$ is a tuple $\langle \Sigma, Q, \Gamma, \mathcal{M}, \delta, q_0, \gamma_0, m_0 \rangle$ where Σ is a ranked alphabet of* terminals*, Q is a set of* states*, Γ is a finite* stack alphabet*, $\mathcal{M}$ is a finite* memory alphabet*, $q_0 \in Q$ is the* starting state*, $\gamma_0 \notin \Gamma$ is the* bottom-of-stack marker *and $m_0 \in \mathcal{M}$ is the* initial local memory*. Letting $\Gamma^\bullet = \Gamma \uplus \{\gamma_0\}$, the transition function δ has type:*

$$\delta : Q \times \mathcal{M} \times \Gamma^\bullet \rightharpoonup Q + \{\mathsf{b}(q_1, \dots, q_{|\mathsf{b}|}) \mid q_i \in Q, \mathsf{b} \in \Sigma\} + Q \times \mathcal{M} \times (\{up_\gamma \mid \gamma \in \Gamma\} + \{down\}) .$$

Informally, the transitions operate as follows. Initially, only γ_0 is on the stack. Subsequently, given state q, local memory m, and top of the stack $\gamma \in \Gamma^\bullet$:

1. if $\delta(q, m, \gamma) = q'$, the automaton changes state to q', leaving the stack and local memory unchanged;
2. if $\delta(q, m, \gamma) = \mathsf{b}(q_1, \dots, q_{|\mathsf{b}|})$, it outputs $\mathsf{b} \in \Sigma$ and branches – to explore the ith child ($1 \leq i \leq |b|$) it proceeds to state q_i leaving other components unchanged;
3. if $\delta(q, m, \gamma) = (q', m', up_{\gamma'})$, it updates the local memory to m', changes state to q' and pushes γ' onto the stack / moves up in direction γ' (if this is the first visit to that node, its local memory is set to m_0);
4. if $\delta(q, m, \gamma) = (q', m', down)$, it updates the local memory to m' and the state to q', and then pops / moves down (we adopt the convention that γ_0 cannot be popped so, if $\gamma = \gamma_0$ in this case, the automaton blocks).

Running a TSA $\mathcal{A}$ produces a possibly infinite tree $\mathsf{Tree}(\mathcal{A})$.

In the degenerate case where $\mathcal{M} = \{m_0\}$, tree-generating TSAs turn out to be precisely tree-generating deterministic pushdown automata (PDA): the local memory cannot store information, so only the stack remains. In general, however, it is not hard to see that TSAs are Turing-complete; fortunately we will only need TSAs satisfying a further condition called *restriction* [7]. A tree-generating TSA is k**-restricted** if every node can be accessed from below at most k times. It is **restricted** if it is k-restricted for some $k \in \mathbb{N}$.

We implement the evaluation of a MAHORS $\mathcal{G}$ with a restricted TSA $\mathcal{A}(\mathcal{G})$ with states

$$\mathcal{Q} = (\sum_{F \in \mathcal{N}} M^O_{\bigotimes_{1 \leq i \leq N} \bigotimes_{G \in \mathcal{N}} [\![\mathcal{N}(G)]\!] \multimap [\![\mathcal{N}(F)]\!]}) + (\sum_{F \in \mathcal{N}} \mathcal{M}_+^{\langle\!\langle \mathcal{R}'(F) \rangle\!\rangle}).$$

$$\begin{array}{rcll}
(\mathsf{Move}(F,a),(F,m),\_) & \mapsto & \mathsf{State}(F,\delta_-^F(m,a)) & \\
(\mathsf{State}(F,m),\_,\_) & \mapsto & \mathsf{State}(F,m') & \text{if } \delta_+^F(m) = m' \\
(\mathsf{State}(F,m),\_,\_) & \mapsto & \mathsf{b}(\mathsf{State}(F,m_1),\dots,\mathsf{State}(F,m_{|\mathsf{b}|})) & \text{if } \delta_+^F(m) = \mathsf{b}(m_1,\dots,m_{|\mathsf{b}|}) \\
(\mathsf{State}(F,m),\_,\_) & \mapsto & (\mathsf{Move}(G,(2,a)),(F,m'),up_{(F,i)}) & \text{if } \delta_+^F(m) = (m',(1,i,G,a)) \text{ with } a \in M^O_{\mathcal{N}(G)} \\
(\mathsf{State}(G,m),\_,(F,i)) & \mapsto & (\mathsf{Move}(F,(1,i,G,a)),(G,m'),down) & \text{if } \delta_+^G(m) = (m',(2,a)) \text{ with } a \in M^P_{\mathcal{N}(G)}
\end{array}$$

Figure 7 Transition function for the GoI TSA.

We use constructors Move and State to refer to elements from the left and right components of $\mathcal{Q}$ respectively. The *memory alphabet* is $\mathcal{M} = \sum_{F\in\mathcal{N}} \mathcal{M}_-^{\langle\mathcal{R}'(F)\rangle} / \equiv$, where $\equiv$ is the smallest equivalence relation with $(F,m_0) \equiv (G,m_0)$ for all $F,G \in \mathcal{N}$. We write m_0 for this equivalence class, providing the *initial memory state*. The *stack alphabet* is $\Gamma = N \times \mathcal{N}$ where N is the smallest integer such that all non-terminals have fewer than N occurrences in $\mathcal{R}(F)$, for all $F \in \mathcal{N}$. The start state is $q_0 = \mathsf{Move}(S,\circ)$ and the transition function is given in Figure 7.

The TSA $\mathcal{A}(\mathcal{G})$ is designed so that a run of stack size bounded by n simulates a run of $\mathcal{T}_S^n$. When in state $\mathsf{State}(F,m)$, the automaton is currently operating in a F node of $\mathcal{T}_S^n$ (as in Figure 6), performing internal computation following δ_+^F. If this internal computation produces a move, this move will be addressed either up or down the stack, depending of whether it is a Player move in $\mathcal{N}(F)$ (in which case we must move down), or an Opponent move in $\otimes_{1\le i\le N} \otimes_{G\in\mathcal{N}} [\![\mathcal{N}(G)]\!]$ (in which case we must move up, passing the control to a recursive call). If the state is $\mathsf{State}(G,m)$ and the top of the stack is (F,i), that means that we are currently running non-terminal G, which was called as the i-th occurrence of G in F. So the stack, together with the non-terminal symbol in the state, indicate the address of a node in Figure 6. When moving up or down the stack, we first change to a transient state $\mathsf{Move}(F,a)$ in which the automaton reads the input move using δ_-^F and resumes as above.

▶ **Theorem 7.** *For any MAHORS $\mathcal{G}$, there exists a restricted TSA $\mathcal{A}(\mathcal{G})$ (constructed in polynomial time) such that* $\mathsf{Tree}(\mathcal{A}(\mathcal{G})) = \mathrm{BT}(\mathcal{G})$.

Proof. For $n \ge 1$, write $\mathsf{Tree}_n(\mathcal{A}(\mathcal{G}))$ for the tree obtained from the truncated run-tree where the stack size is bounded by $n-1$ (where γ_0 has size 0). By construction, this truncated run-tree is weakly bisimilar to that of $\mathcal{T}_S^n$. In particular, $\mathsf{Tree}_n(\mathcal{A}(\mathcal{G})) = \mathsf{Tree}(\mathcal{T}_S^n) = \mathrm{BT}(\mathsf{unf}_n(S))$ by Proposition 5, so $\mathsf{Tree}(\mathcal{A}(\mathcal{G})) = \mathrm{BT}(\mathcal{G})$ by continuity.

This TSA is restricted: for any type φ, there is a bound on the length of plays in $P_{[\![\varphi]\!]}$ – in fact $M_{[\![\varphi]\!]}$ is finite, and plays in $P_{[\![\varphi]\!]}$ cannot use the same move twice. Let k be an upper bound to the maximal length of a play in $P_{\otimes_{G\in\mathcal{N}}[\![\mathcal{N}(G)]\!]}$. Then, $\mathcal{A}(\mathcal{G})$ is k-restricted. Indeed, fix a stack value $\gamma_{n+1}\gamma_n\dots\gamma_0$ with $\gamma_{n+1} = (F,i)$. Then, all transitions moving between $\gamma_{n+1}\dots\gamma_0$ and $\gamma_n\dots\gamma_0$ carry a move from $M_{\otimes_{G\in\mathcal{N}}[\![\mathcal{N}(G)]\!]}$. By construction, the sequence of such moves forms a play in $P_{\otimes_{G\in\mathcal{N}}[\![\mathcal{N}(G)]\!]}$. Hence, it is bounded by k. ◀

If the input scheme is an MHORS then each $\mathcal{R}'(F)$ is interpreted by a history-free strategy: $\mathcal{M}_-^{\langle\mathcal{R}'(F)\rangle}$ is a singleton. Consequently, $\mathcal{A}(\mathcal{G})$ has trivial memory and is in fact simply a PDA. This PDA is still k-restricted but also satisfies a stronger *linearity* property:

▶ **Lemma 8.** *Let $\mathcal{G}$ be an MHORS. Then the tree-generating PDA $\mathcal{A}(\mathcal{G})$ is **linear**, in the sense that the associated graph of reachable configurations is a tree.*

Proof. A strategic transducer on A is **reversible** if for each $a \in M_A^P$ there is at most one $m \in \mathcal{M}_+$ such that $\delta_+(m) = (\_,a)$ and for each $m \in \mathcal{M}_+$ there is at most one $(m',a) \in \mathcal{M}_- \times M_A^O$ such that $\delta_-(m',a) = m$ or at most one $m' \in \mathcal{M}_+$ such that $\delta(m') = m$, and the two possibilities are mutually exclusive. Reversible strategic transducers are closed under all operations used in the interpretation, hence if $\Sigma \mid \Delta \vdash t : A$ involves no product, $\langle t\rangle$ is *reversible* (this phenomenon is well-known in GoI [6]). This entails that $\mathcal{A}(\mathcal{G})$ is linear. ◀

5 TSA to MAHORS

In this section we show how to simulate a k-restricted TSA $\mathcal{A} = \langle \Sigma, Q, \Gamma, \mathcal{M}, \delta, q_0, \gamma_0, m_0 \rangle$ in MAHORS, i.e. we establish the converse of Theorem 7.

Let $B = |\Gamma|$ and $\Gamma = \{\gamma_1, \cdots, \gamma_B\}$. Nodes of the tree store will be represented using non-terminals $F^{u_1,\cdots,u_B;d}_{q,m,\gamma}$, where $(q,m,\gamma) \in Q \times \mathcal{M} \times \Gamma^{\bullet}$ represent the current state, node label and top of the stack respectively, d $(1 \le d \le k+1)$ is the number of times the node has already been visited from below and each u_j $(0 \le u_j \le k,\ 1 \le j \le B)$ is the number of times that the jth child has been visited from below. For brevity, we will write $\vec{u}$ instead of $u_1, \cdots, u_B$.

For $1 \le d \le k$, $F^{\vec{u};d}_{q,m,\gamma}$ has $B+1$ arguments: the first B arguments are used to simulate up_{γ_j} $(1 \le j \le B)$ and the last one corresponds to *down*. Each of the arguments is a Q-indexed tuple of continuations, so that projection can be used to select the right component to model the associated state change. When moving up the tree (up_{γ_j}), we call the jth argument passing as an argument another continuation that makes it possible to return (move down) later. Dually, when moving down the tree, we call the last argument passing as an argument a continuation that represents a further visit up. Using these ideas, one could code unrestricted TSA in an untyped setting, but we shall rely on carefully crafted types that allow, for each node, for up to k visits from below. In particular, if the automaton is moving down having visited a node k times from below, the corresponding upwards continuation for the $k+1$ visit is of type o, i.e. it is not usable for any future calls. The rules for $1 \le d \le k$ are summarised in the table below, using λ notation for brevity (for $F^{\vec{u};k+1}_{q,m,\gamma}$ we set $F^{\vec{u};k+1}_{q,m,\gamma} x_1 \cdots x_B = \bot$).

$\delta(q,m,\gamma)$	rule
q'	$F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y = F^{\vec{u};d}_{q',m,\gamma}\, x_1 \cdots x_B y$
$\mathsf{b}(q_1, \cdots, q_{\lvert\mathsf{b}\rvert})$	$F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y = \mathsf{b}\, \langle F^{\vec{u};d}_{q_1,m,\gamma}\, x_1 \cdots x_B y, \cdots, F^{\vec{u};d}_{q_{\lvert\mathsf{b}\rvert},m,\gamma}\, x_1 \cdots x_B y \rangle$
$(q', m', down)$	$F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y = (\pi_{q'} y)\, \langle F^{\vec{u};d+1}_{q'',\underline{m'},\gamma}\, x_1 \cdots x_B \mid q'' \in Q \rangle$
(q', m', up_{γ_j})	$F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y = (\pi_{q'} x_j)\, \langle \lambda z^{\overline{T_j}}. F^{\vec{u}+e_j;d}_{q'',m',\gamma}\, x_1 \cdots x_{j-1} z x_{j+1} \cdots x_B y \mid q'' \in Q \rangle$

In the *down* case, note that the q'th component of y is used to model state change and that the continuation features m' instead of m to reflect the local memory update. Note also the change from d to $d+1$, which updates the count of visits from below.

In the *up* case, the q'th component of x_j is used to model state change and the direction of the upward move (γ_j). The use of the same γ on both sides captures the same position on the stack and m' is used on the rhs to simulate the local memory update. d does *not* change, because the continuation represents revisiting the node from above (rather than from below). However, once the node is revisited from above in the future, its jth child will have been visited $u_j + 1$ times from below: hence the change to u_j (we write $\vec{u} + e_j$ for $\vec{u}$ with the jth component incremented by 1). In the *up* case, we use a λ-term inside a rule to highlight the intention more clearly, this can be avoided by using an auxiliary non-terminal.

The start symbol $S : o$ has rule $S = (F^{0,\cdots,0;1}_{q_0,m_0,\gamma_0})\, N_1 \cdots N_B \langle \bot \mid q \in Q \rangle$. The divergent terms correspond to our convention that the automaton blocks when *down* is called at the root node. N_j $(1 \le j \le B)$ stands for $\langle N_{q,j} \mid q \in Q \rangle$, where $N_{q,j}$ are auxiliary non-terminals that represent nodes visited for the first time. They are subject to the rule $N_{q,j} y = F^{0,\cdots,0;1}_{q,m_0,\gamma_j}\, N_1 \cdots N_B y$.

The scheme depends on types of the form T_i $(0 \le i \le k)$ defined by $T_k = o$ and $T_i = \overline{(\overline{T_{i+1}} \multimap o)} \multimap o$, where $\overline{T}$ stands for $\&_{q \in Q} T$, i.e. $|Q|$ copies of T. In particular, we have $F^{\vec{u};d}_{q,m,\gamma} : \overline{T_{u_1}} \multimap \cdots \multimap \overline{T_{u_B}} \multimap T_{d-1}$ and $N_{q,j} : T_0$.

▶ **Theorem 9.** *For any restricted TSA, there exists an equivalent MAHORS (constructible in exponential time).*

In conjunction with Theorem 7, this shows that MAHORS and restricted TSA are equivalent.

6 Expressivity of MAHORS

It is easy to see that any (classic) first-order recursion scheme (1-HORS) can be viewed as a MAHORS, simply by giving the terminals types of the form $o \,\&\, \cdots \,\&\, o \multimap o$. Hence, MAHORS are at least as expressive as first-order HORS. Next, informed by results from the preceding sections, we can discuss their relationship with schemes of higher orders. Because our TSA model is a tree-generating variant of the automata from [7], which capture multiple context-free languages [21], we can immediately conclude the following.

▶ **Lemma 10.** *The branch language of a tree generated by a MAHORS is multiple context-free.*

Thanks to the Lemma, we can show that MAHORS and second-order HORS are incomparable.

▶ **Example 11.** There exists a second-order HORS, which is not equivalent to any MAHORS. For example, consider the 2-HORS given by: $S = Fb$, $Ff = a(f\$)(F(Gf))$, $Gfx = f(fx)$, where $a : o \to o \to o$, $b : o \to o$ and $\$: o$ are terminals and $F : (o \to o) \to o$ and $G : (o \to o) \to o \to o$ are non-terminals. The scheme generates an infinite tree whose finite branches correspond to the language $L = \{a^n b^{2^{n-1}} \$ \,|\, n \geq 1\}$. Because it is known that L is not multiple context-free [21, Lemma 3.5], it cannot be the branch language of a MAHORS by Lemma 10.

▶ **Example 12.** We give a MAHORS that is not equivalent to any second-order HORS, exploiting the fact that the language $L = \{w\#w\#w \,|\, w \in D\}$, where D is the Dyck language ($D = \epsilon \,|\, [D]D$), is not indexed [8] (see also page 2 of [16]). The MAHORS given below (using λ-syntax for brevity) has been obtained by lifting the grammar rules for D to triples of words, encoded with the type $\mathsf{T}_3 = ((o \multimap o) \multimap (o \multimap o) \multimap (o \multimap o) \multimap o) \multimap o$. Consequently, it generates a tree whose finite branches are the words of L prefixed by a segment of b's and followed by \$. The terminal $\mathsf{b} : (o \,\&\, o) \multimap o$ represents rule choice and the other terminals $([,], \# : o \multimap o,\ \$: o)$ are used to build the word. The scheme relies on the following non-terminals: $S : o$, $D : \mathsf{T}_3$, $K : (o \multimap o) \multimap (o \multimap o) \multimap (o \multimap o)$ and $I : o \multimap o$, which are subject to the following rules:

$$\begin{aligned}&S = D(\lambda xyz.x(\#(y(\#(z\$))))), \qquad Kxyv = [(x(](yv))), \qquad Iv = v,\\ &Df = \mathsf{b}\langle fIII, D(\lambda x_1y_1z_1.D(\lambda x_2y_2z_2.f(Kx_1x_2)(Ky_1y_2)(Kz_1z_2)))\rangle\end{aligned}$$

If the scheme were equivalent to a 2-HORS, the language of its branches would be accepted by a 2-CPDA [12], i.e. it would be indexed [2]. However, indexed languages are closed under homomorphism, so L would be indexed too, because erasing b's and \$ is a homomorphism.

Lemma 10 identifies a strong restriction on branch languages of trees generated by MAHORS. Since multiple context-free languages form a subset of third-order collapsible pushdown languages [20], it is natural to ask whether every MAHORS might be equivalent to a third-order HORS. One could try to establish this, for example, by showing that, for every restricted TSA, there is an equivalent MAHORS that uses third-order types. Unfortunately, our proof of Theorem 7 uses types whose order grows linearly in the restriction parameter k. At the time of writing, we believe this necessary to capture the complexity of run-trees generated by our (infinite-)tree-generating TSA, though we are aware that similar hierarchies for (finite-)word languages and (finite-)tree languages do collapse, e.g. second-order abstract categorial grammars [19, 15]. The main difficulty that prevents us from translating TSA into MAHORS of order 3 is that there may be infinitely many (sub)runs that start from a given node, visit only nodes above it and return to the same node, and all such runs have to be captured in a single MAHORS. In contrast, for word languages, when TSA are seen as acceptors of finite words, it suffices to focus on the representation of a single run [7].

7 Multiplicative HORS (MHORS)

In this section we consider MHORS, i.e. &-free MAHORS. Recall from Lemma 8 that, for any MHORS, there exists an equivalent *linear* PDA (LPDA) $\langle \Sigma, Q, \Gamma, \delta, q_0, \gamma_0\rangle$ with transition function $\delta : Q \times \Gamma^\bullet \rightharpoonup Q + \{\mathsf{b}(q_1, \dots, q_{|\mathsf{b}|}) \mid q_i \in Q,\, \mathsf{b} \in \Sigma\} + Q \times (\{up_\gamma \mid \gamma \in \Gamma\} + \{down\})$ such that any reachable configuration must be reachable through a unique path. Next we prove the converse using first-order MHORS only. In combination with Lemma 8, this amounts to a polynomial-time translation from arbitrary MHORS to first-order MHORS.

In what follows, we view an LPDA as a pushdown system with a successor relation $\Rightarrow$, in order to exploit standard reachability techniques [4, 9]. We work with configurations of the form $(q,t) \in Q \times (\Gamma^\bullet)^*$. As we do not have the space to review all the necessary definitions, let us just recall that the techniques employ *multi-automata* over $\Gamma^\bullet$ to recognise sets of configurations. Multi-automata are finite-state machines with multiple initial states, one for each state of the analysed pushdown system. Let i_q be the initial state of a multi-automaton corresponding to $q \in Q$. Then a multi-automaton is said to recognise (q,t) if it accepts t once started from i_q (this corresponds to processing stack content top-down). In particular, we take advantage of the following facts.

- For any LPDA $\mathcal{A}$, there exists a multi-automaton $\mathcal{A}_{era}$, constructible in polynomial time, which captures erasable stack content, i.e. $\{(q,t) \in Q \times \Gamma^* \mid \exists q' \in Q.\, (q,t) \Rightarrow^* (q', \epsilon)\}$. Using terminology from [4], this corresponds to $pre^*(Q \times \{\epsilon\})$. Hence, given $\mathcal{A}$, one can calculate the relation $R_{\mathcal{A}} = \{(q,\gamma,q') \in Q \times \Gamma \times Q \mid (q,\gamma) \Rightarrow^* (q',\epsilon)\}$ in polynomial time.
- For any LPDA $\mathcal{A}$, there exists a multi-automaton $\mathcal{A}_{rea}$, constructible in polynomial time, which represents all configurations reachable from (q_0,γ_0), i.e. all $(q, t\gamma_0)$ such that $(q_0,\gamma_0) \Rightarrow^* (q, t\gamma_0)$. This corresponds to representing $post^*(\{(q_0,\gamma_0)\})$ [9].

▶ **Lemma 13.** *For any LPDA $\mathcal{A}$, there exists an equivalent MHORS (of order 1) and its construction can be carried out in polynomial time.*

Proof. The translation is similar to the PDA-to-1-HORS translation in [12] except that reachability analysis ($R_{\mathcal{A}}$) is used to identify places where variables actually get used. This is needed to produce a term that is linearly typable. ◀

Consequently, LPDA and MHORS are equivalent. We end this section by showing they generate regular trees. Our first lemma states that, if the stack of an LPDA grows sufficiently, there is a point after which elements lying below a certain level will no longer be accessible.

▶ **Lemma 14.** *Let $s \in \Gamma^*$. There exists a bound $H_s \ge 0$ such that, for any $t \in \Gamma^*$, if $(q, ts\gamma_0)$ is reachable and $|t| > H_s$ then there is no q' such that $(q,t) \Rightarrow^* (q',\epsilon)$.*

Proof. Consider $X = \{(q, s\gamma_0) \mid (q_0,\gamma_0) \Rightarrow^* (q, s\gamma_0)\}$. Observe that $0 \le |X| \le |Q|$. Because we work with an LPDA, there can be at most $|Q|$ runs from (q_0,γ_0) to X. Let H_s be the maximum stack height occurring in these runs (take 0 if $X = \varnothing$). Suppose $(q, ts\gamma_0)$ is reachable and $|t| > H_s$. If we had $(q,t) \Rightarrow^* (q',\epsilon)$ for some q' then there would be a run $(q_0,\gamma_0) \Rightarrow^* (q, ts\gamma_0) \Rightarrow^* (q', s\gamma_0)$ in which the stack height exceeds H_s (because it visits $(q, ts\gamma_0)$). This contradicts the choice of H_s. ◀

The above bound depends on s. We show that there is a uniform bound, polynomial with respect to the size of $\mathcal{A}$. First, given $s \in \Gamma^*$, the multi-automaton $\mathcal{A}_{rea}$ discussed earlier can be modified to represent $\{(q,t) \mid (q_0,\gamma_0) \Rightarrow^* (q, ts\gamma_0)\}$ simply by changing the accepting states of $\mathcal{A}_{rea}$ (to those from which an original accepting state is reachable via an $s\gamma_0$-labelled path). Let $\mathcal{A}^s_{rea}$ be the resultant automaton. Note that the size of $\mathcal{A}^s_{rea}$ is bounded by a polynomial in $|\mathcal{A}|$ that is independent of s, because the only difference between $\mathcal{A}^s_{rea}$ and $\mathcal{A}_{rea}$ is the set of accepting states, and its size bounded by $|Q|$.

Observe that $\{(q,t) \mid (q_0,\gamma_0) \Rightarrow^* (q, ts\gamma_0),\, (q,t) \Rightarrow^* (q',\epsilon) \text{ for some } q'\}$ is exactly the set of configurations that are represented by both $\mathcal{A}^s_{rea}$ and $\mathcal{A}_{era}$. Consider the product $\mathcal{A}'$ of the two multi-automata. By Lemma 14, $\mathcal{A}'$ cannot have reachable loops. Consequently, the longest word that it accepts from any initial state is bounded by the number of states of the automaton, which is polynomial in $|\mathcal{A}|$. As this reasoning is independent of s, we obtain:

▶ **Lemma 15.** *For any LPDA $\mathcal{A}$, there exists a bound H, polynomial in $|\mathcal{A}|$, such that, for any $s,t \in \Gamma^*$, if $(q, ts\gamma_0)$ is reachable and $|t| > H$ then there is no q' such that $(q,t) \Rightarrow^* (q',\epsilon)$.*

This implies that an LPDA can only use H top elements from its stack, i.e. its stack can be simulated by a finite state automaton, which is exponentially bigger. Because any 0-HORS is also an MHORS, MHORS and 0-HORS are equivalent, i.e. they generate exactly the regular trees. However, it is worth noting that MHORS may be more succinct.

▶ **Example 16.** The MHORS built from terminals $a, b : o \multimap o$, non-terminals $S : o, F_i : o \multimap o$ $(1 \le i \le n)$ with $S = F_n(bS)$, $F_0(x) = ax$ and $F_i(x) = F_{i-1}(F_{i-1}x)$ for $1 \le i \le n$ generates an infinite branch $(a^{2^n}b)^\omega$, which could only be generated by a 0-HORS of exponential size in n.

References

1 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full Abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000. doi:10.1006/inco.2000.2930.

2 Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. Safety Is not a Restriction at Level 2 for String Languages. In *Proceedings of FOSSACS*, volume 3441 of *Lecture Notes in Computer Science*, pages 490–504. Springer, 2005.

3 Patrick Baillot. *Approches dynamiques en sémantique de la logique linéaire: jeux et géométrie de l'interaction.* PhD thesis, Aix-Marseille 2, 1999.

4 Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *Proceedings of CONCUR*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

5 Pierre Clairambault, Charles Grellois, and Andrzej S. Murawski. Linearity in higher-order recursion schemes. *PACMPL*, 2(POPL):39:1–39:29, 2018.

6 Vincent Danos and Laurent Regnier. Reversible, Irreversible and Optimal lambda-Machines. *Theor. Comput. Sci.*, 227(1-2):79–97, 1999.

7 Tobias Denkinger. An Automata Characterisation for Multiple Context-Free Languages. In *Proceedings of DLT*, volume 9840 of *Lecture Notes in Computer Science*, pages 138–150. Springer, 2016.

8 Joost Engelfriet and Sven Skyum. Copying Theorems. *Inf. Process. Lett.*, 4(6):157–161, 1976.

9 Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient Algorithms for Model Checking Pushdown Systems. In *Proceedings of CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2000.

10 Jean-Yves Girard. Linear Logic. *Theoretical Computer Science*, 50:1–102, 1987.

11 Jean-Yves Girard. Geometry of Interaction 1: Interpretation of System F. *Studies in Logic and the Foundations of Mathematics*, 127:221–260, 1989.

12 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible Pushdown Automata and Recursion Schemes. *ACM Trans. Comput. Log.*, 18(3):25:1–25:42, 2017.

13 Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 52:1–52:10, 2014. doi:10.1145/2603088.2603124.

14 Martin Hyland. Game semantics. *Semantics and logics of computation*, 14:131, 1997.

15 Makoto Kanazawa. Second-Order Abstract Categorial Grammars as Hyperedge Replacement Grammars. *Journal of Logic, Language and Information*, 19(2):137–161, 2010.

16 Makoto Kanazawa and Sylvain Salvati. The Copying Power of Well-Nested Multiple Context-Free Grammars. In *Proceedings of LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 344–355. Springer, 2010.

17 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of POPL*, pages 416–428, 2009.

18 Olivier Laurent. A Token Machine for Full Geometry of Interaction. In *Proceedings of TLCA*, pages 283–297, 2001. doi:10.1007/3-540-45413-6_23.

19 Sylvain Salvati. Encoding second order string ACG with deterministic tree walking transducers. In *Proceedings of FG*, pages 143–156. CSLI Publications, 2006.

20 Sylvain Salvati. MIX is a 2-MCFL and the word problem in Z^2 is captured by the IO and the OI hierarchies. *J. Comput. Syst. Sci.*, 81(7):1252–1277, 2015.

21 Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On Multiple Context-Free Grammars. *Theor. Comput. Sci.*, 88(2):191–229, 1991.

Uniform Random Expressions Lack Expressivity

Florent Koechlin
Université Paris-Est, LIGM (UMR 8049), CNRS, ENPC, ESIEE, UPEM, France
florent.koechlin@u-pem.fr

Cyril Nicaud
Université Paris-Est, LIGM (UMR 8049), CNRS, ENPC, ESIEE, UPEM, France
cyril.nicaud@u-pem.fr

Pablo Rotondo
Université Paris-Est, LIGM (UMR 8049), CNRS, ENPC, ESIEE, UPEM, France
pablo.rotondo@u-pem.fr

Abstract

In this article, we question the relevance of uniform random models for algorithms that use expressions as inputs. Using a general framework to describe expressions, we prove that if there is a subexpression that is absorbing for a given operator, then, after repeatedly applying the induced simplification to a uniform random expression of size n, we obtain an equivalent expression of constant expected size. This proves that uniform random expressions lack expressivity, as soon as there is an absorbing pattern. For instance, $(a+b)^\star$ is absorbing for the union for regular expressions on $\{a,b\}$, hence random regular expressions can be drastically reduced using the induced simplification.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases Random expressions, simplification algorithms, analytic combinatorics

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.51

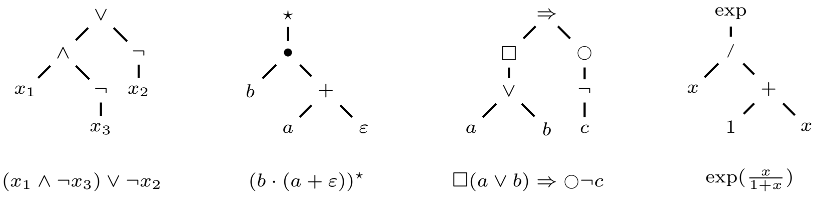
1 Introduction

Although the classical paradigm to analyze the efficiency of an algorithm is to study its worst case complexity, in many situations it is not a relevant way to describe what is observed in practice, when one actually implements and tests the algorithm. This is typically the case when worst case instances are very unlikely, and when the algorithm behaves well on most inputs. A natural way to deal with this issue is to set a probability distribution on the inputs of size n, for each n, and to study the average running time of the algorithm when the inputs are randomly chosen following the prescribed distribution.[1] The textbook example is QuickSort algorithm with a simple choice of pivot, which runs in $\mathcal{O}(n \log n)$ expected time and $\mathcal{O}(n^2)$ in the worst case. But the average case analysis raises the question of the input distribution, which is usually a difficult one. The result on QuickSort holds when the input is a permutation taken uniformly at random, but are real-life arrays uniformly shuffled? Probably not, whatever "real-life" means. For QuickSort, one can randomize the choice of the pivot to ensure the $\mathcal{O}(n \log n)$ expected running time, which is equivalent to uniformly shuffling the input. This is probably not a good idea in practice though, since we lose the property that some input can be often partially sorted: popular programming languages as Python or Java now use the recent TimSort algorithm that takes advantage of the presortedness of the input.

[1] There are other relevant theoretical alternatives, as the *smoothed analysis* of algorithms, but we will only consider average case analysis in this article.

Still, if one wants to test the efficiency of an implementation of an algorithm, or to compare two tools, and if there are not sufficiently many benchmarks available, doing the analysis on random inputs sounds reasonable. In addition, using the uniform distribution has several advantages: it ensures a lot of diversity on the generated input (as they are all equally likely) and there are several very generic methods to perform the random generation efficiently: the two main ones being the *recursive method* [20] and *Boltzmann samplers* [7]; both directly translate a combinatorial description of the inputs into a uniform random generator. And also, beyond the experiments, the theoretical average case analysis of the algorithm can be done using techniques from discrete probabilities or from analytic combinatorics.

In this article we focus on algorithms whose inputs are formulas, encoded by expressions. Informally, an expression is a tree, whose internal nodes are labelled with operators, and whose leaves are labelled with variables or constants. Such trees are very natural ways to represent formulas in many different contexts; several examples are given in Fig. 1. Observe that the equivalent reverse polish notation of the expression can be obtained by a simple tree traversal.



$(x_1 \wedge \neg x_3) \vee \neg x_2$ $(b \cdot (a+\varepsilon))^\star$ $\square(a \vee b) \Rightarrow \bigcirc\neg c$ $\exp(\frac{x}{1+x})$

Figure 1 Four expression trees and their associated formulas. From left to right: a logical formula, a regular expression, an LTL formula and a function.

We define the set of expressions by the finite sets of operators of any given degree: for regular expressions on an alphabet $\{a, b\}$, we consider that there are three kinds of leaves $\{a, b, \varepsilon\}$, one unary internal node $\{\star\}$ for the Kleene star and two binary internal nodes $\{\bullet, +\}$ for the concatenation and the union. We define the *size* of an expression as its number of nodes (including leaves). For the set of regular expressions $\mathcal{L}_R$, we have the following formal inductive description:

$$\mathcal{L}_R = a + b + \varepsilon + \overset{\star}{\underset{\mathcal{L}_R}{|}} + \overset{\bullet}{\underset{\mathcal{L}_R\ \mathcal{L}_R}{/\backslash}} + \overset{+}{\underset{\mathcal{L}_R\ \mathcal{L}_R}{/\backslash}}\,. \tag{1}$$

Observe that if we consider plane trees (where the children of a node are ordered, as when they are drawn on a plan), Eq. (1) is an unambiguous description of the expressions in $\mathcal{L}_R$. It is, of course, not an unambiguous description of the associated languages, since a given language admits several representations using expressions. It is clear that the other expressions of Fig. 1 also have a specification that is similar to Eq. (1). Efficient methods were developed in the field of analytic combinatorics to deal with that kind of combinatorial inductive descriptions. In a nutshell, we first associate the formal power series $L(z) := \sum_{n \geq 0} \ell_n z^n$, where ℓ_n is the number of expressions of size n. Eq. (1) translates into an equation on $L(z)$:

$$L(z) = z\phi(L(z)), \text{ with } \phi(X) = 3 + X + 2X^2. \tag{2}$$

Then we see $L(z)$ as a function from $\mathbb{C}$ to $\mathbb{C}$, which is analytic at 0, and use its functional equation Eq. (2) to derive a lot of information on ℓ_n, typically asymptotic equivalents. When considering the uniform distribution, one can easily go further and introduce formal parameters in the specification (for instance, the number of unary operators), which can also be interpreted in the analytic world, and directly obtain statistics of interest (for instance, the asymptotic equivalent of the expected number of unary operators). Combinatorial structures whose generating series satisfy an implicit functional equation $L(z) = z\phi(L(z))$ are called *simple varieties of trees* (there are some analytic conditions on ϕ, which are always satisfied in the sequel), and we refer the reader to the textbook on analytic combinatorics [9, VII.3.] for further information. For the remainder of this introduction, the reader has just to keep in mind that the technical framework used to deal with uniform random expressions is well established.

As can be expected at this point of the discussion, there are a lot of results in the literature on simple varieties of trees and on specific random expressions. They exhibit some kind of "universal" behavior: for instance, the number of such trees always grows in $\ell_n \sim \alpha\beta^n n^{-3/2}$, for some computable constants α and β [16]. Their expected height is asymptotically equivalent to $\gamma\sqrt{n}$, for some computable γ [8]. In [10], the authors proved that if a uniform random expression is compressed by identifying common subexpressions (changing the tree into a directed acyclic graph), the expected size of the result is asymptotically equivalent to $\frac{\delta n}{\sqrt{\log n}}$. It was also established [11] that computing the derivative of a random function, defined by a tree as in Fig. 1, costs $\Theta(n^{3/2})$ in expected time and space. Concerning regular expressions, the second author studied a standard construction that builds an ε-free non-deterministic automaton recognizing the same language [18]: he proved that there is an expected linear number of transitions in the resulting automaton, although it is $\Theta(n^2)$ in the worst case. This result was subsequently refined and adapted to various other similar constructions by Moreira, Reis and their co-authors (see [2] for a survey). A lot of efforts were also made for Boolean formulas, in slightly different settings: we consider the set $\mathcal{F}_k$ of the 2^{2^k} Boolean formulas on k variables. For a suitable set of logical operators, for instance $\{\vee, \wedge\}$, and for any $n \geq 1$, we consider the probability $p_n(f)$ that a random Boolean expression represents the Boolean function $f \in \mathcal{F}_k$; then we let n tend to infinity, and it can be proved that $p_n(f)$ has a limit $p(f)$, and that p is a probability distribution on $\mathcal{F}_k$. So the distribution is different, and more theoretical than what is usually used for testing implementations. But, since it relies on uniform random expressions, that kind of distributions has a lot of similarities with uniform random expressions (see [12] for a survey). While investigating this distribution on Boolean formulas, its lack of complexity was discovered: with high probability the formulas obtained are very simple [13, 14]. In fact, there is an older result in Nguyên Thê PhD's dissertation [17, Ch 4.4], where he considered and/or Boolean expressions whose leaves are either `true`, `false` or a literal; he studied the impact of inductively applying 8 different reduction rules, such as $\texttt{true} \vee \phi \rightarrow \texttt{true}$, on uniform random expressions of size n, and proved that the expected size of the simplified expression tends to a constant as n tends to infinity. This was a very negative result for uniformly chosen random Boolean expressions (with $\{\vee, \wedge\}$ operators and $\{\texttt{true}, \texttt{false}, x\}$ leaves), as most of them can be drastically simplified using simple rules.

In this article, we aim at extending Nguyên Thê's result to much more general families of expressions, proving its universality in the context of simple varieties of trees. Informally, our main result is the following:

▶ **Theorem.** *Consider a simple variety of expressions, where there is a tree pattern $\mathcal{P}$ that is an absorbing element for one of the operators $\circledast$. We consider the simplification algorithm that consists in inductively changing a $\circledast$-node by $\mathcal{P}$ whenever one of its children can be simplified into $\mathcal{P}$. Then the expected size of the simplification of a uniform random expression of size n tends to a constant as n tends to infinity.*

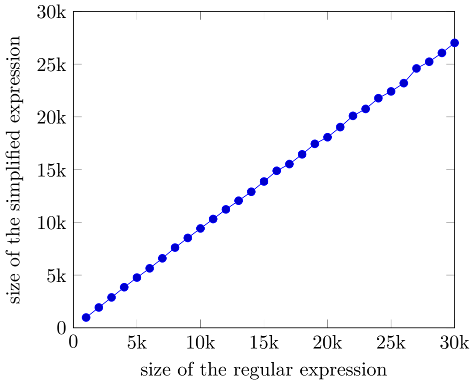


Figure 2 Experimental study of the size of a random uniform regular expression after applying the simplification rules $E + \mathcal{P} \rightsquigarrow \mathcal{P}$ and $\mathcal{P} + E \rightsquigarrow \mathcal{P}$.

As an example, the tree pattern $\mathcal{P}$ corresponding to $(a + b)^\star$ is absorbing for the union $+$ of regular expressions: for any regular expression E, one can simplify $E + \mathcal{P}$ or $\mathcal{P} + E$ into $\mathcal{P}$ without changing the described language. Hence, applying this simple simplification rule to a very large random regular expression results in an expression of constant expected size. Also, the tree pattern associated with $x_1 \vee \neg x_1$ is absorbing for $\vee$, since it evaluates to `true`: random Boolean expressions are also drastically simplified using this transformation rule. It is the same for LTL formulas, and for functions: if there is the constant 0 (or a way to produce it with a tree pattern) and the multiplication, then the same phenomenon occurs. In fact, for most natural notions of expressions, there are absorbing patterns for some operators, and uniform random expressions lack expressivity, since they can be simplified into much smaller equivalent expressions with high probability. This is a universal and negative result on uniform random expressions, which is worth knowing when designing empirical tests. Indeed, comparing two tools on uniform random expressions probably consists in simply comparing their efficiency for simplifying the input (preprocessing step). It also questions the relevance of further theoretical studies on that kind of models. The consequences are discussed in details in Section 4.

To conclude this introduction, we would like to mention that our result is not always easy to observe experimentally. We implemented uniform random samplers for regular expressions, and computed (experimentally) the expected size when using the fact that $(a + b)^\star$ is absorbing for $+$. The results are depicted on Fig. 2. It is not clear that the curve converges to some constant, as stated by our theorem. In fact, if we compute the value of the limit (we used SageMath software for the computations) we obtain a bit more than 3 624 217, a huge constant. This value can be considerably reduced if we introduce several more rewriting rules (see Section 4 for the details).

2 Definitions and settings

2.1 Combinatorial constructions and analytic combinatorics

In this article, the proofs rely on mathematical techniques developed in the framework of analytic combinatorics [9]. In this section, we recall the basic tools of this theory that will be used in the sequel (more advanced results will be introduced when needed). A *combinatorial class* is a set $\mathcal{C}$ equipped with a notion of *size* $|\cdot|$ (a mapping from $\mathcal{C}$ to $\mathbb{Z}_{\geq 0}$) such that for any $n \in \mathbb{Z}_{\geq 0}$ there are finitely many elements of $\mathcal{C}$ of size n. Let $\mathcal{C}_n = \{C \in \mathcal{C} : |C| = n\}$

and let $c_n = |\mathcal{C}_n|$ be its cardinality. The *ordinary generating series* (OGS for short) of the combinatorial class $\mathcal{C}$ is the formal power series $C(z) := \sum_{C\in\mathcal{C}} z^{|C|} = \sum_{\geq 0} c_n z^n$. Let $[z^n]C(z) := c_n$ denote the n-th coefficient of $C(z)$.

Let $\mathcal{A}$ and $\mathcal{B}$ be two combinatorial classes, and let $A(z)$ and $B(z)$ be their OGS, respectively. If $\mathcal{A}$ and $\mathcal{B}$ are disjoint, then the OGS of their union is simply $A(z) + B(z)$. Moreover, if we define the size of a pair (α, β) in $\mathcal{A} \times \mathcal{B}$ as the sum of the sizes of α and β, then $\mathcal{A} \times \mathcal{B}$ is a combinatorial class with OGS $A(z)B(z)$, as a simple computation shows. There are many other set constructions on combinatorial classes that directly translate on OGS, but we will not need them in the sequel (see [9, Part A]).

Let us take an example that will be used throughout this article, the combinatorial set $\mathcal{L}_R$ of regular expressions on the alphabet $\{a, b\}$. Its elements are plane rooted trees, whose nodes (internal and leaves) are labelled: leaves are labelled with a, b or ε, unary nodes are labelled with $\star$, and binary nodes with $\bullet$ or $+$. Rooted means that there is a root (contrary to the graph-theory notion of tree), and plane means that the order of the children matters: $\overset{+}{\underset{a\ b}{/\backslash}}$ and $\overset{+}{\underset{b\ a}{/\backslash}}$ are different elements of $\mathcal{L}_R$. Encoding trees as tuples (for instance $(\bullet, \texttt{left}, \texttt{right})$ for a tree of root $\bullet$ and left and right children `left` and `right`), we have the identity:

$$\mathcal{L}_R = \{a, b, \varepsilon\} \cup \{\star\} \times \mathcal{L}_R \cup \{\bullet, +\} \times \mathcal{L}_R \times \mathcal{L}_R, \tag{3}$$

which we rewrite as in Eq. (1) for readability: $\mathcal{L}_R = a + b + \varepsilon + \overset{\star}{\underset{\mathcal{L}_R}{|}} + \overset{\bullet}{\underset{\mathcal{L}_R\ \mathcal{L}_R}{/\backslash}} + \overset{+}{\underset{\mathcal{L}_R\ \mathcal{L}_R}{/\backslash}}$. From this we directly obtain the following functional equation for the associated OGS $L_R(z)$:

$$L_R(z) = 3z + zL_R(z) + 2zL_R(z)^2 = z\phi(L_R(z)), \text{ with } \phi(y) = 3 + y + 2y^2.$$

Going further, one can also encode statistics on the elements of a combinatorial class into generating series. Let ξ be a mapping from a combinatorial class $\mathcal{C}$ to $\mathbb{Z}_{\geq 0}$. We are interested in the double sequence $c_{n,k}$ that counts the number of elements $C \in \mathcal{C}_n$ such that $\xi(C) = k$. The associated bivariate generating series (BGS for short) is by definition:[2]

$$C(z, u) := \sum_{C\in\mathcal{C}} z^{|C|} u^{\xi(C)} = \sum_{n,k\geq 0} c_{n,k} z^n u^k.$$

When the statistics ξ is *additive*, which means that it is defined on pairs by $\xi((\alpha, \beta)) = \xi(\alpha) + \xi(\beta)$, then we also have a generic dictionary to translate combinatorial constructions into BGS directly (see [9, Ch III]). This is for instance the case if ξ is the number of a-leaves in a regular expression. Moreover, formal differentiation with respect to u allows us to compute the expected value of ξ for the uniform distribution: by definition, the expectation of ξ for objects of size n is $\mathbb{E}_n[\xi] = \frac{1}{|\mathcal{C}_n|} \sum_{C\in\mathcal{C}_n} \xi(C)$. But observe that

$$\frac{d}{du} C(z, u) = \frac{d}{du} \sum_{C\in\mathcal{C}} z^{|C|} u^{\xi(C)} = \sum_{n\geq 0} \left(\sum_{C\in\mathcal{C}_n} \xi(C) u^{\xi(C)-1} \right) z^n.$$

Hence, if we extract the n-th coefficient[3] in z, and set $u = 1$, we have c_n times the expectation. We thus get the following formula to compute the expectation for the uniform distribution:

$$\mathbb{E}_n[\xi] = \frac{[z^n] \frac{d}{du} C(z, u) \Big|_{u=1}}{[z^n]C(z, 1)}. \tag{4}$$

[2] Observe that $C(z, 1) = C(z)$.

[3] Recall that it is exactly what the operator $[z^n]$ does.

Higher moments are obtained similarly, by differentiating several times with respect to u.

Let us continue our example on regular expressions over a binary alphabet by studying the statistics ξ associated with the number of unary operators $\star$ in an expression. The technique consists in *marking* the $\star$-nodes (we use a circle, symbolically) and in changing the marks into the formal variable u when translating the combinatorial specification:

$$\mathcal{L}_R = a + b + \varepsilon + \underset{\mathcal{L}_R}{\overset{\circledast}{|}} + \underset{\mathcal{L}_R\ \mathcal{L}_R}{\overset{\bullet}{/\backslash}} + \underset{\mathcal{L}_R\ \mathcal{L}_R}{\overset{+}{/\backslash}} \longrightarrow L_R(z,u) = 3z + zuL_R(z,u) + 2zL_R(z,u)^2.$$

After differentiating with respect to u and setting $u = 1$ we get:

$$\frac{d}{du}L_R(z,u)\Big|_{u=1} = \frac{zL_R(z)}{1 - z - 4zL_R(z)}. \tag{5}$$

At this point, the values of interests are hidden as coefficients of formal power series, and we need to extract or estimate them. This is where we use complex analysis: when their radius of convergence at 0 is not zero, the formal power series are seen as functions from $\mathbb{C}$ to $\mathbb{C}$. Using theorems developed in the field, this proves very useful to obtain an asymptotic equivalent of the n-th coefficient. We will not state formally the theorems here, because there are some analytic conditions on the series that must be satisfied and that are a bit too long for this extended abstract (we refer to the textbook [9] for all the details). What is needed to know is the following: since we are doing combinatorics, the coefficients in our series $C(z)$ are non-negative reals. Hence, Pringsheim's theorem applies and yields that, if finite, the radius of convergence ρ of $C(z)$ is a singularity. If there are no other singularity of modulus ρ then, together with other analytic conditions, the behavior of the function $C(z)$ as $z \to \rho$ determines the asymptotic equivalent of $[z^n]C(z)$ as $n \to \infty$. For instance, the *Transfer Theorem* [9, Ch VI.3] states that, under analytic conditions, if $C(z) \sim_{z\to\rho} \lambda(1 - z/\rho)^{-\alpha}$ with $\alpha \notin \{0, -1, -2, \ldots\}$, then $[z^n]C(z) \sim \lambda\rho^{-n}n^{\alpha-1}/\Gamma(\alpha)$, where Γ is Euler's gamma-function, the generalization of factorial.

Back to our example, as a polynomial equation of degree two, Eq. (2) can be solved in $L_R(z)$. Only one of the two solutions is combinatorially meaningful (the other one has negative coefficients). The result is amenable to the Transfer Theorem, with $\rho_R = \frac{2\sqrt{6}-1}{23}$ and an exponent $-\alpha = -1/2$ near ρ_R, yielding $[z^n]L_R(z) \sim \lambda\rho_R^{-n}n^{-3/2}$, for some computable $\lambda > 0$. Similarly, we can estimate the expected number of $\star$-nodes using Eq. (4) and applying the Transfer Theorem to Eq. (5). This routine exercise yields that the expected number of $\star$-nodes is asymptotically equivalent to γn, for some computable positive constant γ.

Observe that the latter result has a direct consequence on the algorithm applying the simplification pattern $(\mathcal{T}^\star)^\star \rightsquigarrow \mathcal{T}^\star$: since it cannot remove more nodes than there are stars in the expression, the expected size after simplification is in $\Theta(n)$. In fact, we can easily be more precise using our tools, as stated in the following proposition.

▶ **Proposition 1.** *Consider the algorithm that simplifies consecutive stars in a regular expression. For the uniform distribution on $\mathcal{L}_R$, the expected size after reduction is asymptotically equivalent to κn, with $\kappa = \frac{4}{529}(126 + \sqrt{6}) \approx 0.97$.*

Hence, this simplification rule is not powerful enough to demonstrate the lack of expressivity.

2.2 Combinatorial expressions: simple varieties of trees

As announced above, our combinatorial model for expressions is the notion of simple varieties of trees [9, VII.3]. We consider expressions as a combinatorial class of $\mathcal{L}$ defined in terms of a sequence of sets of labels $\mathcal{A} = (\mathcal{A}_i)_{i\in\mathbb{Z}_{\geq 0}}$ indexed by their arity (or degree): $f \in \mathcal{A}_i$ has arity i and labels nodes with i children. For instance, the set of regular expressions over the alphabet $\{a, b\}$, can be described in this formalism using $\mathcal{A}_0 = \{\varepsilon, a, b\}$, $\mathcal{A}_1 = \{\star\}$, and $\mathcal{A}_2 = \{+, \bullet\}$, the other $\mathcal{A}_i$'s being empty for $i \geq 3$.

To avoid trivially uninteresting cases, we define combinatorial expressions as follows:

▶ **Definition 2.** *A set of* combinatorial expressions *of labels* $\mathcal{A} = (\mathcal{A}_i)_{i \in \mathbb{Z}_{\geq 0}}$ *is a combinatorial set of rooted plane trees whose nodes of arity* i *are labelled with elements of a finite set* $\mathcal{A}_i$*, under the additional conditions that* $\mathcal{A}_0 \neq \emptyset$ *and that there exists* $i \geq 2$ *such that* $\mathcal{A}_i \neq \emptyset$*. Its* characteristic series *is the formal power series* $\phi(y) = \sum_{i \geq 0} \phi_i z^i$*, where* $\phi_i = |\mathcal{A}_i|$*.*

Note that there can be infinitely many non-empty $\mathcal{A}_i$ and that a label can be in several $\mathcal{A}_i$'s: we can have the union label in $\mathcal{A}_i$ for any $i \geq 2$, to take its associativity into account.

If $\mathcal{L}$ is a set of combinatorial expressions of characteristic series $\phi(y)$, then we have the fundamental formal equation for its OGS $L(z)$:

$$L(z) = z\,\phi(L(z))\,. \tag{6}$$

In our running example, rational expressions, ϕ is a quadratic polynomial and Eq. (6) can be solved explicitly. But usually this is not the case. Fortunately, we can still work out the asymptotic equivalents directly from the functional equation Eq. (6), using implicit functions. Several conditions, gathered under the name of *Smooth inverse-function schema for trees* [9, Definition VII.3], need to be checked in order to apply this fundamental result:

▶ **Definition 3** (Smooth inverse-function schema for trees [9])**.** *A function* $L(z)$ *satisfying the functional equation* $L(z) = z\phi(L(z))$ *is said to belong to the smooth inverse-function schema if* $L(z)$ *is analytic at* $z = 0$*,* $\phi(y)$ *is analytic at* $y = 0$*,* $\phi(0) \neq 0$*, the coefficients of* ϕ *are non-negative,* ϕ *is not linear, and finally there exists a solution* τ *of the equation* $\phi(\tau) - \tau\phi'(\tau) = 0$*, with* $0 < \tau < \rho_\phi$ *where* ρ_ϕ *is the radius of convergence of* ϕ*.*

Let us check this definition when $\mathcal{L}$ is a set of combinatorial expressions of characteristic series $\phi(y)$. The fact that $\phi(y)$ is analytic at 0 means that there are not too many operators of arity i (typically, not super-exponential in i), which is indeed the case for all the examples of Fig. 1. The other conditions are satisfied by definition, except the last one. The solution τ always exists, but we have to ensure that it is smaller than the radius of convergence of ϕ. The following elementary observation is quite useful:

▶ Remark 4. If ϕ is a polynomial, i.e. only finitely many $\mathcal{A}_i$'s are non-empty, then a set of expressions of characteristic series ϕ belongs to the smooth inverse-function schema.

Consider a variation on regular expressions $\mathcal{L}'_R$, where $\mathcal{A}_0 = \{\varepsilon, a, b\}$, $\mathcal{A}_1 = \{\star\}$, $\mathcal{A}_2 = \{+, \bullet\}$, and $\mathcal{A}_i = \{+\}$ for any $i \geq 3$. Its characteristic series is $\phi(y) = 3 + y + 2y^2 + \frac{y^3}{1-y} = \frac{1}{1-y} + y^2 + 2$. Its radius of convergence is $\rho_\phi = 1$, and the positive solution of $\phi(\tau) - \tau\phi'(\tau) = 0$ is $\frac{\sqrt{5}-1}{2} \approx 0.618$. Then the characteristic series belong to the smooth inverse-function schema.

The following theorem is a key property in analytic combinatorics for the functions belonging to the smooth inverse-function schema. It is stated when $\phi(y)$ is *aperiodic*, i.e. when $\phi(y) = \psi(y^d)$ implies $d = 1$. This is not a real restriction as shown in [9, Ex. VI.17], and this implies that $L(z)$ is also aperiodic and then has a unique dominant singularity.

▶ **Theorem 5** (see [5, 9, 16])**.** *Let* $L(z)$ *that belongs to the smooth inverse-function schema with* $L(z) = z\phi(L(z))$*. Assume that* $\phi(y)$ *is aperiodic, then* $L(z)$ *has a unique dominant singularity* $\rho_L = \tau/\phi(\tau)$*. Moreover, there exist two analytic functions* $g(z)$ *and* $h(z)$ *analytic at* $z = \rho_L$*, such that, around* $z = \rho_L$ *we have:*

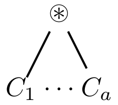
$$L(z) = g(z) - h(z)\sqrt{1 - z/\rho_L},\ \textit{with}\ h(\rho_L) \neq 0. \tag{7}$$

Moreover, we have $[z^n]L(z) \sim C_L \frac{\rho_L^{-n}}{n^{3/2}}$*, for some computable positive constant* C_L*.*

3 Main results

3.1 Simplification using an absorbing pattern: main result

Let $\mathcal{L}$ be a set of combinatorial expressions of sets of labels $(\mathcal{A}_i)_{i\geq 0}$. Let $\mathcal{P}$ be an element of $\mathcal{L}$ and let $\circledast$ be an element of $\mathcal{A}_a$ for some $a \geq 2$. If $L \in \mathcal{L}$, the *simplification of L following the absorbing pattern $\mathcal{P}$ for $\circledast$* is the element $S \in \mathcal{L}$ obtained by applying bottom-up the following rewriting rule:



$\rightsquigarrow \mathcal{P}$, whenever $C_i = \mathcal{P}$ for some $i \in \{1, \ldots, a\}$.

More formally, the simplification $s(L, \mathcal{P}, \circledast)$ of L following the absorbing pattern $\mathcal{P}$ for $\circledast$ is inductively defined by: $s(L, \mathcal{P}, \circledast) = L$ if L has size 1, $s(L, \mathcal{P}, \circledast) = \mathcal{P}$ if the root of L is $\circledast$ and at least one of its children C is such that $s(C, \mathcal{P}, \circledast) = \mathcal{P}$, and

$$s(L, \mathcal{P}, \circledast) = (\oplus, s(C_1, \mathcal{P}, \circledast), \ldots, s(C_d, \mathcal{P}, \circledast)), \text{ otherwise for } L = (\oplus, C_1, \ldots, C_d).$$

Of course, this simplification is purely syntactic, but the idea behind is that, when interpreted, the pattern $\mathcal{P}$ represents an absorbing element for the operator $\circledast$, hence the name. For instance, the pattern $\mathcal{P} = (a + b)^\star$ and operator $\circledast = +$ correspond to the fact that for languages, $\mathcal{P}$ represents $\Sigma^\star$, which is absorbing for the union $+$.

Let $\sigma_{\mathcal{P},\circledast}(L) = |s(L, \mathcal{P}, \circledast)|$ denote the size of the simplification of L, or just $\sigma(L)$ when there is no ambiguity about the $\mathcal{P}$ and the $\circledast$ considered. Our main result concerns the random variable σ when L is taken uniformly at random in $\mathcal{L}_n$.

▶ **Theorem 6.** *Let $\mathcal{L}$ be a set of combinatorial expressions whose OGS $L(z)$ belongs to the smooth inverse-function schema $L(z) = \phi(L(z))$, with ϕ aperiodic. Let $\mathcal{P} \in \mathcal{L}$ and let $\circledast$ be an operator of arity at least 2 of $\mathcal{L}$. Then the expected size of the simplification of a uniform random expression of size n in $\mathcal{L}$ following the absorbing pattern $\mathcal{P}$ for $\circledast$ tends to a constant as n tends to infinity: $\lim_{n\to\infty} \mathbb{E}_n[\sigma] = \delta$, for some positive δ. Furthermore, all moments of σ also converge: for every $i \in \mathbb{Z}_{\geq 1}$, $\lim_{n\to\infty} \mathbb{E}_n[\sigma^i] = \delta_i$ for some positive δ_i.*

Observe that it is not because the expectation converges that the moments converge too. Moreover, convergence of the moments has an implication in the analysis of algorithm, as stated in the following corollary.

▶ **Corollary 7.** *Let $\mathcal{L}$ be a set of expressions that satisfies the conditions of Theorem 6. Consider a polynomial time algorithm working on elements of $\mathcal{L}$. If we first simplify the expression before running the algorithm, then the process takes $\mathcal{O}(1)$ expected time. The simplification itself is done in linear expected time.*

Hence, the automaton construction studied in [18] is useless in the uniform random setting, since the output automaton has size polynomial in the input in the worst case. It is thus of expected constant size if we first simplify the input, according to Corollary 7.

We fix $\mathcal{L}$, ϕ, $\mathcal{P}$ and $\circledast$ for the remainder of Section 3, which is devoted to the proof of our main theorem, i.e., Theorem 6. A discussion of its consequences can be found in Section 4.

3.2 Completely reducible expression trees

An expression tree $L \in \mathcal{L}$ is *completely reducible* when $s(L, \mathcal{P}, \circledast) = \mathcal{P}$. We first study the set $\mathcal{R} \subset \mathcal{L}$ of all the completely reducible expression trees and denote by $R(z)$ its OGS.

Let $p = |\mathcal{P}|$ and let a be the arity of $\circledast$. An element of $\mathcal{R}$ is either $\mathcal{P}$ itself, or a tree of root $\circledast$ such that at least one of its subexpression is completely reducible. Hence, translating into OGS, we have the following equation for $R(z)$:

$$R(z) = z^p + z\left((L(z))^a - (L(z) - R(z))^a\right) . \tag{8}$$

We can also easily establish the following property:

▶ **Lemma 8.** *The radii of convergence of $R(z)$ and $L(z)$ are equal. Moreover, there exists $N_R \in \mathbb{Z}_{\geq 0}$ such that for every $n \geq N_R$, $[z^n]R(z) > 0$.*

We need to be more precise, as a fine grain description of $R(z)$ is an important ingredient of our proof of Theorem 6. We use an advanced analytic result established[4] by Drmota [5, Th. 1 and Prop. 3]. His theorem provides a precise estimate of the solutions of strongly connected systems of equations involving analytic functions around their (then) shared dominant singularity.

▶ **Proposition 9.** *The probability that a uniform random expression in $\mathcal{L}_n$ is completely reducible tends to a strictly positive constant γ as $n \to \infty$. Furthermore, locally around $z = \rho$, we have*

$$R(z) = g_R(z) - h_R(z)\sqrt{1 - \frac{z}{\rho}} , \tag{9}$$

where ρ is the radius of convergence of $L(z)$ and $R(z)$, and the functions $g_R(z)$ and $h_R(z)$ are analytic around $z = \rho$, and $h_R(\rho) \neq 0$.

Proof sketch. We define a complementary class $\mathcal{G} := \mathcal{L} \setminus \mathcal{R}$ of elements that do not reduce (completely) to $\mathcal{P}$. Let $G(z)$ denote its OGS. We observe that from Eq. (8) we have

$$\begin{cases} R(z) = & z^p + z\Big((G(z) + R(z))^a - G(z)^a\Big) , \\ G(z) = & \quad z\,\underline{\phi}\big(G(z) + R(z)\big) + zG(z)^a , \end{cases} \tag{10}$$

where $\underline{\phi}(x) = \phi(x) - x^a$. We demonstrate, simultaneously, that $R(z)$ and $G(z)$ verify the conclusions of the proposition by applying Drmota's multidimensional version of Theorem 5 (see [5, Thm. 1 and Prop. 3]). Thus we consider the implicit system above, in the variables z, R and G, and prove that it satisfies the conditions of Drmota's Theorem. This will then imply immediately our desired conclusions. We consider the corresponding algebraic system

$$[R, G] = \boldsymbol{F}(z, R, G), \text{ where } \boldsymbol{F}(z, R, G) = \Big[z^p + z\big((G + R)^a - G^a\big), \quad z\,\underline{\phi}\big(G + R\big) + zG^a\Big].$$

Now we need to check that all the hypotheses of Drmota's theorem are satisfied; this includes studying the Jacobian matrix of the system, which is

$$\boldsymbol{J}(z, R, G) = \begin{pmatrix} za(G + R)^{a-1} & za((G + R)^{a-1} - G^{a-1}) \\ z\underline{\phi}'(G + R) & z\underline{\phi}'(G + R) + zaG^{a-1} \end{pmatrix} .$$

Once done, we obtain Eq. (9).

From this formula and the Transfer Theorem, we conclude that $[z^n]R(z)$ satisfies an asymptotic formula similar to the one of $[z^n]L(z)$ in Theorem 5, but with a different multiplicative constant. This yields the existence of γ and thus concludes the proof. ◀

[4] It turns out that Drmota's theorem needs a little rectification, given in [1]. The theorem, among other things, gives a way to localize the dominant singularity, which was proved wrong. Fortunately, we do not need to rely on this to characterize the singularity in our case, so there is no issue for us, as we do not use the incorrect part of this result.

▶ Remark 10. If we had several reduction rules with the same pattern, rather than just one, the conclusions from Proposition 9 remain true. For instance, for regular expressions, we could add the reduction rule $\mathcal{P}^\star \rightsquigarrow \mathcal{P}$ in addition to the fact that $\mathcal{P}$ is absorbing for $+$.

For this we introduce an auxiliary series $A(x) = \sum_{i\geq 1} a_i x^i$ where a_i is the number of operations of arity i having $\mathcal{P}$ as an absorbing element. The system in the proof of Proposition 9 rewrites

$$\begin{cases} R(z) = z^p + z\Big(A(G(z)+R(z)) - A(G(z))\Big)\,, \\ G(z) = z\,\underline{\phi}\big(G(z)+R(z)\big) + zA(G(z))\,, \end{cases} \tag{11}$$

with $\underline{\phi}(x) = \phi(x) - A(x)$. The limit probability γ of Proposition 9 admits an elegant formula

$$\gamma = \frac{A'(\tau_L) - A'(\tau_L - \tau_R)}{\phi'(\tau_L) - A'(\tau_L - \tau_R)}, \text{ where } \tau_L = L(\rho) \text{ and } \tau_R = R(\rho).$$

3.3 Estimating the expectation and the moments

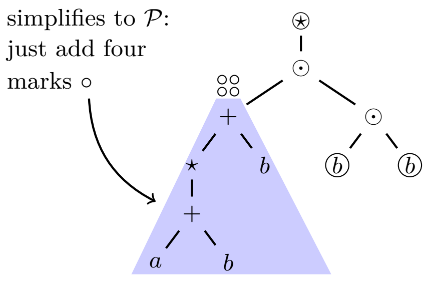
Following the method explained in Section 2.1, we use bivariate generating series to keep track of the size of the reduced expression. We therefore define $L(z,u)$ and $R(z,u)$ by (recall that $\sigma(L)$ is the size of the simplification of L):

$$L(z,u) = \sum_{L\in\mathcal{L}} z^{|L|} u^{\sigma(L)} \text{ and } R(z,u) = \sum_{R\in\mathcal{R}} z^{|R|} u^{\sigma(R)}.$$

Of course, since all elements of $\mathcal{R}$ simplify to $\mathcal{P}$, we have $R(z,u) = u^p R(z)$, with $p = |\mathcal{P}|$.

As in Section 2.1, we want to mark the expressions so that the number of marks is equal to the size of the simplified expression. For this, we put one mark on the nodes that are not simplified, and p marks on the root of a subtree that simplifies to $\mathcal{P}$, to take into account the size of $\mathcal{P}$ once reduced. We just have to reformulate the combinatorial specification and to enrich it with marks. As an example, let us consider regular expressions $\mathcal{L}_R$, with $\mathcal{P} = (a+b)^\star$ and $\circledast = +$. We mark an element $L \in \mathcal{L}_R$ the following way (we denote by $\mathcal{R}_R$ the completely reducible regular expressions):

- If $L \in \mathcal{R}_R$ then we add p marks at the root.
- If the root of L is not $+$, then we mark its children inductively (if any), and mark the root of L.
- If the root of L is $+$ and its children are all not in $\mathcal{R}_R$, then we mark its children inductively, and mark the root of L.



This expression has size 9 after simplificiation.

This way of rewriting the decomposition of $\mathcal{L}_R$ is unambiguous except for $\mathcal{P}$ which is in $\mathcal{R}_R$ and which can also be built by the standard induction, so we have to remove it once in the specification. This yields, where $\mathcal{L}_R^\circ$ and $\mathcal{R}_R^\circ$ respectively denote the marked versions of $\mathcal{L}_R$ and $\mathcal{R}_R$, and where $\mathcal{G}_R^\circ := \mathcal{L}_R^\circ \setminus \mathcal{R}_R^\circ$:

$$\mathcal{L}_R^\circ = (\mathcal{R}_R - \mathcal{P})^{\circ\circ}_{\circ\circ} + \textcircled{a} + \textcircled{b} + \textcircled{ε} + \begin{array}{c}\circledast\\ | \\ \mathcal{L}_R^\circ\end{array} + \begin{array}{c}\odot\\ /\,\backslash \\ \mathcal{L}_R^\circ\ \mathcal{L}_R^\circ\end{array} + \begin{array}{c}\oplus\\ /\,\backslash \\ \mathcal{G}_R^\circ\ \mathcal{G}_R^\circ\end{array}.$$

From this we get $L_R(z,u) = (R(z) - z^4)u^4 + 3zu + uzL_R(z,u) + zuL_R(z,u)^2 + zuG_R(z,u)^2$.

This construction generalizes easily, and we get the following lemma.

▶ **Lemma 11.** *We have the following equation for the BGS $L(z,u)$ where z counts the size of an expression and u the size of its simplification:*

$$L(z,u) = (R(z) - z^p)u^p + zu\big(\underline{\phi}(L(z,u)) + (L(z,u) - R(z,u))^a\big), \qquad (12)$$

where $\underline{\phi}(x) = \phi(x) - x^a$ and where a is the arity of $\circledast$.

At this point, we want to apply Eq. (4) to compute the expectation of σ. Hence, we differentiate with respect to u, set $u = 1$ and re-arrange the expression, obtaining

$$\frac{d}{du}L(z,u)\Big|_{u=1} = \frac{N\big(z, L(z), R(z)\big)}{D\big(z, L(z), R(z), \phi'(L(z))\big)}, \text{ where } N \text{ and } D \text{ are polynomials.} \qquad (13)$$

From this, and using an adaptation of another result of Drmota [6, Lemma 2.26] to handle quotients, we get the following lemma.

▶ **Lemma 12.** *There exist two functions $\tilde{g}(z)$ and $\tilde{h}(z)$ that are analytic at $z = \rho$, such that locally around $z = \rho$ we have the expansion $\frac{d}{du}L(z,u)\Big|_{u=1} = \tilde{g}(z) - \tilde{h}(z)\sqrt{1 - z/\rho}$.*

We can now conclude the proof of the first part of Theorem 6: using the Transfer Theorem, we obtain from Lemma 12 that, if $\tilde{h}(\rho) > 0$, then $[z^n]\frac{d}{du}L(z,u)|_{u=1} \sim \frac{\tilde{h}(\rho)}{2\sqrt{\pi}}\rho^{-n}n^{-3/2}$, so that, by Theorem 5, $\mathbb{E}_n[\sigma]$ tends to the constant $\delta = \frac{\tilde{h}(\rho)}{2C_L\sqrt{\pi}}$, since $\rho = \rho_L$. The other cases are not possible, as $\tilde{h}(\rho) < 0$ yields a negative asymptotic equivalent and $\tilde{h}(\rho) = 0$ implies that the expected size of the simplification tends to 0, but it is not possible as it is at least $p > 0$.

▶ Remark 13. Using a computer algebra software, we can deal symbolically with all the formulas obtained in the article. When applied to regular expressions, we obtain that $\delta \approx 3{,}624{,}217.39$, which is a prohibitively large number for most simulations, while the proportion of completely reducible expressions tends to $\gamma \approx 0.0016336$.

The proof for higher moments follows a similar principle as for the expectation; when we differentiate several times on u Eq. (12) from Lemma 11, we get the same factor for the highest order derivative, hence the same denominator $D(z, L(z), R(z), \phi'(L(z)))$ as in Eq. (13). As the remaining terms also fulfill a local expansion like the one in Lemma 12, we can conclude exactly as for the expectation.

4 Conclusion

We have seen in this paper that uniform random expressions have to be considered with great care when using them to analyze the efficiency of algorithms or tools. As soon as there is an absorbing pattern, everything becomes mostly trivial, even for polynomial algorithms or constructions, as stated in Corollary 7.

One could alter the specifications to try to obtain a different behavior. For instance, by weighting some of the labels (trying to get less $\star$ in random expressions). This can easily be done directly on the specification, by adding several identical operators or even positive weights on the operators. The random generation can still be done quite efficiently. However, we have not changed the framework, and if we try to specify weighted regular expressions with, for instance:

$$\mathcal{L}_R = 10 \times a + 10 \times b + \varepsilon + \overset{\star}{\underset{\mathcal{L}_R}{|}} + 20 \times \overset{\bullet}{\underset{\mathcal{L}_R\ \mathcal{L}_R}{/\backslash}} + 15 \times \overset{+}{\underset{\mathcal{L}_R\ \mathcal{L}_R}{/\backslash}}, \tag{14}$$

we can still apply Theorem 6, we just get a larger limit for the expected size.

Another idea is to specify expressions using combinatorial systems. For instance, we can prevent consecutive stars in regular expressions using the system:

$$\mathcal{L}_R = \mathcal{S} + \mathcal{T}; \quad \mathcal{S} = \overset{\star}{\underset{\mathcal{T}}{|}}; \quad \mathcal{T} = a + b + \varepsilon + \overset{\bullet}{\underset{\mathcal{L}_R\ \mathcal{L}_R}{/\backslash}} + \overset{+}{\underset{\mathcal{L}_R\ \mathcal{L}_R}{/\backslash}}.$$

This is not covered by our main theorem, but it is work in progress to prove that the same result holds, using Drmota's tools to handle systems of equations like this one.

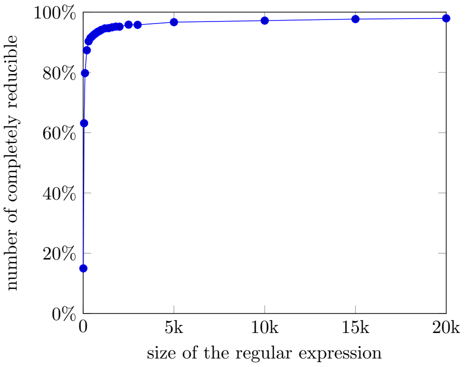


Figure 3 Experimental study of the proportion of completely reducible, for BST-like distribution, with $\mathbb{P}(\star) = \mathbb{P}(\bullet) = \mathbb{P}(+) = \frac{1}{3}$.

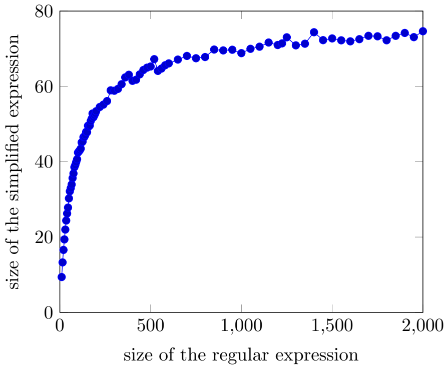


Figure 4 Experimental study of the size of a random uniform regular expression after applying a large set of simplification rules.

Yet a third idea is to completely change the distribution on the set of expressions. The other natural probability on trees is the so-called *BST-like* distribution, obtained on regular expressions as follows to generate an expression of size n: if $n = 1$ return a, b or ε; if $n = 2$ return $a^\star$, $b^\star$ or $\varepsilon^\star$; otherwise, randomly choose an operator in $\{\star, \bullet, +\}$ and if it is binary, choose the size of its left child uniformly in $\{1, \dots, n-2\}$. This distribution is not uniform, but it is often used for testing tools. See [19] for a theoretical study using this distribution on regular expressions. It is not clear if the conclusion of Theorem 6 still holds for these distributions, but we believe that they are not interesting either: on Fig. 3, we experimentally computed the ratio of expressions that can be simplified into $(a + b)^\star$, using a whole set of simplification rules, and obtained that this ratio tends to 1 as n tends to infinity: the situation is even worse than in the uniform case (see also [3, 4]).

If we apply this whole set of simplification rules to uniform random regular expressions, we get the curve depicted in Fig. 4. Apparently, the expected size of the simplified expression tends to a value around 75, which is large, but much smaller than the value 3,624,217 obtained for just using the absorbing pattern.

The real conclusion is that natural distributions on trees seem to be useless when it comes to obtaining good distributions on formulas. At this point, we have no idea on how to produce a good distribution in the general framework, sufficiently simple to have efficient random samplers and to be amenable to mathematical analysis.

References

1 Jason P. Bell, Stanley Burris, and Karen A. Yeats. Characteristic Points of Recursive Systems. *Electr. J. Comb.*, 17(1), 2010.

2 Sabine Broda, António Machiavelo, Nelma Moreira, and Rogério Reis. Average Size of Automata Constructions from Regular Expressions. *Bulletin of the EATCS*, 116, 2015.

3 Nicolas Broutin and Cécile Mailler. And/or trees: A local limit point of view. *Random Struct. Algorithms*, 53(1):15–58, 2018. doi:10.1002/rsa.20758.

4 Brigitte Chauvin, Danièle Gardy, and Cécile Mailler. A sprouting tree model for random boolean functions. *Random Struct. Algorithms*, 47(4):635–662, 2015. doi:10.1002/rsa.20567.

5 Michael Drmota. Systems of functional equations. *Random Struct. Algorithms*, 10(1-2):103–124, 1997.

6 Michael Drmota. *Random Trees: An Interplay Between Combinatorics and Probability*. Springer Publishing Company, Incorporated, 1st edition, 2009.

7 Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann Samplers for the Random Generation of Combinatorial Structures. *Combinatorics, Probability & Computing*, 13(4-5):577–625, 2004.

8 Philippe Flajolet and Andrew M. Odlyzko. The Average Height of Binary Trees and Other Simple Trees. *J. Comput. Syst. Sci.*, 25(2):171–213, 1982. doi:10.1016/0022-0000(82)90004-6.

9 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009. URL: http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521898065.

10 Philippe Flajolet, Paolo Sipala, and Jean-Marc Steyaert. Analytic Variations on the Common Subexpression Problem. In Mike Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 1990. doi:10.1007/BFb0032034.

11 Philippe Flajolet and Jean-Marc Steyaert. A Complexity Calculus for Recursive Tree Algorithms. *Mathematical Systems Theory*, 19(4):301–331, 1987. doi:10.1007/BF01704918.

12 Danièle Gardy. Random Boolean expressions. In David, René, Gardy, Danièle, Lescanne, Pierre, Zaionc, and Marek, editors, *Computational Logic and Applications, CLA '05*, volume DMTCS Proceedings vol. AF, Computational Logic and Applications (CLA '05) of *DMTCS Proceedings*, pages 1–36, Chambéry, France, 2005. Discrete Mathematics and Theoretical Computer Science. URL: https://hal.inria.fr/hal-01183339.

13 Antoine Genitrini and Bernhard Gittenberger. No Shannon effect on probability distributions on Boolean functions induced by random expressions. In *Discrete Mathematics and Theoretical Computer Science*, pages 303–316. Discrete Mathematics and Theoretical Computer Science, 2010.

14 Antoine Genitrini, Bernhard Gittenberger, and Cécile Mailler. No Shannon effect induced by And/Or trees. In *25th International Meeting on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms*, pages 109–120, 2014.

15 Jonathan Lee and Jeffrey Shallit. Enumerating Regular Expressions and Their Languages. In Michael Domaratzki, Alexander Okhotin, Kai Salomaa, and Sheng Yu, editors, *Implementation and Application of Automata, 9th International Conference, CIAA 2004, Kingston, Canada, July 22-24, 2004*, volume 3317 of *Lecture Notes in Computer Science*, pages 2–22. Springer, 2004.

16 A Meir and J.W Moon. On an asymptotic method in enumeration. *Journal of Combinatorial Theory, Series A*, 51(1):77–89, 1989. doi:10.1016/0097-3165(89)90078-2.

17 Michel Nguyên-Thê. *Distribution of Valuations on Trees*. Theses, Ecole Polytechnique X, February 2004. URL: https://pastel.archives-ouvertes.fr/pastel-00000839.

18 Cyril Nicaud. On the Average Size of Glushkov's Automata. In Adrian-Horia Dediu, Armand-Mihai Ionescu, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings*, volume 5457 of *Lecture Notes in Computer Science*, pages 626–637. Springer, 2009.

19 Cyril Nicaud, Carine Pivoteau, and Benoît Razet. Average Analysis of Glushkov Automata under a BST-Like Model. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 388–399. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

20 Albert Nijenhuis and Herbert S Wilf. Combinatorial algorithms for computers and calculators. *Computer Science and Applied Mathematics, New York: Academic Press, 1978, 2nd ed.*, 1978.

21 Mike Paterson, editor. *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*. Springer, 1990. doi:10.1007/BFb0032016.

Lower Bounds for Multilinear Order-Restricted ABPs

C. Ramya
Chennai Mathematical Institute, Chennai, India
ramya@cse.iitm.ac.in

B. V. Raghavendra Rao
IIT Madras, Chennai, India
bvrr@cse.iitm.ac.in

Abstract

Proving super-polynomial lower bounds on the size of syntactic multilinear Algebraic Branching Programs (smABPs) computing an explicit polynomial is a challenging problem in Algebraic Complexity Theory. The order in which variables in $\{x_1, \ldots, x_n\}$ appear along any source to sink path in an smABP can be viewed as a permutation in S_n. In this article, we consider the following special classes of smABPs where the order of occurrence of variables along a source to sink path is restricted:

1. **Strict circular-interval ABPs:** For every sub-program the index set of variables occurring in it is contained in some circular interval of $\{1, \ldots, n\}$.
2. **$\mathcal{L}$-ordered ABPs:** There is a set of $\mathcal{L}$ permutations (orders) of variables such that every source to sink path in the smABP reads variables in one of these $\mathcal{L}$ orders, where $\mathcal{L} \leq 2^{n^{1/2-\epsilon}}$ for some $\epsilon > 0$.

We prove exponential (i.e., $2^{\Omega(n^\delta)}$, $\delta > 0$) lower bounds on the size of above models computing an explicit multilinear $2n$-variate polynomial in VP.

As a main ingredient in our lower bounds, we show that any polynomial that can be computed by an smABP of size S, can be written as a sum of $O(S)$ many multilinear polynomials where each summand is a product of two polynomials in at most $2n/3$ variables, computable by smABPs. As a corollary, we show that any size S syntactic multilinear ABP can be transformed into a size $S^{O(\sqrt{n})}$ depth four syntactic multilinear $\Sigma\Pi\Sigma\Pi$ circuit where the bottom Σ gates compute polynomials on at most $O(\sqrt{n})$ variables.

Finally, we compare the above models with other standard models for computing multilinear polynomials.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Theory of computation → Circuit complexity

Keywords and phrases Computational complexity, Algebraic complexity theory, Polynomials

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.52

Related Version A full version of the paper is available at [17], `https://arxiv.org/abs/1901.04377`.

1 Introduction

Algebraic Complexity Theory is concerned with the classification of polynomials based on the number of arithmetic operations required to compute a polynomial from variables and constants. Arithmetic circuits are standard models for algebraic computation. One of the primary tasks in Algebraic Complexity Theory is to prove lower bounds on the size of arithmetic circuits computing an explicit polynomial. Valiant [23] conjectured that the polynomial defined by the permanent of an $n \times n$ symbolic matrix is not computable by polynomial size arithmetic circuits. This is known as Valiant's hypothesis and is one of the central questions in Algebraic Complexity Theory.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 52; pp. 52:1–52:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The best known size lower bound for general classes of arithmetic circuits is only super-linear in the number of variables [5]. Despite several approaches, the problem of proving lower bounds for general classes of arithmetic circuits has remained elusive. Naturally, there have been efforts to prove lower bounds for special classes of arithmetic circuits which led to the development of several lower bound techniques. Structural restrictions such as depth and fan-out, semantic restrictions such as multilinearity and homogeneity have received widespread attention in the literature.

Agrawal and Vinay [1] showed that proving exponential lower bounds for depth four circuits is sufficient to prove Valiant's hypothesis. This initiated several attempts at proving lower bounds for constant depth circuits. (See [21] for a detailed survey of these results.)

Among other restrictions, *multilinear circuits* where every gate computes a multilinear polynomial are well studied. Multilinear circuits are natural models for computing multilinear polynomials. It is sometimes more useful to consider a natural syntactic sub-class of multilinear circuits. A circuit is *syntactic multilinear* if the children of every product gate depend on disjoint sets of variables. Raz [18] obtained super-polynomial lower bounds on the size of syntactic multilinear formulas computing the determinant or permanent polynomial. This was improved to an exponential lower bound for constant-depth multilinear circuits [20, 7, 6]. However, the best known lower bound for general syntactic multilinear circuits is almost quadratic in the number of variables [2].

Algebraic Branching Programs (ABPs) are special classes of arithmetic circuits that have been studied extensively in the past. Nisan [15] obtained an exact complexity characterization of ABPs in the non-commutative setting. The problem of proving size lower bounds for the class of algebraic branching programs is widely open, even with restrictions such as homogeneity or syntactic multilinearity. When the ABP is restricted to be *homogeneous*, the best known lower bound is only quadratic in the number of variables [13]. The situation is not better in the case of syntactic multilinear ABPs, where no super-quadratic lower bound is known [2].

Models and results. In this article, we are interested in syntactic multilinear ABPs in which the order of appearance of variables along any path in the ABP is restricted. To begin with, we give a decomposition theorem for smABPs. The decomposition obtains two disjoint sets E_1 and E_2 of edges in the branching program P with source s and sink t such that the polynomial computed by P can be expressed as a sum of $\sum_{(u,v)\in E_1}[s,u]\cdot\mathsf{label}(u,v)\cdot[v,t]$ and $\sum_{(w,a)\in E_2}[s,w]\cdot\mathsf{label}(w,a)\cdot[a,t]$, where $[p,q]$ is the polynomial computed by sub-program in P with source p and sink q. Also, the sets E_1 and E_2 are chosen carefully such that the sub-programs obtained are more or less balanced in terms of the number of variables.

In the following theorem, for nodes u, v in an smABP P, we denote $[u,v]$ is the polynomial computed by sub-program in P with source u and sink v. Also, let $X_{u,v}$ denote the set of all variables that occur as labels in any path from node u to node v in P. More formally, we prove:

▶ **Theorem 1.** *Let P be an smABP of size S computing $f \in \mathbb{F}[x_1,\ldots,x_n]$. There exists edges $\{(u_1,v_1),\ldots,(u_m,v_m)\}$ and $\{(w_1,a_1),\ldots,(w_r,a_r)\}$ in P such that*

(1) *For $i \in [m]$, $n/3 \le |X_{s,u_i}| \le 2n/3$; and*

(2) *For $i \in [r]$, $|X_{s,w_i}| < n/3$ and $|X_{a_i,t}| \le 2n/3$; and*

(3) $f = \sum_{i=1}^{m}[s,u_i]\cdot\textit{label}(u_i,v_i)\cdot[v_i,t] + \sum_{i=1}^{r}[s,w_i]\cdot\textit{label}(w_i,a_i)\cdot[a_i,t]$.

Let $\Sigma\Pi^{[\sqrt{n}]}(\Sigma\Pi)^{[\sqrt{n}]}$ denote the class of depth four arithmetic circuits where the top layer of Π gates are products of at most $O(\sqrt{n})$ polynomials each being a multilinear polynomial on $O(\sqrt{n})$ variables. As an immediate corollary of the above decomposition, we obtain the following low-arity version of the depth reduction in [1, 22] for the case of smABPs:

▶ **Corollary 1.** *Let P be a syntactic multilinear ABP of size S computing a polynomial f in $\mathbb{F}[x_1, \ldots, x_n]$. Then there exists a $\Sigma\Pi^{[\sqrt{n}]}(\Sigma\Pi)^{[\sqrt{n}]}$ syntactic multilinear formula of size $2^{O(\sqrt{n}\log S)}$ computing f.*

Using the structural property of the parse trees of formulas obtained from smABPs, we prove exponential size lower bounds for two classes of smABPs with restrictions on the variable order.

(1) Strict circular-interval ABPs: A strict circular-interval ABP is an smABP in which the index set of variables in every subprogram is contained in some circular-interval in $\{1, \ldots, n\}$ (see Section 4.1 for a formal definition). Every multilinear polynomial can be computed by a strict circular-interval ABP and hence it is a universal model for computing multilinear polynomials. We obtain an exponential lower bound on the size of any strict circular-interval ABP computing the explicit multilinear polynomial defined by Raz and Yehudayoff in [19] (see Section 2 for definition of the polynomial).

▶ **Theorem 2.** *There exists an explicit multilinear polynomial g in $\mathbb{F}[x_1, \ldots, x_n]$ such that any strict circular-interval ABP computing g requires size $2^{\Omega(\sqrt{n}/\log n)}$.*

Another sub-class of smABPs that we study is the class of *bounded-order* smABPs.

(2) $\mathcal{L}$-ordered smABPs: Jansen [11] introduced *Ordered smABPs* which are branching programs with source s and sink t such that every path from s to t reads variables in a fixed order $\pi \in S_n$. Jansen [11] translated the exponential lower bound for the non-commutative model in [15] to ordered smABPs. Ordered smABPs have also been studied in the context of the polynomial identity testing problem. Further, it is shown in [12] that ordered smABPs are equivalent to Read-Once Oblivious Algebraic Branching Programs (ROABPs for short., see Section 2 for a definition). Polynomial time white-box and quasi-polynomial time black-box algorithms were obtained for identity testing of polynomials computed by ordered smABPs, (see [12, 9, 10] and the references therein).

A natural generalization of ordered smABPs is to allow multiple orders. An smABP is $\mathcal{L}$-ordered if variables can occur along any source to sink path in one of the $\mathcal{L}$ fixed orders. Since there are at most 2^n multilinear monomials in n variables, any multilinear n variate polynomial can be computed by 2^n-ordered ABPs. In this article, by exploiting a simple structural property of $\mathcal{L}$-ordered ABPs, we prove a sub-exponential lower bound for sum of $\mathcal{L}$-ordered smABPs when $\mathcal{L} = O(2^{n^{1/2-\epsilon}})$ for some $\epsilon > 0$:

▶ **Theorem 3.** *Let $\mathcal{L} \leq 2^{n^{1/2-\epsilon}}, \epsilon > 0$ and $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ be multilinear polynomials computed by $\mathcal{L}$-ordered ABPs of size $S_1, \ldots, S_m$ respectively. There exists an explicit multilinear polynomial g in $\mathbb{F}[x_1, \ldots, x_n]$ such that if $g = f_1 + \cdots + f_m$ then either $m = 2^{\Omega(n^{1/40})}$ or there is an $i \in [m]$ such that $S_i = 2^{\Omega(n^{1/40})}$.*

Further, we compare strict circular-interval ABPs and $\mathcal{L}$-ordered ABPs to other mulitlinear models of computation. We show that the construction given in [12] for the equivalence of ROABPs and 1-ordered ABPs can be generalized to $\mathcal{L}$-ordered smABPs. In particular, in Theorem 11 we prove that an $\mathcal{L}$-ordered ABP of size S can be transformed into an equivalent $\mathcal{L}$-pass smABP (see Section 2 for a definition) of size $\mathsf{poly}(S, \mathcal{L})$. Though the overall idea is simple, the construction requires a lot of book-keeping of variable orders. In the context of strict circular-interval ABPs, we compare strict circular-interval ABPs with the simplest class of smABPs: Read-Once Oblivious Algebraic Branching Programs. [15] gives exponential size lower bounds for ROABPs (ABPs on n variables with $n+1$ layers such that every edge from a node in one layer L_i to a node in the successive layer is labeled by a uni-variate polynomial in $\mathbb{F}[x_i]$). Trivially, every strict circular-interval ABP is an ROABP. In Corollary 16, we show that a generalization of strict circular-interval ABPs called *circular-interval* ABPs are more powerful than sum of ROABPs.

Related works. In an independent and simultaneous work, Kumar, Oliveira and Saptharishi [14] show an improved depth reduction (along with several other results) for syntactic multilinear circuits. In [14] (Lemma 6.1 together with Theorem 5.1), the authors show that any *multi-k-ic circuit* of size S can be transformed into an equivalent multi-k-ic $\Sigma\Pi(\Sigma\Pi)^t$ circuit of size $2^{kt} \cdot S^{O(kn/t)}$. Thus, we have $k = 1$ in the case of syntactic multilinear circuits and the circuit size bound in Lemma 6.1 of [14] matches the size bound in Corollary 1 of this paper when $t = \sqrt{n}$. However, Corollary 1 works for syntactic multilinear ABPs while the result in [14] is for the more general class of syntactic multilinear circuits.

In [16], the authors prove lower bounds for sum of $\mathcal{L}$-pass ABPs computing a multilinear polynomial in VP. The result in Theorem 3 is much stronger than the results in [16], since it allows exponentially many orders, whereas the arguments in [16] work only when $\mathcal{L} = o(n^{1/12})$.

Saptharishi and coauthors (through a personal communication) have shown that $\mathcal{L}$-ordered smABPs can be written as a sum of $\mathcal{L}$ ROABPs. While, this will help in significantly improving the lower bound given in Theorem 3, the restriction on $\mathcal{L}$ will be much smaller than $2^{n^{1/2-\epsilon}}$ as far as we are aware.

In [4], Arvind and Raja considered interval ABPs where for every node v reachable from the source, the index set of variables in the sub-program with v as the sink node must be an interval in $[1, n]$. They proved an exponential size lower bound for interval ABPs assuming the *sum-of-squares* conjecture. Although our model is more restrictive than the one in [4], our lower bound argument is unconditional.

Proofs that are omitted due to space constraints can be found int he full version [17].

2 Preliminaries

In this section, we include necessary definitions of models and notations used in this article. Let $\mathbb{F}$ be a field and $X = \{x_1, \ldots, x_n\}$ denote a finite set of variables.

An *arithmetic circuit* $\mathcal{C}$ over $\mathbb{F}$ is a directed acyclic graph with vertices of in-degree zero or two. A vertex of out-degree 0 is called an output gate. A node of in-degree zero is called an input gate and is labeled by elements from $X \cup \mathbb{F}$. Every other gate is labeled either by $+$ or $\times$. Every gate in $\mathcal{C}$ naturally computes a polynomial in $\mathbb{F}[X]$ and the polynomial computed by $\mathcal{C}$ is the polynomial computed at the output gate. When the circuit has more than one output gate, the circuit computes a set of polynomials. The *size* of $\mathcal{C}$ is the number of gates in $\mathcal{C}$ and *depth* of $\mathcal{C}$ is the length of the longest path from an input gate to the output gate in $\mathcal{C}$. An *arithmetic formula* is an arithmetic circuit where the underlying undirected graph is a tree.

A *parse tree* T of an arithmetic formula F is a sub-tree of F containing the output gate of F such that for every $+$ gate v of F that is included in T, exactly one child of v is in T and for every $\times$ gate u that is in T, both children of u are in T.

An *algebraic branching program* (ABP) P is a directed acyclic graph with one vertex s of in-degree zero (source) and one vertex t of out-degree zero (sink). The vertices of the graph are partitioned into layers $L_0, L_1, \ldots, L_\ell$ where edges are from vertices in layer L_i to those in L_{i+1} for every $0 \leq i \leq \ell - 1$. The source node s is the only vertex in layer L_0 and the sink t is the only vertex in layer L_ℓ. Every edge in e in P is labeled by an element in $X \cup \mathbb{F}$ (denoted by $\mathsf{label}(e)$). The width of P is $\max_i\{|L_i|\}$ and size is the number of nodes in P. For a path ρ the weight, $\mathsf{wt}(\rho)$ be the product of its edge labels. The polynomial computed by an ABP P is the sum of weights of all s to t paths in P. For nodes u and v in P, let $[u, v]_P$ denote the polynomial computed by the sub-program of P with u as the source node and v as the sink node. We drop the subscript from $[u, v]_P$ when P is clear from the context.

Let $X_{u,v}$ denote the set of all variables that occur as labels in any path from u to v in P.

An ABP P is said to be *syntactic multilinear* (smABP) if every variable occurs at most once in every s to t path in P, it is said to be an *oblivious* ABP if for every layer L in P, there is a variable x_{i_L} such that every edge from the layer L is labeled from $\{x_{i_L}\} \cup \mathbb{F}$. A *Read-Once Oblivious* (ROABP) is an oblivious smABP such that every variable appears as an edge label in at most one layer.

Anderson et al. [3] defined the class of $\mathcal{L}$-pass smABPs. An oblivious smABP P is $\mathcal{L}$-pass, if there are layers $i_1 < i_2 < \ldots < i_{\mathcal{L}}$ such that for every j, between layers i_j and i_{j+1} the program P is an ROABP. Let π be a permutation of $\{1, \ldots, n\}$ and P be an smABP computing an n-variate multilinear polynomial. An s to t path ρ in P is said to be *consistent* with π, if the variable labels in ρ occur as per the order given by π, i.e., if x_i and x_j occur as edge labels in ρ in that order, then $\pi(i) < \pi(j)$. For a node v of P, v is said to be consistent with π, if every s to v path is consistent with π.

An smABP P is said to be $\mathcal{L}$-*ordered*, if there are $\mathcal{L}$ permutations $\pi_1, \ldots, \pi_{\mathcal{L}}$ such that for every s to t path ρ in P, there is an $1 \le i \le \mathcal{L}$ such that ρ is consistent with π_i.

We now review the partial derivative matrix of a polynomial introduced in [18]. Let $X = Y \cup Z$ be such that $Y \cap Z = \emptyset$ and $|Y| = |Z|$. It is convenient to represent the partition $X = Y \cup Z$ as an injective function $\varphi : X \to Y \cup Z$. For a polynomial f, let f^{φ} be the polynomial obtained by substituting each variable $x_i = \varphi(x_i)$ for $1 \le i \le n$.

▶ **Definition 1** (Partial Derivative Matrix., [18]). *Let $f \in \mathbb{F}[X]$ be a multilinear polynomial. The* partial derivative matrix *of f (denoted by M_f) with respect to the partition $\varphi : X \to Y \cup Z$ is a $2^m \times 2^m$ matrix defined as follows. For multilinear monomials p and q in variables Y and Z respectively, the entry $M_f[p, q]$ is the coefficient of the monomial pq in f^{φ}.*

For a polynomial f and a partition φ, let $\mathsf{rank}_{\varphi}(f)$ denote the rank of the matrix $M_{f^{\varphi}}$ over the field $\mathbb{F}$. The following properties of $\mathsf{rank}_{\varphi}(f)$ are useful:

▶ **Lemma 4** ([18], Propositions 3.1, 3.2 and 3.3). *Let $f, g \in \mathbb{F}[Y, Z]$ be multilinear polynomials. Then, (1) $\mathsf{rank}_{\varphi}(f + g) \le \mathsf{rank}_{\varphi}(f) + \mathsf{rank}_{\varphi}(g)$; (2) If $\mathsf{var}(f) \cap \mathsf{var}(g) = \emptyset$, then $\mathsf{rank}_{\varphi}(fg) = \mathsf{rank}_{\varphi}(f) \cdot \mathsf{rank}_{\varphi}(g)$; and (3) If $f \in \mathbb{F}[Y_1, Z_1]$ for $Y_1 \subseteq Y, Z_1 \subseteq Z$, then $\mathsf{rank}_{\varphi}(f) \le 2^{\min\{|Y_1|,|Z_1|\}}$.*

Let $\mathcal{D}$ denote the uniform distribution on the set of all partitions $\varphi : X \to Y \cup Z$, where $|Y| = |Z| = |X|/2$. The following is known about the rank of ROABPs:

▶ **Lemma 5** ([16], Corollary 1). *Let f be an N-variate multilinear polynomial computed by an ROABP of size S. Then,*

$$\Pr_{\varphi \sim \mathcal{D}}[\mathsf{rank}_{\varphi}(f) \le S^{\log N} 2^{N/2 - N^{1/5}}] \ge 1 - 2^{-N^{1/5}}.$$

We need the following polynomial defined in [19] to prove lower bounds:

▶ **Definition 2** (Full rank Polynomial., [19]). *Let $n \in \mathbb{N}$ be even and $\mathcal{W} = \{w_{i,k,j}\}_{i,k,j \in [n]}$. For any two integers $i, j \in \mathbb{N}$, we define an interval $[i, j] = \{k \in \mathbb{N}, i \le k \le j\}$. Let $|[i, j]| = j - i + 1$, $X_{i,j} = \{x_p \mid p \in [i, j]\}$. Let $\mathbb{G} = \mathbb{F}(\mathcal{W})$, the rational function field. For every $[i, j]$ such that $|[i, j]|$ is even we define a polynomial $g_{i,j} \in \mathbb{G}[X]$ as $g_{i,j} = 1$ when $|[i, j]| = 0$ and if $|[i, j]| > 0$ then, $g_{i,j} \triangleq (1 + x_i x_j) g_{i+1,j-1} + \sum_k w_{i,k,j} g_{i,k} g_{k+1,j}$. where x_k, $w_{i,k,j}$ are distinct variables, $i \le k \le j$ and the summation is over $k \in [i + 1, j - 2]$ such that $|[i, k]|$ is even. Let $g \triangleq g_{1,n}$.*

It is known that for any partition $\varphi \sim \mathcal{D}$, $\mathsf{rank}_{\varphi}(g)$ is the maximum possible value:

▶ **Lemma 6** ([19], Lemma 4.3). *Let $n \in \mathbb{N}$ be even and $\mathbb{G}$ be as above. Let $g \in \mathbb{G}[X]$ be the polynomial in Definition 2. Then for any $\varphi \sim \mathcal{D}$, $\mathsf{rank}_{\varphi}(g) = 2^{n/2}$.*

3 A variable-balanced decomposition for syntactic multilinear ABPs

In this section, we give a new decomposition for smABPs. The decomposition can be seen as a variable-balanced version of the well known decomposition of arithmetic circuits given by Valiant et al. [24] for the case of smABPs. In fact, we show that a smABP can be divided into sub-programs that are almost balanced in terms of the number of variables.

▶ **Theorem 1.** *Let P be an smABP of size S computing $f \in \mathbb{F}[x_1, \ldots, x_n]$. There exists edges $\{(u_1, v_1), \ldots, (u_m, v_m)\}$ and $\{(w_1, a_1), \ldots, (w_r, a_r)\}$ in P such that*

(1) *For $i \in [m]$, $n/3 \leq |X_{s,u_i}| \leq 2n/3$; and*

(2) *For $i \in [r]$, $|X_{s,w_i}| < n/3$ and $|X_{a_i,t}| \leq 2n/3$; and*

(3) $f = \sum_{i=1}^{m} [s, u_i] \cdot \textsf{label}(u_i, v_i) \cdot [v_i, t] + \sum_{i=1}^{r} [s, w_i] \cdot \textsf{label}(w_i, a_i) \cdot [a_i, t]$.

Proof. The proof is by a careful subdivision of the program P. We assume without loss of generality that t is reachable from every node in P and that every node in P has in-degree and out-degree at most 2. Consider the following coloring procedure:

(1) Initialize by coloring t as blue. Repeat (2) until no new node is colored.

(2) Consider node u that is colored blue such that at least one of nodes v or w are uncolored, where (v, u) and (w, u) are the only edges incoming to u. For $a \in \{v, w\}$ do following :

   a. If $|X_{s,a}| > 2n/3$, then color a as blue.

   b. If $n/3 \leq |X_{s,a}| \leq 2n/3$, then color a as red.

   c. If $|X_{s,a}| < n/3$, then color a as green.

At the end of the above coloring procedure we have the following:

1. For every node u with incoming edges (v, u) and (w, u), if u is colored blue then both v and w are colored.
2. For every directed $s \rightsquigarrow t$ path ρ in P, exactly one of the following holds:
   a. ρ has exactly one edge (v, w) such that v is colored red and w is colored blue.
   b. ρ has exactly one edge (v, w) such that v is colored green and w is colored blue.
3. If a node u is colored blue, then every node v reachable from u must have color blue.

Property 1 follows from the fact that a node v is colored if and only if there is an edge (v, u) such that u is colored blue. For property 3, clearly, a node u is colored blue if and only if $|X_{s,u}| > 2n/3$, thus every node reachable from a blue node is also colored blue. For property 2, let ρ be a directed $s \rightsquigarrow t$ path and v be the first node along ρ that is colored blue. Note $|X_{s,s}| = 0$, so s cannot be colored blue. Clearly, every node that follows v in ρ is colored blue and $v \neq s$. Let u be the node that immediately precedes v in ρ, then clearly, u is either red or green. Uniqueness follows from the fact that no node that precedes u in ρ is colored blue and every node that succeeds v in ρ is colored blue, hence there cannot be another such edge.

Let $E_{rb} = \{(u, v) \in P \mid u \text{ is colored red and } v \text{ is colored blue}\}$ and $E_{gb} = \{(u, v) \in P \mid u \text{ is colored green and } v \text{ is colored blue}\}$. Let $E_{rb} = \{(u_1, v_1), \ldots, (u_m, v_m)\}$ and $E_{gb} = \{(w_1, a_1), \ldots, (w_r, a_r)\}$ where $m, r \leq 2S$. We now prove that sets E_{rb} and E_{gb} satisfy the required properties.

(1) For $i \in [m]$, since $(u_i, v_i) \in E_{rb}$, u_i is colored red. By Step 2(b) of the coloring procedure, $n/3 \leq |X_{s,u_i}| \leq 2n/3$.

(2) For $i \in [r]$, since $(w_i, a_i) \in E_{gb}$, w_i is colored green and a_i is colored blue. By Step 2(c) of the coloring procedure, $|X_{s,w_i}| < n/3$ and by Step 2(a), $|X_{s,a_i}| > 2n/3$. Since P is syntactic multilinear, $|X_{s,a_i}| + |X_{a_i,t}| \leq n$ implying $|X_{a_i,t}| \leq n/3$.

(3) By Property 2, $s \rightsquigarrow t$ paths in P are partitioned into paths that have exactly one edge in E_{rb} and paths that have exactly one edge in E_{gb}. Therefore,

$$f = \sum_{\rho: s \rightsquigarrow t} \mathsf{wt}(\rho) = \sum_{\rho: s \rightsquigarrow t,\ \rho \cap E_{rb} \neq \emptyset} \mathsf{wt}(\rho) + \sum_{\rho: s \rightsquigarrow t,\ \rho \cap E_{gb} \neq \emptyset} \mathsf{wt}(\rho)$$
$$= \sum_{i=1}^{m} [s, u_i] \cdot \mathsf{label}(u_i, v_i) \cdot [v_i, t] + \sum_{i=1}^{r} [s, w_i] \cdot \mathsf{label}(w_i, a_i) \cdot [a_i, t]. \qquad \blacktriangleleft$$

The above decomposition allows us to obtain low-depth formulas for syntactic multilinear ABPs with quasi-polynomial blow-up in size. In the following, we show that a syntactic multilinear ABP can be computed by a log-depth syntactic multilinear formula where each leaf represents a multilinear polynomial in $O(\sqrt{n})$ variables.

▶ **Lemma 7.** *Let P be a syntactic multilinear ABP of size S computing a multilinear polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$. Then, there is a syntactic multilinear formula Φ with gates of unbounded fan-in computing f of size $S^{O(\log n)}$ and depth $O(\log n)$ such that every leaf w in Φ represents a multilinear polynomial $[u, v]_{P_w}$ for some nodes u, v in P_w with $|X_{u,v}| \leq \sqrt{n}$, where P_w is a sub-program of P. Further, any parse tree of Φ has at most $3\sqrt{n}$ leaves.*

Proof. Let nodes s and t be source and sink of P. The proof constructs a formula Φ by induction on the number of variables $|X_{s,t}|$ in the program P.
Base Case : If $|X_{s,t}| \leq \sqrt{n}$, then $\phi_{s,t}$ is a leaf gate with label $[s, t]$.
Induction Step : For induction step, suppose $|X_{s,t}| > \sqrt{n}$. By Theorem 1, we have

$$f = \sum_{i=1}^{m} [s, u_i] \cdot \mathsf{label}(u_i, v_i) \cdot [v_i, t] + \sum_{i=1}^{r} [s, w_i] \cdot \mathsf{label}(w_i, a_i) \cdot [a_i, t], \qquad (1)$$

where u_i, v_i, w_i and a_i are nodes in P, with $|X_{s,t}|/3 \leq |X_{s,u_i}| \leq 2|X_{s,t}|/3$ and $|X_{s,w_i}| + |X_{a_i,t}| \leq 2|X_{s,t}|/3$. Further, $[s, u_i] \cdot \mathsf{label}(u_i, v_i)$ (resp., $[s, w_i] \cdot \mathsf{label}(w_i, a_i)$) is an smABP with at most $2|X_{s,t}|/3 + 1$ (resp., $|X_{s,t}|/3$) variables. Let

$$f = \sum_{i=1}^{m} g_i h_i + \sum_{i=1}^{r} g'_i h'_i, \qquad (2)$$

where $g_i = [s, u_i] \cdot \mathsf{label}(u_i, v_i)$, $h_i = [v_i, t]$, $g'_i = [s, w_i] \cdot \mathsf{label}(w_i, a_i)$ and $h'_i = [a_i, t]$. For any i, if $|X_{s,w_i}| + |X_{a_i,t}| < \sqrt{n}$, then we set $g'_i = [s, w_i] \cdot \mathsf{label}(w_i, a_i) \cdot [a_i, t]$ and $h'_i = 1$. By induction, suppose ϕ_i (resp. ϕ'_i) be the multilinear formula that computes g_i (resp. g'_i) and ψ_i (resp. ψ'_i) be that for h_i (resp. h'_i). Set $\Phi = \sum_{i=1}^{m} (\phi_i \times \psi_i) + \sum_{i=1}^{r} (\phi'_i \times \psi'_i)$. Let $T(n)$ denote the size of the resulting formula on n variables. Then, $T(n) \leq 2 \cdot S \cdot 2 \cdot T(2n/3) = S^{O(\log n)}$. Thus, Φ is a syntactic multilinear formula of size $S^{O(\log n)}$ and depth $O(\log n)$ computing f. By construction, every leaf represents a multilinear polynomial $[u, v]_P$ for some nodes u, v in P with $|X_{u,v}| \leq \sqrt{n}$.

It remains to prove that any parse tree of Φ has at most $3\sqrt{n}$ leaves. We begin with a description of the process for constructing parse sub-trees of Φ. By Equation (2), constructing a parse tree of Φ is equivalent to the process:

1. Choose $b \in \{0, 1\}$ (corresponds to choosing one of the summations in Equation (2)).
2. If $b = 0$ choose $i \in \{1, \ldots, m\}$, else if $b = 1$ choose $j \in \{1, \ldots, r\}$.
3. Repeat steps 1 and 2 for sub-formulas ϕ_i, ϕ'_i, ψ_i and ψ'_i (depending on choice of b).

Consider any parse tree T of Φ. It is enough to prove that every leaf in T that is not labeled by 1 is a polynomial in at least $\sqrt{n}/3$ variables. Since Φ is syntactic multilinear, it follows that any parse tree of Φ has at most $3\sqrt{n}$ leaves, as required. However, it may be noted that this may not be true always. Instead, we argue that every leaf in T can be associated with a set of at least $\sqrt{n}/3$ variables such that no other leaf in T can be associated with these variables, hence implying that the number of leaves in any parse tree T of Φ is at most $3\sqrt{n}$.

Consider a leaf v in T having less than $\sqrt{n}/3$ variables. Let u be the first sum gate on the path from v to root with $|X_u| > \sqrt{n}$. Note that such a node u exists always, excluding the cases when the smABP P is a just a product of variable disjoint ABPs. Rest of the argument is split based on whether $b = 0$ or $b = 1$ at the step for choosing v in the construction of parse tree T. For the remainder of this proof, for any gate u, let X_u denote the set of variables in the sub-formula rooted at u in Φ.

Firstly, suppose that in the construction of T, $b = 0$ at the step for choosing v. Then, either $v = [p, u_i]\cdot\mathsf{label}(u_i, v_i)$ or $v = [v_i, q]$ for some nodes p, q, u_i, v_i in P, where u_i (respectively v_i) is colored red (respectively blue) when the coloring procedure (described in the proof of Theorem 1) is performed on the sub-program with source p and sink q. If $v = [p, u_i] \cdot \mathsf{label}(u_i, v_i)$, $|X_v| \geq |X_{p,u_i}| \geq |X_u|/3 \geq \sqrt{n}/3$, a contradiction to fact that v is a leaf in T with fewer than $\sqrt{n}/3$ variables. Hence, $v = [v_i, q]$. Set $A(v) = X_u \setminus (X_{p,u_i} \cup \{\mathsf{label}(u_i, v_i)\})$. Clearly, as $|X_u| \geq \sqrt{n}$ and $|X_{p,u_i}| \leq 2|X_u|/3$, we have $|A(v)| \geq \sqrt{n}/3$.

When $b = 1$, we have the following possibilities:

Case 1 $v = [p, w_i] \cdot \mathsf{label}(w_i, a_i) \cdot [a_i, q]$. In this case, set $A(v) = X_u$. Then $|A(v)| \geq \sqrt{n}/3$.

Case 2 $v = [p, w_i] \cdot \mathsf{label}(w_i, a_i)$. In this case, set $A(v) = X_u \setminus X_{a_i,q}$. Then $|A(v)| = |X_u| - |X_{a_i,q}| \geq \sqrt{n}/3$ as $|X_{a_i,q}| \leq 2|X_u|/3$ and $|X_u| > \sqrt{n}$.

Case 3 $v = [a_i, q]$. Set $A(v) = X_u \setminus (X_{p,w_i} \cup \{\mathsf{label}(w_i, a_i)\})$. Then $|A(v)| = |X_u| - |X_{p,w_i}| \geq \sqrt{n}/3$ as $|X_{p,w_i}| \leq 2|X_u|/3$ and $|X_u| > \sqrt{n}$.

It remains to prove that, for any two distinct leaves v and v' in T such that $A(v)$ and $A(v')$ are defined, $A(v) \cap A(v') = \emptyset$. Let u and u' respectively be parents of v and v' in T.

When $u = u'$, there are four possibilities for v and v': $v = [p, w_i] \cdot \mathsf{label}(w_i, a_i) \cdot [a_i, q]$ and $v' = 1$, $v = 1$ and $v' = [p, w_i] \cdot \mathsf{label}(w_i, a_i) \cdot [a_i, q]$, $v = [p, w_i] \cdot \mathsf{label}(w_i, a_i)$ and $v' = [a_i, q]$, or $v' = [p, w_i] \cdot \mathsf{label}(w_i, a_i)$ and $v = [a_i, q]$. As $A(v)$ is defined only for non-constant leaves, the only case is when $v = [p, w_i] \cdot \mathsf{label}(w_i, a_i), v' = [a_i, q]$ or vice-versa. In either of the cases, we have $A(v) \cap A(v') = \emptyset$. Now suppose, $u \neq u'$ and $A(v) \cap A(v') \neq \emptyset$. Then, we have $X_u \cap X_{u'} \neq \emptyset$ as $A(v) \subseteq X_u$ and $A(v') \subseteq X_{u'}$. From the fact that u and u' appear in the same parse tree we can conclude that the least common ancestor of u and u' in Φ must be a $\times$ gate. Let $[p, q]$ and $[p'q']$ be the sub-programs of P that correspond to u and u' respectively. By the construction of Φ, we can conclude that either there is a path from q to p' or there is a path from q' to p in P. Either of the cases is a contradiction to the fact that P is syntactic multilinear. ◀

Now, we obtain a reduction to depth-4 formulas for syntactic multilinear ABPs. Denote by $\Sigma^{[T]}\Pi^{[d]}(\Sigma\Pi)^{[r]}$ the class $\Sigma_{i=1}^{T}\Pi_{j=1}^{O(d)}Q_{ij}$ where Q_{ij}'s are mulitlinear polynomials in $O(r)$ variables. As a corollary to Lemma 7, we have the following reduction to syntactic multilinear $\Sigma\Pi^{[\sqrt{n}]}(\Sigma\Pi)^{[\sqrt{n}]}$ formulas for smABPs.

▶ **Corollary 1.** *Let P be a syntactic multilinear ABP of size S computing a polynomial f in $\mathbb{F}[x_1, \ldots, x_n]$. Then there exists a $\Sigma\Pi^{[\sqrt{n}]}(\Sigma\Pi)^{[\sqrt{n}]}$ syntactic multilinear formula of size $2^{O(\sqrt{n}\log S)}$ computing f.*

Proof. Let P be a syntactic multilinear ABP of size S computing a multilinear polynomial f in $\mathbb{F}[x_1, \ldots, x_n]$ and Φ be the equivalent syntactic multilinear formula with the properties mentioned in Lemma 7. The leaves of any parse tree of Φ represents the multilinear polynomial $[u,v]_{P_w}$ for some nodes u,v in P_w, a sub-program of P. As Φ is a formula, every parse tree T of Φ is uniquely identified by the set of $3\sqrt{n}$ leaves in T where every leaf in Φ represents a multilinear polynomial computed by some sub-program $[u,v]_{P_w}$ for some nodes u,v in P_w, a sub-program of P. Hence, it suffices to count the number of sub-programs of P. As every sub-program $([u,v])$ in P is obtained by choosing two vertices u and v in P, there are at most $\binom{S}{2}$ sub-programs of P. The number of parse trees of Φ are $S^{O(\sqrt{n})} = 2^{O(\sqrt{n}\log S)}$. As $f = \sum_{T:\text{parse tree of }\Phi} m(T)$ where $m(T)$ is the product of multilinear polynomials corresponding to the leaves of Φ in T, there exists a $\Sigma\Pi^{[\sqrt{n}]}(\Sigma\Pi)^{[\sqrt{n}]}$ syntactic multilinear formula with gates of unbounded fan-in of size $2^{O(\sqrt{n}\log S)}$ computing f. ◀

4 Lower Bounds for special classes of smABPs

This section is devoted to lower bounds for restricted classes of smABPs. Our arguments rely on the depth reduction proved in Section 3 and the full rank polynomial given by Raz and Yehhudayoff [19] (see Defintion 2).

4.1 Lower Bounds for strict circular-interval ABPs

In this section, we prove an exponential size lower bound for a special class of smABPs that we call as strict circular-interval ABPs.

An set $I \subseteq \{1,\ldots,n\}$ is a circular π-interval if $I = \{\pi(i),\pi(i+1),\ldots,\pi(j)\}$ for some $i,j \in [n], i < j$ or $I = \{\pi(i),\pi(i+1),\ldots,\pi(n),\pi(1),\ldots,\pi(j)\}$ for some $i,j \in [n], i > j$. These intervals are called *circular intervals* as every such interval $[i,j]$ in $\{1,\ldots,n\}$ can be viewed as a chord on the circle containing n points. Two circular intervals I and J are said to be *overlapping* if the corresponding chords in the circle intersect and *non-overlapping* otherwise. We define a special class of syntactic multilinear ABPs where the set of variables involved in every sub-program is in some circular π-interval.

▶ **Definition 3** (Strict circular-interval ABP). *Let $\pi \in S_n$ be a permutation. A syntactic mulitlinear ABP P is said to be a* strict π-circular-interval ABP *if*

1. *For any pair of nodes u,v in P, the index set of $X_{u,v}$ is contained in some circular π-interval I_{uv} in $[1,n]$; and*
2. *For any u,a,v in P, the circular π-intervals I_{ua} and I_{av} are non-overlapping.*

P is said to be strict circular-interval ABP if it is a strict π-circular-interval ABP for some permutation π.

We require a few preliminaries to prove the lower bound:

1. For every permutation π in S_n, define the partition $\varphi_\pi : X \to Y \cup Z$ such that
$$\text{for all } 1 \le i \le n/2 \ \varphi(x_{\pi(i)}) = y_i \text{ and } \varphi(x_{\pi(n/2+i)}) = z_i. \tag{3}$$
2. For any π in S_n, $|\varphi_\pi(X) \cap Y| = |\varphi_\pi(X) \cap Z| = |X|/2$. For the polynomial g in Definition 2, $\mathsf{rank}_{\varphi_\pi}(g) = 2^{n/2}$ by Lemma 6.
3. For any set $X_i \subseteq X$, let $\varphi_\pi(X_i) = \{\varphi_\pi(x) \mid x \in X_i\}$. We say X_i is *monochromatic* if either $\varphi_\pi(X_i) \cap Y = \emptyset$ or $\varphi_\pi(X_i) \cap Z = \emptyset$. Observe that if X_i is monochromatic then for any polynomial $p_i \in \mathbb{F}[X_i]$, we have $\mathsf{rank}_{\varphi_\pi}(p_i) \le 1$. Further, we say set $X_i \subseteq X$ is *bi-chromatic* if $\varphi_\pi(X_i) \cap Y \neq \emptyset$ and $\varphi_\pi(X_i) \cap Z \neq \emptyset$.

In the following theorem, we show that for any strict circular-interval ABP P computing a polynomial f, there is a partition φ such that $\mathsf{rank}_\varphi(f)$ is small.

▶ **Theorem 8.** *Let P be a strict circular-interval ABP of size S computing f in $\mathbb{F}[x_1, \ldots, x_n]$. There exists a $\varphi : X \to Y \cup Z$ with $|\varphi(X) \cap Y| = |\varphi(X) \cap Z| = |X|/2$ such that $\mathsf{rank}_\varphi(f) \leq 2^{\sqrt{n} \log n \log S + \sqrt{n}}$.*

Proof. Let Φ be the syntactic multilinear formula constructed from P as given by Lemma 7. Note that any parse tree of Φ has at most $3\sqrt{n}$ leaves. The number of parse trees of Φ is at most $\binom{2^{O(\sqrt{n} \log n \log S)}}{3\sqrt{n}} \leq 2^{\epsilon \sqrt{n} \log n \log S}$. Let T be any parse tree of Φ with leaves $w_1, \ldots, w_\ell$ computing polynomials $p_1, \ldots, p_\ell$. We have $f = \sum_{T:\text{parse tree of }\Phi} m(T)$ where $m(T)$ be the product of multilinear polynomials corresponding to the leaves of Φ in T. Let $X_1, \ldots, X_\ell \subseteq X$ be such that p_i is a polynomial in $\mathbb{F}[X_i]$. For every $i \in [\ell]$, let $M_i = \{j \mid x_j \in X_i\}$ be the index set of X_i. As P is a strict circular-interval ABP, we have that sets $M_1, \ldots, M_\ell$ are circular π-intervals in $\{1, \ldots, n\}$ for some $\pi \in S_n$. Let $\varphi_\pi : X \to Y \cup Z$ be the partition function described in Equation (3). If X_i is bi-chromatic then $\mathsf{rank}_{\varphi_\pi}(p_i) \leq 2^{\sqrt{n}/2}$ as $|X_i| \leq \sqrt{n}$ by construction of formula Φ when w_i is a leaf in Φ.

A crucial observation is that for any parse tree T of Φ, at most two of $\varphi_\pi(X_1), \ldots, \varphi_\pi(X_\ell)$ are bi-chromatic. This is because the existence of bi-chromatic sets $\varphi_\pi(X_i), \varphi_\pi(X_j), \varphi_\pi(X_k)$ for some $i, j, k \in [\ell]$ implies that the circular π-intervals M_i, M_j, M_k are overlapping from the way partition φ_π is defined. As X_i, X_j, X_k are variable sets associated with leaves of the same parse tree T, we can conclude that when $\varphi_\pi(X_i), \varphi_\pi(X_j), \varphi_\pi(X_k)$ are bi-chromatic there exists nodes u, a, v in P such that circular π-intervals I_{ua} and I_{av} are overlapping, a contradiction to the fact that P is a strict circular-interval ABP.

Therefore, in any parse tree T of Φ, at most two of $\varphi_\pi(X_1), \ldots, \varphi_\pi(X_\ell)$ are bi-chromatic say $\varphi_\pi(X_i)$ and $\varphi_\pi(X_j)$. Hence $\mathsf{rank}_{\varphi_\pi}(p_i) \leq 2^{\sqrt{n}/2}$ and $\mathsf{rank}_{\varphi_\pi}(p_j) \leq 2^{\sqrt{n}/2}$. Also, $\mathsf{rank}_{\varphi_\pi}(p_k) \leq 1$ for all $k \neq i, j$. Thus, $\mathsf{rank}_{\varphi_\pi}(f) \leq 2^{\epsilon \log n \log S \sqrt{n} + \sqrt{n}}$. ◀

With the above, we can prove Theorem 2:

▶ **Theorem 2.** *There exists an explicit multilinear polynomial g in $\mathbb{F}[x_1, \ldots, x_n]$ such that any strict circular-interval ABP computing g requires size $2^{\Omega(\sqrt{n}/\log n)}$.*

Proof. Let P be a strict circular-interval ABP of size $S = 2^{o(\sqrt{n}/\log n)}$ computing g and Φ be the syntactic multilinear formula obtained from P using Lemma 7. By Theorem 8, there exists a partition $\varphi : X \to Y \cup Z$ with $|\varphi(X) \cap Y| = |\varphi(X) \cap Z| = |X|/2$ such that $\mathsf{rank}_\varphi(g) \leq 2^{\sqrt{n} + \epsilon \log n \log S \sqrt{n}} < 2^{n/2}$. However, by Lemma 6, $\mathsf{rank}_\varphi(g) = 2^{n/2}$, a contradiction. Hence, $S = 2^{\Omega(\sqrt{n}/\log n)}$. ◀

4.2 Lower bound for sum of $\mathcal{L}$-ordered ABPs

In this section, we show that by observing a simple property of the ABP to formula conversion given in Lemma 7, we can obtain lower bounds for $\mathcal{L}$-ordered ABPs for larger sub-exponential values $\mathcal{L}$. In the following lemma, we observe that in the formula obtained from an $\mathcal{L}$-ordered ABP using Lemma 7, a lot of the leaves in any parse tree are in fact 1-ordered ABPs:

▶ **Lemma 9.** *Let P be an $\mathcal{L}$-ordered ABP and F be the syntactic multilinear formula obtained from P using Lemma 7. Then, for any parse tree T of F, all but at most $O(\log \mathcal{L})$ many leaves of T are 1-ordered ABPs (ROABPs).*

Proof. Let T be any parse tree of F with leaves $w_1, \ldots, w_\ell$ and let $p_1, \ldots, p_\ell$ be the polynomials labeling $w_1, \ldots, w_\ell$. From the construction given in the proof of Lemma 7, corresponding to each leaf w_i there are nodes u_i, v_i in P such that polynomial $p_i = [u_i, v_i] \cdot \mathsf{label}(v_i, u_{i+1})$. Consider the syntactic multilinear ABP P' obtained by placing programs

$$[u_1, v_1] \cdot \mathsf{label}(v_1, u_2), [u_2, v_2] \cdot \mathsf{label}(v_2, u_3), \ldots, [u_i, v_i] \cdot \mathsf{label}(v_i, u_{i+1}), \ldots, [u_\ell, v_\ell]$$

in the above order. From the construction above, P' is a sub-program of P and hence the number of variable orders in P' is a lower bound on the number of variable orders in P. If r_i is the number of variable orders in the sub-program $[u_i, v_i]$, the total number of variable orders in the sub-program P' (and hence P) is at least $r_1 \cdot r_2 \cdots r_\ell$. Since the number of distinct orders is at most $\mathcal{L}$, we conclude that $|\{i \mid r_i \geq 2\}| \leq \log \mathcal{L}$, as required. ◀

Let $\mathcal{D}$ denote the uniform distribution on the set of all partitions $\varphi : X \to Y \cup Z$ with $|Y| = |Z|$. In the following lemma, we show that rank of a polynomial computed by an $\mathcal{L}$-ordered ABP is far from being full. Proof can be found in the full version of the article [17].

▶ **Lemma 10.** *Let P be an $\mathcal{L}$-ordered ABP of size S computing a polynomial f in $\mathbb{F}[x_1, \ldots, x_n]$. Then for $k = n^{1/20}$, $\Pr_{\varphi \sim \mathcal{D}}[\mathsf{rank}_\varphi(f) > 2^{\log n \log S \sqrt{n}} \cdot 2^{n/2 - k\sqrt{n}}] \leq S^2 \cdot 2^{-O(n^{1/20})}$.*

Now, we are ready to prove Theorem 3:

▶ **Theorem 3.** *Let $\mathcal{L} \leq 2^{n^{1/2-\epsilon}}, \epsilon > 0$ and $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ be multilinear polynomials computed by $\mathcal{L}$-ordered ABPs of size $S_1, \ldots, S_m$ respectively. There exists an explicit multilinear polynomial g in $\mathbb{F}[x_1, \ldots, x_n]$ such that if $g = f_1 + \cdots + f_m$ then either $m = 2^{\Omega(n^{1/40})}$ or there is an $i \in [m]$ such that $S_i = 2^{\Omega(n^{1/40})}$.*

Proof. Set $k = n^{1/20}$. Suppose, for every i, f_i is computed by $\mathcal{L}$-ordered ABP of size $2^{n^{1/40}}$. Then, $\mathsf{rank}_\varphi(f_i) > 2^{\sqrt{n} \log n \log(2^{n^{1/40}})} 2^{n/2 - k\sqrt{n}}$ with probability at most $2^{2n^{1/40}} 2^{-n^{1/20}}$ when $\varphi \sim \mathcal{D}$. Therefore, probability that there is a i such that $\mathsf{rank}_\varphi(f_i) > 2^{\sqrt{n} \log n \log S} 2^{n/2 - k\sqrt{n}}$ is at most $m 2^{2n^{1/40}} 2^{-n^{1/20}} < 1$ for $m < 2^{n^{1/40}}$. By union bound, there is a $\varphi \sim \mathcal{D}$ such that for every i, $\mathsf{rank}_\varphi(f_i) < 2^{\sqrt{n} \log n \log(2^{n^{1/40}})} 2^{n/2 - k\sqrt{n}} < 2^{n/2}$. But by Lemma 6, $\mathsf{rank}_\varphi(g) = 2^{n/2}$ for every partition φ, which is a contradiction. Hence, either $m = 2^{\Omega(n^{1/40})}$ or there is an $i \in [m]$ such that $S_i = 2^{\Omega(n^{1/40})}$. ◀

5 Comparison with other multilinear circuit models

In this section, we compare strict circular-interval ABPs and $\mathcal{L}$-ordered ABPs to other well known models for computing mulitlinear polynomials.

5.1 $\mathcal{L}$-ordered to $\mathcal{L}$-pass

In this section, we show that $\mathcal{L}$-ordered ABPs can be transformed into ABPs that make at most $\mathcal{L}$ passes on the input, although in different orders.

▶ **Theorem 11.** *Let P be an $\mathcal{L}$-ordered ABP of size S computing a polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$. Then there is an $\mathcal{L}$-pass ABP Q of size $\mathsf{poly}(\mathcal{L}, S)$ computing f.*

Proof. Let P be an $\mathcal{L}$-ordered ABP of size S computing a polynomial f. Let $L_0, L_1, \ldots, L_\ell$ be the layers of P where source s and sink t are the only nodes in layers L_0 and L_ℓ respectively. Let $u_{i1}, \ldots, u_{iw}$ be nodes in L_i, where $w \leq S$ is the width of P. Without loss of generality assume every node in P has in-degree and out-degree at most two, and every layer except L_0 and L_ℓ has exactly w nodes. Also, every s to t path in P respects one of the permutations $\pi_1, \pi_2, \ldots, \pi_{\mathcal{L}}$. We now construct an $\mathcal{L}$-pass ABP $\mathcal{Q}$ that reads variables in the order $(x_{\pi_1(1)}, x_{\pi_1(2)}, \ldots, x_{\pi_1(n)}), \ldots, (x_{\pi_{\mathcal{L}}(1)}, x_{\pi_{\mathcal{L}}(2)}, \ldots, x_{\pi_{\mathcal{L}}(n)})$. The source and sink of ABP $\mathcal{Q}$ are denoted by s' and t' respectively. The number of layers in $\mathcal{Q}$ will be bounded by $\mathcal{L}(\ell + 1)$ and are labeled as $L_{ir}, i \in [\mathcal{L}], r \in \{0, \ldots, \ell\}$. Intuitively, for a node u_{rj} in layer L_r in P, we have $\mathcal{L}$ copies, $u_{1rj}, u_{2rj} \ldots, u_{\mathcal{L}rj}$ in $\mathcal{Q}$, where u_{irj} is a vertex in layer L_{ir}.

Intuitively, u_{irj} would have all paths from s to u_{rj} that respect the permutation π_i, but none of the permutations π_p for $p < i$. To ensure that the resulting ABP is $\mathcal{L}$-pass, we place the layers as follows : $L_{11}, \ldots, L_{1\ell}, L_{21}, \ldots, L_{2\ell}, \ldots, L_{\mathcal{L}1}, \ldots, L_{\mathcal{L}\ell}$. We construct $\mathcal{Q}$ as follows :

(1) Base Case : In ABP P, for every edge e from source s in layer L_0 to node $u_{1j}, j \leq w$ in layer L_1 labeled by $\mathsf{label}(e) \in X \cup \mathbb{F}$, if $\mathsf{label}(e) = x_k$, then add the edge (s', u_{m1j}) with label x_k where m is the smallest value such that x_k is consistent with π_m, if $\mathsf{label}(e) = \alpha \in \mathbb{F}$, then add the edge (s', u_{m1j}) with label α.

(2) Induction Step : Consider layer $L_r, r \in \{1, \ldots, \ell\}$:

a. For every node u_{rj} in layer L_r of P, with $1 \leq j \leq w$ and every edge e of the form $e = (u_{rj}, u_{r+1,j'})$ do the following:

Case 1: $\mathsf{label}(e) = x_k \in X$. For every $1 \leq i \leq \mathcal{L}$, let m be the smallest index such that every path from s' to u_{irj} concatenated with the edge e is consistent with π_m. Note that, by the construction, $m \geq i$. Add the edge $(u_{irj}, u_{mr+1j'})$ in $\mathcal{Q}$ for every i with label x_k. For every $1 \leq i \leq \mathcal{L}$, note that the choice of m is unique.

Case 2: $\mathsf{label}(e) = \alpha \in \mathbb{F}$. For every $1 \leq i \leq \mathcal{L}$, add edge (u_{irj}, u_{ir+1j}) with label α.

b. Create the node t' in $\mathcal{Q}$, and add edges $(u_{i\ell 1}, t')$ with label 1 for every $1 \leq i \leq \mathcal{L}$.

Note that in the above construction, the resulting branching program will not be layered. It can be made layered by adding suitable new vertices and edges labeled by $1 \in \mathbb{F}$. The correctness of the construction follows from the following claim whose proof is can be found in the full version [17].

▷ Claim 12.

(1) $\mathcal{Q}$ is an $\mathcal{L}$-pass syntactic multilinear ABP and has size $\mathsf{poly}(\mathcal{L}, S)$.

(2) For $1 \leq r \leq \ell$ and node u_{rj} in layer L_r in P, $1 \leq j \leq w$, $[s, u_{rj}]_P = \sum_{i=1}^{\mathcal{L}} [s', u_{irj}]_{\mathcal{Q}}$. ◀

5.2 Circular-Interval ABP vs. Sum of ROABPs

In this section, we define *circular-interval* ABPs (a generalization of strict circular-interval ABPs) and compare them to sum of ROABPs (and sum of strict circular-interval ABPs).

▶ **Definition 4** (Circular-Interval ABP). *Let $\pi \in S_n$ be a permutation. A syntactic mulitlinear ABP P is said to a* π-circular-interval ABP *if for any node v in P, the index set of $X_{s,v}$ is contained in some circular π-interval I_{sv} in $[1, n]$. P is said to be* circular-interval ABP *if it is a π-circular-interval ABP for some permutation π in S_n.*

Let $h = (h_n)_{n \geq 0}$ be the family of multilinear polynomials defined by Dvir et al. [8].

The following properties of the polynomial h are straightforward from Theorem 3.4 of [8] and the definition of circular-interval ABPs:

▶ **Lemma 13** ([8], Theorem 3.4). *(i) Over any field $\mathbb{F}$, the mulitlinear ABP R computing h is a circular-interval ABP of polynomial size. (ii) For any partition $\Pi \sim \mathcal{D}$, $\mathsf{rank}_\Pi(h) = 2^{n/2}$.*

Now, in order to separate circular-interval ABPs from ROABPs, it suffices to construct one partition Π such that $\mathsf{rank}_\Pi(f) < 2^{n/2}$ where f is the polynomial computed by an ROABP. This is guaranteed by the following lemma, whose proof is based on the ideas in [8] and [16]:

▶ **Lemma 14.** *Let Q be an ROABP computing a multilinear polynomial $f \in \mathbb{F}[x_1, \ldots, x_N]$ and Φ_Q be the multilinear formula obtained from Q computing f. Then $\mathsf{rank}_\Pi(f) \leq |\Phi_Q| \cdot 2^{n/2 - n^{1/5000}}$ with probability at least $1 - n^{\Omega(\log n)}$ for $\Pi \sim \mathcal{D}$.*

▶ **Theorem 15.** *Let $f_1, \dots f_m$ in $\mathbb{F}[x_1, \dots, x_n]$ be multilinear polynomials computed by ROABPs of size $R_1, \dots, R_m$ respectively. There exists an explicit multilinear polynomial h in $\mathbb{F}[x_1, \dots, x_n]$ computable by circular-interval ABPs of polynomial size, such that if $h = f_1 + \cdots + f_m$ then, either $m = n^{\Omega(\log n)}$ or there is an $i \in [m]$ with $R_i = 2^{\Omega\left(n^{1/6000}/\log N\right)}$.*

As an immediate corollary to Theorem 15, we have the following:

▶ **Corollary 16.** *There is a super-polynomial separation between ROABPs and circular interval ABPs.*

References

1 Manindra Agrawal and V. Vinay. Arithmetic Circuits: A Chasm at Depth Four. In *FOCS*, pages 67–75, 2008. doi:10.1109/FOCS.

2 Noga Alon, Mrinal Kumar, and Ben Lee Volk. Unbalancing Sets and an Almost Quadratic Lower Bound for Syntactically Multilinear Arithmetic Circuits. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, pages 11:1–11:16, 2018. doi:10.4230/LIPIcs.CCC.2018.11.

3 Matthew Anderson, Michael A. Forbes, Ramprasad Saptharishi, Amir Shpilka, and Ben Lee Volk. Identity Testing and Lower Bounds for Read-k Oblivious Algebraic Branching Programs. *TOCT*, 10(1):3:1–3:30, 2018. doi:10.1145/3170709.

4 Vikraman Arvind and S. Raja. Some Lower Bound Results for Set-Multilinear Arithmetic Computations. *Chicago J. Theor. Comput. Sci.*, 2016, 2016. URL: http://cjtcs.cs.uchicago.edu/articles/2016/6/contents.html.

5 Walter Baur and Volker Strassen. The Complexity of Partial Derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983. doi:10.1016/0304-3975(83)90110-X.

6 Suryajith Chillara, Christian Engels, Nutan Limaye, and Srikanth Srinivasan. A Near-Optimal Depth-Hierarchy Theorem for Small-Depth Multilinear Circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:62, 2018. URL: https://eccc.weizmann.ac.il/report/2018/062.

7 Suryajith Chillara, Nutan Limaye, and Srikanth Srinivasan. Small-depth Multilinear Formula Lower Bounds for Iterated Matrix Multiplication, with Applications. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 21:1–21:15, 2018. doi:10.4230/LIPIcs.STACS.2018.21.

8 Zeev Dvir, Guillaume Malod, Sylvain Perifel, and Amir Yehudayoff. Separating multilinear branching programs and formulas. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 615–624, 2012. doi:10.1145/2213977.2214034.

9 Michael A. Forbes, Ramprasad Saptharishi, and Amir Shpilka. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 867–875, 2014. doi:10.1145/2591796.2591816.

10 Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Identity Testing for Constant-Width, and Any-Order, Read-Once Oblivious Arithmetic Branching Programs. *Theory of Computing*, 13(1):1–21, 2017. doi:10.4086/toc.2017.v013a002.

11 Maurice J. Jansen. Lower Bounds for Syntactically Multilinear Algebraic Branching Programs. In *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, pages 407–418, 2008. doi:10.1007/978-3-540-85238-4_33.

12 Maurice J. Jansen, Youming Qiao, and Jayalal Sarma. Deterministic Black-Box Identity Testing π-Ordered Algebraic Branching Programs. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, pages 296–307, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.296.

13 Mrinal Kumar. A Quadratic Lower Bound for Homogeneous Algebraic Branching Programs. In Ryan O'Donnell, editor, *32nd Computational Complexity Conference (CCC 2017)*, volume 79 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CCC.2017.19.

14 Mrinal Kumar, Rafael Mendes de Oliveira, and Ramprasad Saptharishi. Towards Optimal Depth Reductions for Syntactically Multilinear Circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:19, 2019. URL: https://eccc.weizmann.ac.il/report/2019/019.

15 Noam Nisan. Lower Bounds for Non-Commutative Computation (Extended Abstract). In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 410–418, 1991. doi:10.1145/103418.103462.

16 C. Ramya and B. V. Raghavendra Rao. Lower Bounds for Special Cases of Syntactic Multilinear ABPs. In *Computing and Combinatorics - 24th International Conference, COCOON 2018, Qing Dao, China, July 2-4, 2018, Proceedings*, pages 701–712, 2018. doi:10.1007/978-3-319-94776-1_58.

17 C. Ramya and B. V. Raghavendra Rao. Lower bounds for multilinear bounded order ABPs. *CoRR*, abs/1901.04377, 2019. arXiv:1901.04377.

18 Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2), 2009. doi:10.1145/1502793.1502797.

19 Ran Raz and Amir Yehudayoff. Balancing Syntactically Multilinear Arithmetic Circuits. *Computational Complexity*, 17(4):515–535, 2008. doi:10.1007/s00037-008-0254-0.

20 Ran Raz and Amir Yehudayoff. Lower Bounds and Separations for Constant Depth Multilinear Circuits. *Computational Complexity*, 18(2):171–207, 2009. doi:10.1007/s00037-009-0270-8.

21 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity, 2017. URL: https://github.com/dasarpmar/lowerbounds-survey/releases.

22 Sébastien Tavenas. Improved Bounds for Reduction to Depth 4 and Depth 3. In *MFCS*, pages 813–824, 2013. doi:10.1007/978-3-642-40313-2_71.

23 Leslie G. Valiant. Completeness Classes in Algebra. In *STOC*, pages 249–261, 1979. doi:10.1145/800135.804419.

24 Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast Parallel Computation of Polynomials Using Few Processors. *SIAM J. Comput.*, 12(4):641–644, 1983. doi:10.1137/0212043.

On the Symmetries of and Equivalence Test for Design Polynomials

Nikhil Gupta
Department of Computer Science and Automation, Indian Institute of Science, India
nikhilg@iisc.ac.in

Chandan Saha
Department of Computer Science and Automation, Indian Institute of Science, India
chandan@iisc.ac.in

Abstract

In a Nisan-Wigderson design polynomial (in short, a design polynomial), every pair of monomials share a few common variables. A useful example of such a polynomial, introduced in [34], is the following:

$$\mathsf{NW}_{d,k}(\mathbf{x}) = \sum_{h \in \mathbb{F}_d[z],\ \deg(h) \le k} \ \prod_{i=0}^{d-1} x_{i,h(i)},$$

where d is a prime, $\mathbb{F}_d$ is the finite field with d elements, and $k \ll d$. The degree of the gcd of every pair of monomials in $\mathsf{NW}_{d,k}$ is at most k. For concreteness, we fix $k = \lceil \sqrt{d} \rceil$. The family of polynomials $\mathcal{NW} := \{\mathsf{NW}_{d,k} : d \text{ is a prime}\}$ and close variants of it have been used as hard explicit polynomial families in several recent arithmetic circuit lower bound proofs. But, unlike the permanent, very little is known about the various structural and algorithmic/complexity aspects of $\mathcal{NW}$ beyond the fact that $\mathcal{NW} \in \mathsf{VNP}$. Is $\mathsf{NW}_{d,k}$ characterized by its symmetries? Is it circuit-testable, i.e., given a circuit C can we check efficiently if C computes $\mathsf{NW}_{d,k}$? What is the complexity of equivalence test for $\mathcal{NW}$, i.e., given black-box access to a $f \in \mathbb{F}[\mathbf{x}]$, can we check efficiently if there exists an invertible linear transformation A such that $f = \mathsf{NW}_{d,k}(A \cdot \mathbf{x})$? Characterization of polynomials by their symmetries plays a central role in the geometric complexity theory program. Here, we answer the first two questions and partially answer the third.

We show that $\mathsf{NW}_{d,k}$ is characterized by its group of symmetries over $\mathbb{C}$, but not over $\mathbb{R}$. We also show that $\mathsf{NW}_{d,k}$ is characterized by circuit identities which implies that $\mathsf{NW}_{d,k}$ is circuit-testable in randomized polynomial time. As another application of this characterization, we obtain the "flip theorem" for $\mathcal{NW}$.

We give an efficient equivalence test for $\mathcal{NW}$ in the case where the transformation A is a block-diagonal permutation-scaling matrix. The design of this algorithm is facilitated by an almost complete understanding of the group of symmetries of $\mathsf{NW}_{d,k}$: We show that if A is in the group of symmetries of $\mathsf{NW}_{d,k}$ then $A = D \cdot P$, where D and P are diagonal and permutation matrices respectively. This is proved by completely characterizing the Lie algebra of $\mathsf{NW}_{d,k}$, and using an interplay between the Hessian of $\mathsf{NW}_{d,k}$ and the evaluation dimension.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Nisan-Wigderson design polynomial, characterization by symmetries, Lie algebra, group of symmetries, circuit testability, flip theorem, equivalence test

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.53

Acknowledgements We would like to thank Neeraj Kayal, Meena Mahajan for some insightful discussions on the design polynomial family. NG would also like to thank Anuj Tawari for his time in sitting through a few presentations on the proof of Theorem 4. We also thank anonymous reviewers for their comments.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 53; pp. 53:1–53:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Proving super-polynomial lower bounds for Boolean and arithmetic circuits computing explicit functions is the holy grail of circuit complexity. Over the past few decades, research on lower bounds has gradually pushed the frontier by bringing in novel methods in the arena and carefully building upon the older ones. Some of the notable achievements are – lower bounds for AC^0 circuits [2, 17, 26], monotone circuits [3, 57], $\mathsf{ACC}(p)$ circuits [58, 62] and ACC circuits [52, 63] in the Boolean case, and lower bounds for homogeneous depth three circuits [54], multilinear formulas [55, 56], homogeneous depth four circuits [23, 31, 42] and the lower bound on the depth of circuits for MaxFlow [46] in the arithmetic case. The slow progress in circuit lower bounds is explained by a few "barrier" type results, particularly by the notion of natural proofs [59] for Boolean circuits, and the notion of algebraically natural proofs [13, 21] for arithmetic circuits [1]. Most lower bound proofs, but not all [2], do fit in the natural proof framework.

It is apparent from the concept of natural proofs and its algebraic version that in order to avoid this barrier, we need to develop an approach that violates the so called *constructivity* criterion or the *largeness* criterion. Focusing on the latter criterion, it means, if an explicit function has a special property that random functions do not have, and if a lower bound proof for circuits computing this explicit function uses this special property critically, then such a proof circumvents the natural proof barrier automatically. For polynomial functions (simply polynomials), *characterization by symmetries* is such a special property[3], and the geometric complexity theory (GCT) program [51] is an approach to proving super-polynomial arithmetic circuit lower bound by crucially exploiting this property of the permanent and the determinant polynomials. From hereon, our discussion will be restricted to polynomial functions and arithmetic circuits.

The permanent family is complete for the class VNP and the determinant family is complete for the class VBP under p-projections. The class $\mathsf{VBP} \subseteq \mathsf{VP}$ consists of polynomial families that are computable by poly-size algebraic branching programs; this class has another interesting complete family, namely the iterated matrix multiplication (IMM) family. These three polynomial families have appeared in quite a few lower bound proofs [9, 15, 20, 23, 36, 42, 45, 54–56] in the arithmetic circuit literature. That permanent and determinant are characterized by their respective groups of symmetries are classical results [16, 44]. It has also been shown that IMM is characterized by its symmetries [19, 32]. There are two other polynomial families in VP, the power symmetric polynomials and the sum-product polynomials, that are known to possess this rare property (see Section 2 in [8]). However, the elementary symmetric polynomial is not characterized by its symmetries [27].

In the recent years, another polynomial, namely the Nisan-Wigderson design polynomial (in short, design polynomial), and close variants of it have been used intensely as hard explicit polynomials in several lower bound proofs for depth three, depth four and depth five circuits [10, 12, 31, 33–35, 37–42]. In some cases, the design polynomial (Definition 7) yielded lower bounds that are not known yet for the permanent, determinant and IMM (as

[1] Presently, the evidences in favor of existence of one-way functions (which implies the natural proof barrier) are much stronger than that of existence of succinct hitting-set generators (which implies the algebraically natural proof barrier). However, there are a few results in algebraic complexity that exhibit, unconditionally [11] or based on more plausible complexity theoretic assumptions [5], the limitations of some of the current techniques in proving lower bounds for certain restricted arithmetic models.

[2] like the lower bounds for monotone and ACC circuits

[3] A random polynomial is not characterized by its symmetries with high probability (see Proposition 3.4.9 in [22])

in [12,33,37,40]). It can be easily shown that the design polynomial defines a family in VNP (see Observation B.1 in [24]). But, very little is otherwise known about the various structural and algorithmic/complexity aspects of this family. Like the permanent, is it characterized by its symmetries? Is it circuit testable? What is the complexity of equivalence test for the Nisan-Wigderson design polynomial? It is reasonable to seek answers to these fundamental questions for a natural family like the design polynomials. Moreover, in the light of some recent developments in GCT [7,28,29], it may be worth studying other polynomial families (like the design polynomials and the IMM) that have some of the "nice features" of the permanent and the determinant and that may also fit in the GCT framework. We refer the reader to [1,22,50,60] for an overview of GCT. If the design polynomial family turns out to be in VP then that would be an interesting result by itself with potentially important complexity theoretic and algorithmic consequences. If a polynomial has a small depth-4 circuit, then it is a projection of a small NW design polynomial (see Observation B.2 in [24])

In this article, we answer some of the above questions on the design polynomial pertaining to its group of symmetries. Our results accord a fundamental status to this polynomial.

1.1 Our results

Some of the basic definitions and notations are given in Section 2. The design polynomial $\mathsf{NW}_{d,k}$ is defined (in Definition 7) using two parameters, d (the degree) and k (the "intersection" parameter). Our results hold for any $k \in [1, \frac{d}{4} - 5]$, but (from the lower bound point of view) it is best to think of k as d^{ϵ} for some arbitrarily chosen constant $\epsilon \in (0,1)$. The number of variables in $\mathsf{NW}_{d,k}$ is $n = d^2$. Any polynomial can be expressed as an affine projection of $\mathsf{NW}_{d,k}$, for a possibly large d (see Observation B.2 in [24]). For notational convenience, we will drop the subscripts d and k whenever they are clear from the context. Let $\mathscr{G}_f$ be the group of symmetries of a polynomial f over an underlying field $\mathbb{F}$ (see Definition 12).

▶ **Theorem 1** (Characterization by symmetries). *Let $\mathbb{F} = \mathbb{C}$ and f be a homogeneous degree-d polynomial in $n = d^2$ variables. If $\mathscr{G}_{\mathsf{NW}} \subseteq \mathscr{G}_f$ then $f = \alpha \cdot \mathsf{NW}$ for some $\alpha \in \mathbb{C}$.*

The theorem, proven in Section 3, holds over any field $\mathbb{F}$ having a d-th root of unity $\zeta \neq 1$ and $|\mathbb{F}| \neq d+1$. We also show in Section 4.3 that NW is not characterized by its symmetries over $\mathbb{R}, \mathbb{Q}$ and finite fields not containing a d-th primitive root of unity – in contrast, the permanent is characterized by its symmetries over these fields. The symmetries of NW have a nice algorithmic application: Although, it is not known if NW is computable by a poly(d) size circuit (Definition 6), the following theorem shows that checking if a given circuit computes NW can be done efficiently. In this article, whenever we mention size-s circuit, we mean size-s circuit with degree bounded by $\delta(s)$, which is an arbitrarily fixed polynomial function[4] of s. Let $\mathbf{x}$ be the set of n variables of NW. We will identify a circuit with the polynomial computed by it.

▶ **Theorem 2** (Circuit testability). *There is a randomized algorithm that takes input as black-box access to a circuit $\mathsf{C}(\mathbf{x})$ of size s over a finite field $\mathbb{F}$, where $|\mathbb{F}| \geq 4 \cdot \delta(s)$ (recall $\delta(s)$ is an upper bound on the degree of size s circuits), and determines correctly whether or not $\mathsf{C}(\mathbf{x}) = \mathsf{NW}$ with high probability, using* poly(s) *field operations.*

[4] This is the interesting scenario in algebraic complexity theory as polynomial families in VP admit circuits with degree bounded by a polynomial function of size.

A suitable version of the theorem also holds over $\mathbb{Q}, \mathbb{R}$ and $\mathbb{C}$. Such a theorem is known for the permanent with two different proofs, one using self-reducibility of the permanent [43] and the other using its symmetries [48]. We do not know if NW has a self-reducible property like the permanent, but its symmetries are powerful enough to imply the above result. The theorem is proven in Section 5 by showing that NW is characterized by circuit identities over *any* field (see Definition 18). This characterization, which uses the symmetries of NW, also implies the following result. For this result, we can assume $\delta(s) \geq d$, without any loss of generality.

▶ **Theorem 3** (Flip theorem). *Suppose* NW *is not computable by circuits of size s over a finite field $\mathbb{F}$, where $|\mathbb{F}| \geq 4 \cdot \delta(s)$ and $\delta(s)$ is an upper bound on the degree of size s circuits. Then, there exist points $\mathbf{a}_1, \ldots, \mathbf{a}_m \in \mathbb{F}^n$, where $m = \mathrm{poly}(s)$, such that for every circuit* C *over $\mathbb{F}$ of size at most s, there is an $\ell \in [m]$ satisfying $\mathsf{C}(\mathbf{a}_\ell) \neq \mathsf{NW}(\mathbf{a}_\ell)$. A set of randomly generated points $\mathbf{a}_1, \ldots, \mathbf{a}_m \in_r \mathbb{F}^n$ has this property with high probability. Moreover, black-box derandomization of polynomial identity testing for size-$(10s)$ circuits over $\mathbb{F}$ using $\mathrm{poly}(s)$ field operations implies that the above-mentioned points can be computed deterministically using $\mathrm{poly}(s)$ field operations.*

An appropriate version of the theorem also holds over $\mathbb{Q}, \mathbb{R}$ and $\mathbb{C}$. The flip theorem is known for the permanent [48, 49] [5]. Similar theorems have also been shown for the $3SAT$ problem [4, 14]. Results of this kind show that if a certain function ($3SAT$ or permanent or NW) is not computable by small circuits then there exists a short list of efficiently computable "hard instances" that fail all small circuits.
We show another algorithmic application of the knowledge of the symmetries of NW in solving a natural case of the equivalence test problem for NW, namely block-diagonal permutation-scaling equivalence test (BD-PS equivalence test, in short). An equivalence test for NW checks if a given polynomial $f \in \mathbb{F}[\mathbf{x}]$ satisfies $f = \mathsf{NW}(A \cdot \mathbf{x})$, where A is an invertible linear transformation. A BD-PS equivalence test is the special case where A is a product of a block-diagonal permutation matrix and an invertible scaling matrix. The following theorem is proved in Section 6.

▶ **Theorem 4** (BD-PS equivalence test for NW). *Let $k \in [1, \frac{d}{3}]$, $\mathbb{F}$ be a finite field such that $d \nmid (|\mathbb{F}| - 1)$ and $|\mathbb{F}| \geq 4d$. There is a randomized algorithm that takes input black-box access to a degree d polynomial $f \in \mathbb{F}[\mathbf{x}]$ and correctly decides if f is BD-PS equivalent to* NW *with high probability. If the answer is yes then it outputs a A such that $f = \mathsf{NW}(A \cdot \mathbf{x})$, where A is a product of a block-diagonal permutation matrix and an invertible scaling matrix. The running time is $\mathrm{poly}(d, \log |\mathbb{F}|)$.*

An appropriate version of the theorem holds over $\mathbb{R}$ (details given in Section F.4 of [24]). Efficient equivalence tests are known for the Permanent and IMM over $\mathbb{C}$, $\mathbb{Q}$ and finite fields [30, 32] and for the Determinant over $\mathbb{C}$ and finite fields [18, 30]. In [30], it was shown that equivalence test for the Permanent reduces to permutation-scaling (PS) equivalence test. We show in Section 6 that equivalence test for NW reduces to block-permuted equivalence test[6],i.e., we can assume without loss of generality that A is a block-permuted matrix. Theorem 4 solves the equivalence test for NW in the case where A is a block-diagonal matrix and additionally has the permutation-scaling (PS) structure. Even this case is quite nontrivial and may serve as an important ingredient for an efficient general equivalence test for NW.

[5] We have borrowed the name "flip theorem" from these work.
[6] It decides if there exists a block-permuted matrix (Definition 8) $A \in \mathrm{GL}_{d^2}(\mathbb{F})$ such that $f = \mathsf{NW}(A \cdot \mathbf{x})$

The design of the test in Theorem 4 is facilitated by a near complete understanding of the symmetries of NW as stated in the following theorem. The proof is given in Section 4.2.

▶ **Theorem 5** (Structure of $\mathscr{G}_{\mathsf{NW}}$)**.** *Let $\mathbb{F}$ be the underlying field of size greater than $\binom{d}{2}$ and* $\mathrm{char}(\mathbb{F}) \neq d$*. If $A \in \mathscr{G}_{\mathsf{NW}}$ then $A = D \cdot P$, where $D, P \in \mathscr{G}_{\mathsf{NW}}$ are diagonal and permutation matrices respectively.*

The group of symmetries of the permanent has a similar structure [44]. The above structure also plays a crucial role in showing that NW is not characterized by its symmetries over $\mathbb{R}$. The proof of the theorem involves a complete characterization of the Lie algebra of NW, and an interplay between the Hessian of NW and the evaluation dimension measure. We first prove the structural results (Theorems 1 and 5) and then show their algorithmic applications (Theorems 2, 3 and 4). The proof details are shifted to the appendix. A comparison between the Permanent and NW is summarized in a table in Section A of [24].

2 Preliminaries

Notations. The set of natural numbers is $\mathbb{N} = \{0, 1, 2 \ldots\}$ and $\mathbb{N}^{\times} = \mathbb{N}\backslash\{0\}$. For $r \in \mathbb{N}^{\times}$, $[r] = \{0, \ldots, r-1\}$. The general linear group $\mathrm{GL}_r(\mathbb{F})$ is the group of all $r \times r$ invertible matrices over $\mathbb{F}$. Throughout this article, $\mathrm{poly}(r)$ means $r^{O(1)}$ and $\exp(r)$ means 2^r. For a prime d, $\mathbb{F}_d$ is the finite field of order d whose elements are naturally identified with $[d] = \{0, 1, \ldots, d-1\}$. Let $\mathbf{x}$ be the following disjoint union of variables,

$$\mathbf{x} := \biguplus_{i \in [d]} \mathbf{x}_i, \tag{1}$$

where $\mathbf{x}_i := \{x_{i,0}, \ldots, x_{i,d-1}\}$. The total number of variables in $\mathbf{x}$ is $n = d^2$. $\mathbb{F}[\mathbf{x}]$ and $\mathbb{F}_d[z]$ denote the rings of multivariate and univariate polynomials over $\mathbb{F}$ and $\mathbb{F}_d$ in $\mathbf{x}$ and z variables respectively, and the set $\mathbb{F}_d[z]_k := \{h \in \mathbb{F}_d[z] : \deg(h) \leq k\}$. We will represent elements of $\mathbb{F}$ by lower case Greek alphabets $(\alpha, \beta, \ldots)$, elements of $\mathbb{F}_d$ by lower case Roman alphabets $(a, b, \ldots)$, multivariate polynomials over $\mathbb{F}$ by f, g and q, univariate polynomials over $\mathbb{F}_d$ by p and h, matrices over $\mathbb{F}$ by capital letters $(A, B, C, \ldots)$, and the set of variables by $\mathbf{x}, \mathbf{y}, \mathbf{z}$ and vectors over $\mathbb{F}$ by $\mathbf{a}, \mathbf{b}$. Variable sets are interpreted as column vectors when left multiplied to a matrix. For instance, in $A \cdot \mathbf{x}$, $\mathbf{x}$ is the vector $(x_{0,0}\ x_{0,1} \ldots\ x_{0,d-1}\ \ldots\ x_{d-1,0}\ x_{d-1,1} \ldots\ x_{d-1,d-1})^T$, and we say A is applied on $\mathbf{x}$.

2.1 Algebraic preliminaries

A polynomial f is homogeneous if the degree of all the monomials of f are the same. Polynomial $f \in \mathbb{F}[\mathbf{x}]$ is set-multilinear in the sets $\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}$ (as defined in Equation (1)) if every monomial contains exactly one variable from each set $\mathbf{x}_i$ for $i \in [d]$.

▶ **Definition 6** (Arithmetic circuit)**.** *An arithmetic circuit C over $\mathbb{F}$ is a directed acyclic graph in which a node with in-degree zero is labelled with either a variable or an $\mathbb{F}$-element, an edge is labelled with an $\mathbb{F}$-element, and other nodes are labelled with $+$ and $\times$. Computation proceeds in a natural way: a node with in-degree zero computes its label, an edge scales a polynomial by its label, and a node labelled with $+/\times$ computes the sum/product of the polynomials computed at the end of the edges entering the node. The polynomials computed by nodes with out-degree zero are the outputs of C. The size of C is the sum of the number of nodes and edges in the graph. The* degree *of C is the maximum over the degree of the polynomials computed at all nodes of C.*

▶ **Definition 7** (Nisan-Wigderson polynomial). *Let $d > 2$ be a prime and $k \in \mathbb{N}$. The Nisan-Wigderson design polynomial is defined as in [34] (which is inspired by the Nisan-Wigderson set-systems [53]),*

$$\mathsf{NW}_{d,k}(\mathbf{x}) := \sum_{h \in \mathbb{F}_d[z]_k} \prod_{i \in \mathbb{F}_d} x_{i,h(i)}.$$

It is a degree-d homogeneous and set-multilinear polynomial in $n = d^2$ variables, having d^{k+1} monomials. We drop the subscripts d, k for notational convenience. NW satisfies the "low intersection" property, meaning any two monomials of NW have at most k variables in common. This follows because the monomials are obtained from polynomials in $\mathbb{F}_d[z]_k$.

▶ **Definition 8** (Block-permuted matrix). *A matrix $A \in \mathbb{F}^{d^2 \times d^2}$ is a block-permuted matrix with block size d if $A = B \cdot (P \otimes I_d)$, where $B \in \mathbb{F}^{d^2 \times d^2}$ is a block-diagonal matrix with block size d, $P \in \mathbb{F}^{d \times d}$ is a permutation matrix, and I_d is the $d \times d$ identity matrix.*

▶ **Definition 9** (Evaluation dimension). *Let $f \in \mathbb{F}[\mathbf{y}]$ and $\mathbf{z} \subseteq \mathbf{y}$. The evaluation dimension of f with respect to $\mathbf{z}$ is, $evalDim_{\mathbf{z}}(f) := \dim(\ \mathbb{F}\text{-}span\ \{f(\mathbf{y})|_{\mathbf{z}=\mathbf{a}} : \boldsymbol{a} \in \mathbb{F}^{|\mathbf{z}|}\}\ )$.*

▶ **Definition 10** (Hessian). *Let $f \in \mathbb{F}[\mathbf{y}]$ be a polynomial in $\mathbf{y} = \{y_1, y_2, \ldots, y_n\}$ variables. The Hessian of f is the following matrix in $(\mathbb{F}[\mathbf{y}])^{n \times n}$,*

$$H_f(\mathbf{y}) := \Big(\frac{\partial^2 f}{\partial y_i \cdot \partial y_j}\Big)_{i,j \in [n]}.$$

The following property of $H_f(\mathbf{y})$ that can be proved using chain-rule of derivatives.

▶ **Lemma 11** (Lemma 2.6 of [8]). *Let $g \in \mathbb{F}[\mathbf{y}]$ and $f = g(A \cdot \mathbf{y})$ for some $A \in \mathbb{F}^{n \times n}$. Then, $H_f(\mathbf{y}) = A^T \cdot H_g(A \cdot \mathbf{y}) \cdot A$.*

▶ **Definition 12** (Group of symmetries). *Let $f \in \mathbb{F}[\mathbf{y}]$ be an n-variate polynomial. The set $\mathscr{G}_f = \{A \in \mathrm{GL}_n(\mathbb{F}) : f(A \cdot \mathbf{y}) = f(\mathbf{y})\}$ forms a group under matrix multiplication and it is called the group of symmetries of f over $\mathbb{F}$.*

▶ **Definition 13** (Lie algebra). *Let $f \in \mathbb{F}[\mathbf{y}]$ be a polynomial in $\mathbf{y} = \{y_1, y_2, \ldots, y_n\}$ variables. The Lie algebra of f, denoted by $\mathfrak{g}_f$, is the set of matrices $B = (b_{i,j})_{i,j \in [n]} \in \mathbb{F}^{n \times n}$ satisfying the relation $\sum_{i,j \in [n]} b_{i,j} \cdot y_j \cdot \frac{\partial f}{\partial y_i} = 0$.*

It is easy to check that $\mathfrak{g}_f$ is a vector space over $\mathbb{F}$. The following property relates the Lie algebras of $f(\mathbf{y})$ and $f(A \cdot \mathbf{y})$ for $A \in \mathrm{GL}_n(\mathbb{F})$. See Proposition 58 of [30] for its proof.

▶ **Lemma 14** (Conjugacy of Lie algebras). *Let $g \in \mathbb{F}[\mathbf{y}]$ be an n-variate polynomial. If $f(\mathbf{y}) = g(A \cdot \mathbf{y})$ for $A \in \mathrm{GL}_n(\mathbb{F})$, then $\mathfrak{g}_f = A^{-1} \cdot \mathfrak{g}_g \cdot A$.*

▶ **Lemma 15.** *[30] Given black-box access to an n-variate degree d polynomial $f \in \mathbb{F}[\mathbf{x}]$, a basis of $\mathfrak{g}_f$ can be computed in randomized $poly(n, d, \rho)$ time, where ρ is the bit complexity of the coefficients of f.*

Over $\mathbb{C}$, the Lie algebra $\mathfrak{g}_f$ is related to the group of symmetries $\mathscr{G}_f$ as stated in the following definition. For $B \in \mathbb{C}^{n \times n}$, let $e^B := \sum_{i \in \mathbb{N}} \frac{B^i}{i!} \in \mathbb{C}^{n \times n}$ (the series always converges).

▶ **Definition 16** (Continuous and discrete symmetries). *Let $f \in \mathbb{C}[\mathbf{y}]$. If $A \in \mathfrak{g}_f$ then $e^{tA} \in \mathscr{G}_f$ for every $t \in \mathbb{R}$ (see [25] for a proof of this fact). Elements of the set $\{e^{tA} : A \in \mathfrak{g}_f$ and $t \in \mathbb{R}\}$ are the continuous symmetries of f. All the other symmetries in $\mathscr{G}_f$ are the discrete symmetries of f.*

▶ **Definition 17** (Characterization by symmetries). *A homogeneous degree-d polynomial $g \in \mathbb{F}[\mathbf{y}]$ is said to be characterized by its symmetries if for every degree-d homogeneous polynomial $f \in \mathbb{F}[\mathbf{y}]$, $\mathscr{G}_g \subseteq \mathscr{G}_f$ implies that $f(\mathbf{y}) = \alpha \cdot g(\mathbf{y})$ for some $\alpha \in \mathbb{F}$.*

▶ **Definition 18** (Characterization by circuit identities). *Let $g \in \mathbb{F}[\mathbf{y}]$ be an n-variate polynomial, and $\mathbf{z}, \mathbf{u}$ be two sets of constantly many variables and $|\mathbf{z}| = c$. Suppose that there exist $m = \mathrm{poly}(n)$ polynomials $q_1(\mathbf{z}, \mathbf{u}), \ldots, q_m(\mathbf{z}, \mathbf{u})$ over $\mathbb{F}$ such that for every $i \in [m]$, q_i is computable by a constant size circuit and there exist $A_{i1}, \ldots, A_{ic} \in \mathbb{F}[\mathbf{u}]^{n \times n}$ computable by $\mathrm{poly}(n)$ size circuits, and the following condition is satisfied: For $f \in \mathbb{F}[\mathbf{y}]$, $q_i(f(A_{i1} \cdot \mathbf{y}), \ldots, f(A_{ic} \cdot \mathbf{y}), \mathbf{u}) = 0$ for every $i \in [m]$ if and only if $f = \alpha \cdot g$ for some $\alpha \in \mathbb{F}$. Then, g is* characterized by circuit identities *over $\mathbb{F}$.*

The above definition is taken (after slight modifications to suit our purpose) from Definition 3.4.7 in [22] and is attributed to an article by Mulmuley [47].

3 Characterization of NW by symmetries and circuit identities

3.1 Symmetry characterization: Theorem 1

Let $\mathbb{F}$ be a field having a d-th root of unity $\zeta \neq 1$ and $|\mathbb{F}| \neq d + 1$.[7] As d is a prime, ζ is primitive, i.e., $\zeta^d = 1$ and $\zeta^t \neq 1$ for $0 < t < d$. The rows and columns of a matrix in $\mathscr{G}_{\mathsf{NW}}$ are indexed by the set $\{(i, j) : i, j \in \mathbb{F}_d\}$.

▷ Claim 19. The following matrices in $\mathbb{F}^{n \times n}$ are in $\mathscr{G}_{\mathsf{NW}}$:

1. $A_{\boldsymbol{\beta}}$, a diagonal matrix with $A_{\boldsymbol{\beta}}((i, j), (i, j)) = \beta_i \in \mathbb{F}^{\times}$ for $i, j \in [d]$, s.t. $\prod_{i \in [d]} \beta_i = 1$.
2. A_{ℓ}, a diagonal matrix with $((i, j), (i, j))$-th entry as $\zeta^{i^{\ell} \cdot j}$ for $i, j \in [d]$ and $\ell \in [d - k - 1]$.[8]
3. A_h, $h \in \mathbb{F}_d[z]_k$,such that $A_h((i, j), (i, j + h(i))) = 1$ for $i, j \in [d]$ and other entries are 0.

The proof of Claim 19 is given in Section C.1 in [24]. The matrices $A_{\boldsymbol{\beta}}$ are the continuous symmetries while A_{ℓ}, A_h are discrete symmetries of NW for all choices of $\boldsymbol{\beta}, \ell, h$. The symmetries in 2 are very different from the symmetries of the Determinant and the Permanent. The following Claim immediately implies Theorem 1. Its proof is given in Section C.2 in [24].

▷ Claim 20. Let f be a homogeneous degree-d polynomial in $\mathbb{F}[\mathbf{x}]$. If $\mathscr{G}_f$ contains $A_{\boldsymbol{\beta}}, A_{\ell}$ and A_h (for all choices of $\boldsymbol{\beta}$, ℓ and h, mentioned above) then $f = \alpha \cdot \mathsf{NW}$ for some $\alpha \in \mathbb{F}$.

3.2 Characterization by circuit identities

Here we show that NW is characterized by circuit identities (Definition 18). The lemma is crucially used to prove Theorems 2 and 3 in Section 5. Its proof is given in Section C.3 in [24].

▶ **Lemma 21.** *Polynomial* NW *is characterized by circuit identities over any field $\mathbb{F}$.*

4 Lie algebra and symmetries of NW

We first give a complete description of the Lie algebra of NW by giving an explicit $\mathbb{F}$-basis. Then, using this knowledge, we analyse the structure of the symmetries of NW and prove

[7] For a prime d, $|\mathbb{F}| = d + 1$ if and only if d is a Mersenne prime.

[8] Recall, $[d - k - 1] = \{0, 1, \ldots, d - k - 2\}$

Theorem 5. Thereafter, using Theorem 5, we show that NW is not characterized by its symmetries over fields that do not contain a d-th primitive root of unity. The rows and columns of a $n \times n$ matrix in $\mathfrak{g}_{\mathsf{NW}}$ and $\mathscr{G}_{\mathsf{NW}}$ are indexed by the set $\{(i,j) : i,j \in \mathbb{F}_d\}$, which is naturally identified with the $\mathbf{x}$-variables, where $\mathbf{x} = (x_{0,0} \dots x_{0,d-1} \, \dots \, x_{d-1,0} \dots x_{d-1,d-1})^T$.

4.1 Lie algebra of NW

It turns out that the Lie algebra of NW is a subspace of the Lie algebra of every set-multilinear polynomial. (The default partition of a set-multilinear polynomial is $\mathbf{x} = \uplus_{i\in[d]}\mathbf{x}_i$.)

▶ **Lemma 22.** *Let $\mathbb{F}$ be a field and $\mathrm{char}(\mathbb{F}) \neq d$. The dimension of $\mathfrak{g}_{\mathsf{NW}}$ over $\mathbb{F}$ is $d-1$, and the diagonal matrices $B_1, \dots, B_\ell$ (defined below) form a $\mathbb{F}$-basis of $\mathfrak{g}_{\mathsf{NW}}$. For $\ell \in \{1, \dots, d-1\}$,*

$$(B_\ell)_{(i,j),(i,j)} = \begin{cases} 1, & \text{if } i = 0, j \in [d] \\ -1, & \text{if } i = \ell, j \in [d] \\ 0, & \text{otherwise.} \end{cases}$$

The lemma is proven in Section D.1 in [24] by carefully analysing a system of linear equations obtained from the monomials of NW. It follows that every $B \in \mathfrak{g}_{\mathsf{NW}}$ is of the form $\mathrm{diag}(\alpha_0, \dots, \alpha_{d-1}) \otimes I_d$, where each $\alpha_i \in \mathbb{F}$ and $\sum_{i\in[d]} \alpha_i = 0$. The continuous symmetries of NW consist of matrices $A = \mathrm{diag}(\beta_0, \dots, \beta_{d-1}) \otimes I_d$, where each $\beta_i \in \mathbb{C}$ and $\prod_{i\in[d]} \beta_i = 1$.

4.2 Structure of $\mathscr{G}_{\mathsf{NW}}$: Theorem 5

Lemma 22 implies the following.

▷ Claim 23. Every $A \in \mathscr{G}_{\mathsf{NW}}$ is a block-permuted matrix with block size d.

The proof of the claim is given in Section D.2 in [24]. Using Claim 23, Hessian and the evaluation dimension of NW, we give a proof of Theorem 5 in Section D.3 in [24].

4.3 NW is not characterized by its symmetries over $\mathbb{R}$

Let $\mathbb{F}$ be either $\mathbb{R}, \mathbb{Q}$ or a finite field such that $d \nmid |\mathbb{F}| - 1$. Then, $\mathbb{F}$ does not contain a d-th primitive root of unity, and so the matrices A_ℓ, for $\ell \in [d-k-1]$ mentioned in Claim 19, are no longer the symmetries of NW over $\mathbb{F}$. The next lemma shows that over such $\mathbb{F}$ all the diagonal symmetries of NW are of the type $A_{\boldsymbol{\beta}}$ mentioned in Claim 19. This then implies the following theorem, which may seem somewhat surprising as we do not know all the permutation symmetries of NW. The proofs are given in Section D.4 in [24].

▶ **Lemma 24.** *If $D \in \mathscr{G}_{\mathsf{NW}}$ is a diagonal matrix over $\mathbb{F}$ then $D = diag(\beta_0, \dots, \beta_{d-1}) \otimes I_d$, where each $\beta_i \in \mathbb{F}$ and $\prod_{i\in[d]} \beta_i = 1$.*

▶ **Theorem 25.** NW *is not characterized by its symmetries over $\mathbb{F}$.*

5 Circuit testability and the flip theorem for NW

In this and the next section, we show that the knowledge of the symmetries of NW plays a crucial role in answering some of the algorithmic questions related to NW. This section is devoted to Theorems 2 and 3. The main ingredient of their proofs is Lemma 21. We present the circuit testing algorithm here and push the proof of the Flip theorem to Section E in [24].

Proof of Theorem 2. Let $\mathtt{C}$ be a given circuit of size s over $\mathbb{F}$ that computes an n-variate polynomial $f = \mathtt{C}(\mathbf{x})$. Naturally, $\deg(f) \le \delta(s)$. Algorithm 1 intends to check, in steps 2 and 3, if f satisfies the identities given in the proof of Lemma 21. If $f \neq \alpha \cdot \mathsf{NW}$ for all $\alpha \in \mathbb{F}$, then at least one of the identities is not satisfied. For the polynomials q_1, q_2 and q_3 defined in the proof of Lemma 21, observe that the degree of $q_1(f(A_i(u) \cdot \mathbf{x}), f(\mathbf{x}), u)$ is bounded by $2 \cdot \delta(s)$, whereas the degrees of $q_2(f(A_{a,r} \cdot \mathbf{x}), f(\mathbf{x}))$ and $q_3(f(A_t \cdot \mathbf{x}))$ are at most $\delta(s)$. As $|\mathbb{F}| \ge 4 \cdot \delta(s)$, by Schwartz-Zippel lemma [61, 64], step 4 returns "False" with probability at least $\frac{1}{2}$. If $f = \alpha \cdot \mathsf{NW}$ for some $\alpha \in \mathbb{F}$ then all the identities are satisfied, and step 7 ensures that $\alpha = 1$. Clearly, the algorithm uses poly(s) field operations. The success probability is boosted from $\frac{1}{2}$ to $1 - \exp(-s)$ by repeating the algorithm poly(s) times. ◀

Algorithm 1 Circuit testing for NW.

```
Input: Black-box access to a circuit $\mathtt{C}$ of size $s$ over $\mathbb{F}$.
Output: "True" if $\mathtt{C}(\mathbf{x}) = \mathsf{NW}$, else "False".
1. Pick $\mathbf{a} \in_r \mathbb{F}^n$ and $\mu \in_r \mathbb{F}$.
2. for $i \in [d], a \in \mathbb{F}_d^{\times}, r \in [k+1], t \in [d]\backslash[k+1]$ do
3.     if $(\mathtt{C}(A_i(\mu) \cdot \mathbf{a}) - \mu \cdot \mathtt{C}(\mathbf{a}) \neq 0)$ or $(\mathtt{C}(A_{a,r} \cdot \mathbf{a}) - \mathtt{C}(\mathbf{a}) \neq 0)$ or $(\mathtt{C}(A_t \cdot \mathbf{a}) \neq 0)$ then
4.         return "False".
5.     end if
6. end for
7. Let $\mathbf{b} \in \mathbb{F}^n$ be an assignment obtained by setting $x_{i0} = 1$, for $i \in [d]$, and all other
   variables to zero. If $f(\mathbf{b}) \neq 1$, return "False". Else, return "True".
```

6 Equivalence test for NW

First, we show a randomized reduction of equivalence test for NW to block-permuted equivalence test (in short, BP equivalence test) in Lemma 26. Then, we give an efficient equivalence test for NW in the special case where the linear transformation is block-diagonal and is a product of a permutation matrix and a scaling matrix (Theorem 4).

▶ **Lemma 26** (Reduction to BP equivalence test). *Let $\mathbb{F}$ be a field such that* $\mathrm{char}(\mathbb{F}) \neq d$ *and $|\mathbb{F}| \ge 2d^2$. There is a randomized algorithm that takes input as black-box access to a degree d polynomial $f \in \mathbb{F}[\mathbf{x}]$ and does the following with high probability: It outputs black-box access to a degree d polynomial $g \in \mathbb{F}[\mathbf{x}]$ such that f is equivalent to* NW *if and only if g is BP equivalent to* NW*. Moreover, the transformation for f can be recovered efficiently from the transformation for g. The running time of this reduction is* poly(d, ρ)*, where ρ is bit complexity of the coefficients of f* [9].

Proof of correctness. The efficiency of Step 1 follows from Lemma 15. The correctness of Step 2 and 3 follow from the next claim whose proof is given in Section F.1 in [24].

▷ Claim 27. With high probability, matrix D can be computed in poly(d, ρ) time. Moreover, f is equivalent to NW if and only if $f(D \cdot \mathbf{x})$ is BP equivalent to NW.

[9] We assume that univariate polynomial factorization over $\mathbb{F}$ can be done in polynomial time.

Algorithm 2 Reduction of equivalence test for NW to BP equivalence test.

```
Input: Black-box access to $f \in \mathbb{F}[\mathbf{x}]$.
Output: Black-box access to $g \in \mathbb{F}[\mathbf{x}]$.

1. Compute a basis $L_1, \ldots, L_r$ of $\mathfrak{g}_f$. If $r \neq d-1$, output "$f$ is not equivalent to NW".
2. Let $S$ be an arbitrary subset of $\mathbb{F}$ of size $d^2$. Let $L = a_1L_1 + \ldots + a_rL_r$, where $a_i \in_r S$.
   Compute $D \in \mathrm{GL}_{d^2}(\mathbb{F})$ such that $D^{-1} \cdot L \cdot D = \mathrm{diag}(\beta_1, \ldots, \beta_d) \otimes I_d$, where $\beta_j \in \mathbb{F}$. If
   no such $D$ exists then output "$f$ is not equivalent to NW."
3. Output black-box access to $f(D \cdot \mathbf{x})$.
```

6.1 BD-PS equivalence test for NW: Theorem 4

Lemma 26 implies that to solve equivalence test for NW it is sufficient to focus on BP equivalence test. Here, we solve a special case of BP equivalence test, namely BD-PS equivalence test. We prove Theorem 4 in two steps: first we reduce BD-PS equivalence test to scaling equivalence test and then solve the scaling equivalence test. The algorithm pretends that f is BD-PS equivalent to NW and computes a block-diagonal permutation matrix A and an invertible scaling matrix B. In the end, the circuit testing algorithm of NW (Algorithm 1) is used to check if $f(A^{-1} \cdot B^{-1} \cdot \mathbf{x}) = \mathsf{NW}$.

6.1.1 Reduction of BD-PS equivalence test to scaling equivalence test

Assume $f = \mathsf{NW}(B \cdot A \cdot \mathbf{x})$, where A is a block-diagonal permutation matrix and B is an invertible scaling matrix. Algorithm 3 does not explicitly use the knowledge of the entries of B. Thus, we may assume without loss of generality that $B = I_{d^2}$. Then, the task reduces to solving the BD permutation equivalence test for NW. We identify matrix A with d permutations $\sigma_0, \ldots, \sigma_{d-1}$ on $[d]$ as $A = \mathrm{diag}(M_{\sigma_0}, \ldots, M_{\sigma_{d-1}})$, where M_{σ_i} is the $d \times d$ permutation matrix corresponding to σ_i [10].

▶ **Observation 6.1.** *Suppose f is BD permutation equivalent to* NW*, i.e. $f = \mathsf{NW}(A \cdot \mathbf{x})$. Then, a monomial $\prod_{i \in \mathbb{F}_d} x_{i,h(i)}$ of* NW *gets mapped to a unique monomial $\prod_{i \in \mathbb{F}_d} x_{i,\sigma_i(h(i))}$ of f.*

Algorithm 3 starts by assuming that $\sigma_0(0) = \cdots = \sigma_k(0) = 0$ and $\sigma_0(1) = 1$. The symmetries of NW allow us to make this assumption (Claim 28). The aim is to figure out all the entries of σ_i [11]. This is done by carefully picking a bunch of polynomials from $\mathbb{F}_d[z]_k$ (which we call *nice* polynomials) and then exploiting the association between f and NW mentioned in Observation 6.1 using these polynomials. The algorithm works over every field.

Proof of correctness. The following claims argue the correctness of the algorithm. Their proofs are given in Section F.2 in [24]. In these claims, ρ is the bit complexity of the coefficients of f.

▷ Claim 28. (Canonical form of $\sigma_0, \ldots, \sigma_{d-1}$): Suppose $f \in \mathbb{F}[\mathbf{x}]$ is BD permutation equivalent to NW. Then, there exist permutations $\sigma_0, \ldots, \sigma_{d-1}$ on $[d]$ such that $\sigma_0(0) = \cdots = \sigma_k(0) = 0, \sigma_0(1) = 1$ and $A = \mathrm{diag}(M_{\sigma_0}, \ldots, M_{\sigma_{d-1}})$ satisfies $f = \mathsf{NW}(A \cdot \mathbf{x})$.

[10] For $i, r, s \in [d], M_{\sigma_i}(r, s) = 1$ if and only if $\sigma_i(r) = s$.
[11] σ_i is treated as an ordered tuple $(\sigma_i(0), \ldots, \sigma_i(d-1))$

Algorithm 3 Block-diagonal permutation equivalence test for NW.

```
Input: Black-box access to $f \in \mathbb{F}[\mathbf{x}]$.
Output: Black-box access to $g \in \mathbb{F}[\mathbf{x}]$ such that if $f$ is BD-PS equivalent to NW then $g$ is scaling equivalent to NW.

1. Assume that $\sigma_0(0) = \cdots = \sigma_k(0) = 0$ and $\sigma_0(1) = 1$ (Claim 28).
2. Construct a list of nice polynomials in $\mathbb{F}_d[z]_k$ (Definition 29) as mentioned in Claim 30.
3. Recover $(d-k)$ distinct entries of each $\sigma_0, \ldots, \sigma_{d-1}$ as mentioned in Claim 31.
4. Let $N$ be a $d \times d$ matrix, where the columns and rows are indexed by $(\sigma_0, \ldots, \sigma_{d-1})$ and
   $(0, \ldots, d-1)$ respectively and for $l, i \in [d], N(l, i) := \sigma_i(l)$. Pick $l_0, \ldots, l_k \in [d]$ such that
   in each of the rows indexed by $l_0, \ldots, l_k$ at least $k+1$ entries are known (Claim 32).
5. Use $l_0, \ldots, l_k \in [d]$ to recover all the entries of the rows of $N$ as mentioned in Claim 33.
   Compute $A = \text{diag}(M_{\sigma_0}, \ldots, M_{\sigma_{d-1}})$ and return black box access to $f(A^{-1} \cdot \mathbf{x})$
```

▶ **Definition 29.** *(List of nice polynomials in $\mathbb{F}_d[z]_k$): $\{h_0, \ldots, h_{d-k-1}\} \subseteq \mathbb{F}_d[z]_k$ is called a list of nice polynomials if the following properties are satisfied:*

1. *For distinct $r_1, r_2 \in [d-k]$, $h_{r_1}(\ell) = h_{r_2}(\ell)$ for every $\ell \in [k]$ and $h_{r_1}(\ell) \neq h_{r_2}(\ell)$ for every $\ell \in \{k, \ldots, d-1\}$.*
2. *For every $r \in [d-k], \sigma_0(h_r(0)), \ldots, \sigma_k(h_r(k))$ can be computed in poly(d, ρ) time.*

▷ Claim 30. A list of $d-k$ nice polynomials $\{h_0, \ldots, h_{d-k-1}\}$ can be computed in poly(d, ρ) time.

Using the list of nice polynomials, we recover $d-k$ distinct entries of $\sigma_0, \ldots, \sigma_{d-1}$.

▷ Claim 31. Given a list of nice polynomials $\{h_0, \ldots, h_{d-k-1}\}$, we can recover $d-k$ distinct entries in each of $\sigma_0, \ldots, \sigma_{d-1}$ in poly(d, ρ) time.

The matrix N defined in the algorithm is filled with some known entries and some unknowns. The goal is to recover all the entries of N which is accomplished by the following claims.

▷ Claim 32. Suppose $k \in [1, \frac{d}{3}]$. Then, there exist $k+1$ rows in N such that in each of these rows at least $k+1$ entries are known.

▷ Claim 33. Using $k+1$ rows of N indexed by $l_0, \ldots, l_k$ (as mentioned in Step 4), we can recover all the entries of N in poly(d, ρ) time.

6.1.2 Scaling equivalence test for NW

We present an algorithm for solving the scaling equivalence test for NW over a finite field $\mathbb{F}$, where $d \nmid |\mathbb{F}| - 1$. The same algorithm with appropriate modifications works over $\mathbb{R}$. More details on this are given in Section F.4 in [24]. Assume that f is scaling equivalent to NW.

Proof of correctness. The following claims and observations argue the correctness of the algorithm. The proofs of the claims are given in Section F.3 in [24].

▷ Claim 34. We can assume that $\alpha_{1,0} = \ldots = \alpha_{d-1,0} = 1$ without loss of generality.

The following observation can be proved easily.

▶ **Observation 6.2.** *Given a monomial m, we can recover the coefficient of m in f in* poly(d, ρ) *time.*

▷ Claim 35. In Step 4, $\alpha_{i,j}$ can be computed in poly(d, ρ) time. Further, $f = \mathsf{NW}(B \cdot \mathbf{x})$.

Algorithm 4 Scaling equivalence test for NW over finite fields.

Input: Black box access to $f \in \mathbb{F}[\mathbf{x}]$.
Output: An invertible diagonal matrix B such that $f = \mathsf{NW}(B \cdot \mathbf{x})$.

1. Let $B = \text{diag}(\alpha_{0,0}, \ldots, \alpha_{d-1,d-1})$, where $\{\alpha_{i,j} \ : \ i, j \in [d]\}$ are unknown. Set $\alpha_{1,0} = \ldots = \alpha_{d-1,0} = 1$ (Claim 34).
2. Let $S = (0, z, \ldots, (d-1)z,\ 1, z+1 \ldots, (d-1)z+1,\ \ldots\ , d-2, z+d-2 \ldots, (d-1)z+d-2,\ d-1)$ be the ordered set of $d^2 - d + 1$ polynomials in $\mathbb{F}[z]$. For every $h \in S$, query the coefficient c_h of the monomial $\prod_{i \in \mathbb{F}_d} x_{i,h(i)}$ from the black-box of f (Observation 6.2).
3. Let C be a $0/1$ matrix of size $(d^2-d+1) \times (d^2-d+1)$ whose rows and columns are indexed by S and $\mathbf{y} = (y_{0,0}, \ldots, y_{0,d-1}, y_{1,1}, \ldots, y_{1,d-1}, \ldots, y_{d-1,1}, \ldots, y_{d-1,d-1})$, respectively, such that for $h \in S$ and $y_{i,j} \in \mathbf{y}$, the $(h, y_{i,j})$-th entry of C is 1 if $h(i) = j$. (It is argued in Claim 39 in [24] that $|\det(C)|$ is a power of d). Compute the inverse of $\det(C)$ in $\mathbb{Z}_{|\mathbb{F}|-1}$ and denote it by γ. (Note that $\mathbf{y}$ does not contain the variables $\{y_{1,0}, \ldots, y_{d-1,0}\}$.)
4. Fix $\alpha_{i,j} \in \{\alpha_{0,0}, \ldots, \alpha_{d-1,d-1}\} \setminus \{\alpha_{1,0}, \ldots, \alpha_{d-1,0}\}$ arbitrarily. For every $h \in S$, compute the minor of C with respect to the row and column indexed by h and $y_{i,j}$ respectively and call it δ_h. Set $\alpha_{i,j} = \prod_{h \in S} c_h^{(\delta_h \cdot \gamma) \mod (|\mathbb{F}|-1)}$.
5. Set $B = \text{diag}(\alpha_{0,0}, \ldots, \alpha_{d-1,d-1})$. Return B. (see Claim 35)

7 Few problems

In conclusion, we state a few problems on the NW polynomial which, if resolved, would shed more light on this fundamental polynomial family.

1. Is the $\mathcal{NW} = \{\mathsf{NW}_{d,k} : d \text{ is a prime}\}$ family VNP-complete for a suitable choice of k (say, $k = d^{\epsilon}$ for a constant $\epsilon > 0$)?
2. Is there an efficient algorithm to check if $\mathsf{NW}(\mathbf{a}) = 0$ at a given point $\mathbf{a} \in \{0,1\}^n$? This problem was also posed in [6] [12].
3. Is there an efficient general equivalence test for NW? Theorem 4 may turn out to be a vital ingredient in such a test.
4. Give a complete description of the permutation symmetries of NW. Are all the permutation symmetries captured in Lemma 45 mentioned in Section D in [24]?

For the permanent polynomial, the solutions to these problems are well known.

References

1 Scott Aaronson. P=?NP. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:4, 2017.

2 Miklós Ajtai. Σ^1_1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):1–48, 1983.

3 Noga Alon and Ravi B. Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7(1):1–22, 1987.

4 Albert Atserias. Distinguishing SAT from Polynomial-Size Circuits, through Black-Box Queries. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 88–95, 2006.

[12] We thank Andrej Bogdanov for pointing this out to us.

5 Markus Bläser, Christian Ikenmeyer, Gorav Jindal, and Vladimir Lysikov. Generalized matrix completion and algebraic natural proofs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1193–1206, 2018.

6 Andrej Bogdanov and Muli Safra. Hardness Amplification for Errorless Heuristics. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 418–426, 2007.

7 Peter Bürgisser, Christian Ikenmeyer, and Greta Panova. No Occurrence Obstructions in Geometric Complexity Theory. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 386–395, 2016.

8 Xi Chen, Neeraj Kayal, and Avi Wigderson. Partial Derivatives in Arithmetic Complexity and Beyond. *Foundations and Trends in Theoretical Computer Science*, 6(1-2):1–138, 2011.

9 Suryajith Chillara, Nutan Limaye, and Srikanth Srinivasan. Small-depth Multilinear Formula Lower Bounds for Iterated Matrix Multiplication, with Applications. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 21:1–21:15, 2018.

10 Suryajith Chillara and Partha Mukhopadhyay. Depth-4 Lower Bounds, Determinantal Complexity: A Unified Approach. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, pages 239–250, 2014.

11 Klim Efremenko, Ankit Garg, Rafael Mendes de Oliveira, and Avi Wigderson. Barriers for Rank Methods in Arithmetic Complexity. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 1:1–1:19, 2018.

12 Michael A. Forbes, Mrinal Kumar, and Ramprasad Saptharishi. Functional Lower Bounds for Arithmetic Circuits and Connections to Boolean Circuit Complexity. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 33:1–33:19, 2016.

13 Michael A. Forbes, Amir Shpilka, and Ben Lee Volk. Succinct hitting sets and barriers to proving algebraic circuits lower bounds. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 653–664, 2017.

14 Lance Fortnow, Aduri Pavan, and Samik Sengupta. Proving SAT does not have small circuits with an application to the two queries problem. *J. Comput. Syst. Sci.*, 74(3):358–363, 2008.

15 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.

16 Georg Frobenius. Ueber die darstellung der endlichen gruppen durch linearc substitutionen. *Sitzungber. der Berliner Akademie*, 7:994–1015, 1897.

17 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 260–270, 1981.

18 Ankit Garg, Nikhil Gupta, Neeraj Kayal, and Chandan Saha. Determinant equivalence test over finite fields and over Q. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:42, 2019.

19 Fulvio Gesmundo. Gemetric aspects of iterated matrix multiplication. *Journal of Algebra*, 461:42–64, 2016.

20 Dima Grigoriev and Marek Karpinski. An Exponential Lower Bound for Depth 3 Arithmetic Circuits. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 577–582, 1998.

21 Joshua A. Grochow, Mrinal Kumar, Michael E. Saks, and Shubhangi Saraf. Towards an algebraic natural proofs barrier via polynomial identity testing. *CoRR*, abs/1701.01717, 2017.

22 Joshua Abraham Grochow. *Symmetry and equivalence relations in classical and geometric complexity theory*. PhD thesis, Department of Computer Science, The University of Chicago, Chicago, Illinois, 2012.

23 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Approaching the Chasm at Depth Four. *J. ACM*, 61(6):33:1–33:16, 2014.

24 Nikhil Gupta and Chandan Saha. On the symmetries of and equivalence test for design polynomials. *ECCC*, 2019. URL: `https://eccc.weizmann.ac.il/report/2018/164/`.

25 Brian C Hall. *Lie Groups, Lie Algebras and Representations An Elementary introduction*. Springer, second edition, 2015.

26 Johan Håstad. Almost Optimal Lower Bounds for Small Depth Circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20, 1986.

27 Jesko Hüttenhain. The Stabilizer of Elementary Symmetric Polynomials. *CoRR*, abs/1607.08419, 2016. URL: `https://arxiv.org/abs/1607.08419`.

28 Christian Ikenmeyer, Ketan D. Mulmuley, and Michael Walter. On vanishing of Kronecker coefficients. *Computational Complexity*, 26(4):949–992, 2017.

29 Christian Ikenmeyer and Greta Panova. Rectangular Kronecker Coefficients and Plethysms in Geometric Complexity Theory. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 396–405, 2016.

30 Neeraj Kayal. Affine projections of polynomials: extended abstract. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 643–662, 2012.

31 Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. An Exponential Lower Bound for Homogeneous Depth Four Arithmetic Formulas. *SIAM J. Comput.*, 46(1):307–335, 2017.

32 Neeraj Kayal, Vineet Nair, Chandan Saha, and Sébastien Tavenas. Reconstruction of Full Rank Algebraic Branching Programs. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, pages 21:1–21:61, 2017.

33 Neeraj Kayal and Chandan Saha. Lower Bounds for Depth-Three Arithmetic Circuits with small bottom fanin. *Computational Complexity*, 25(2):419–454, 2016.

34 Neeraj Kayal, Chandan Saha, and Ramprasad Saptharishi. A super-polynomial lower bound for regular arithmetic formulas. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 146–153, 2014.

35 Neeraj Kayal, Chandan Saha, and Sébastien Tavenas. An Almost Cubic Lower Bound for Depth Three Arithmetic Circuits. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 33:1–33:15, 2016.

36 Neeraj Kayal, Chandan Saha, and Sébastien Tavenas. On the size of homogeneous and of depth four formulas with low individual degree. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 626–632, 2016.

37 Mrinal Kumar and Ramprasad Saptharishi. An Exponential Lower Bound for Homogeneous Depth-5 Circuits over Finite Fields. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, pages 31:1–31:30, 2017.

38 Mrinal Kumar and Shubhangi Saraf. Superpolynomial Lower Bounds for General Homogeneous Depth 4 Arithmetic Circuits. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 751–762, 2014.

39 Mrinal Kumar and Shubhangi Saraf. The limits of depth reduction for arithmetic formulas: it's all about the top fan-in. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 136–145, 2014.

40 Mrinal Kumar and Shubhangi Saraf. Arithmetic Circuits with Locally Low Algebraic Rank. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 34:1–34:27, 2016.

41 Mrinal Kumar and Shubhangi Saraf. Sums of Products of Polynomials in Few Variables: Lower Bounds and Polynomial Identity Testing. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 35:1–35:29, 2016.

42 Mrinal Kumar and Shubhangi Saraf. On the Power of Homogeneous Depth 4 Arithmetic Circuits. *SIAM J. Comput.*, 46(1):336–387, 2017.

43 Richard J. Lipton. New Directions In Testing. In *Distributed Computing And Cryptography, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, October 4-6, 1989*, pages 191–202, 1989.

44 Marvin Marcus and Francis May. The permanent function. *Canadian Journal of Mathematics*, 14:177–189, 1962.

45 Thierry Mignon and Nicolas Ressayre. A quadratic bound for the determinant and permanent problem. *International Mathematics Research Notes*, 2004(79):4241–4253, 2004.

46 Ketan Mulmuley. Lower Bounds in a Parallel Model without Bit Operations. *SIAM J. Comput.*, 28(4):1460–1509, 1999.

47 Ketan Mulmuley. On P vs. NP, Geometric Complexity Theory, and the Flip I: a high level view. *CoRR*, abs/0709.0748, 2007.

48 Ketan Mulmuley. Explicit Proofs and The Flip. *CoRR*, abs/1009.0246, 2010. URL: `http://arxiv.org/abs/1009.0246`.

49 Ketan Mulmuley. On P vs. NP and geometric complexity theory: Dedicated to Sri Ramakrishna. *J. ACM*, 58(2):5:1–5:26, 2011.

50 Ketan Mulmuley. The GCT program toward the *P* vs. *NP* problem. *Commun. ACM*, 55(6):98–107, 2012.

51 Ketan Mulmuley and Milind A. Sohoni. Geometric Complexity Theory I: An Approach to the P vs. NP and Related Problems. *SIAM J. Comput.*, 31(2):496–526, 2001.

52 Cody Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime: an easy witness lemma for NP and NQP. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 890–901, 2018.

53 Noam Nisan and Avi Wigderson. Hardness vs Randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

54 Noam Nisan and Avi Wigderson. Lower Bounds on Arithmetic Circuits Via Partial Derivatives. *Computational Complexity*, 6(3):217–234, 1997.

55 Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2):8:1–8:17, 2009.

56 Ran Raz and Amir Yehudayoff. Lower Bounds and Separations for Constant Depth Multilinear Circuits. *Computational Complexity*, 18(2):171–207, 2009.

57 Alexander A. Razborov. Lower bounds on the monotone complexity of some Boolean functions. *Soviet Mathematics Doklady*, 31:354–357, 1985.

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

59 Alexander A. Razborov and Steven Rudich. Natural Proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

60 Kenneth W. Regan. Understanding the Mulmuley-Sohoni Approach to P vs. NP. *Bulletin of the EATCS*, 78:86–99, 2002.

61 Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4):701–717, 1980.

62 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, 1987, New York, New York, USA*, pages 77–82, 1987.

63 Ryan Williams. Nonuniform ACC Circuit Lower Bounds. *J. ACM*, 61(1):2:1–2:32, 2014.

64 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, pages 216–226, 1979.

The Complexity of Homomorphism Indistinguishability

Jan Böker
RWTH Aachen University, Aachen, Germany
boeker@informatik.rwth-aachen.de

Yijia Chen
Fudan University, Shanghai, China
yijiachen@fudan.edu.cn

Martin Grohe
RWTH Aachen University, Aachen, Germany
grohe@informatik.rwth-aachen.de

Gaurav Rattan
RWTH Aachen University, Aachen, Germany
rattan@informatik.rwth-aachen.de

Abstract

For every graph class $\mathcal{F}$, let $\textsc{HomInd}(\mathcal{F})$ be the problem of deciding whether two given graphs are homomorphism-indistinguishable over $\mathcal{F}$, i.e., for every graph F in $\mathcal{F}$, the number $\mathsf{hom}(F, G)$ of homomorphisms from F to G equals the corresponding number $\mathsf{hom}(F, H)$ for H. For several natural graph classes (such as paths, trees, bounded treewidth graphs), homomorphism-indistinguishability over the class has an efficient structural characterization, resulting in polynomial time solvability [6].

In particular, it is known that two non-isomorphic graphs are homomorphism-indistinguishable over the class $\mathcal{T}_k$ of graphs of treewidth k if and only if they are not distinguished by k-dimensional Weisfeiler-Leman algorithm, a central heuristic for isomorphism testing: this characterization implies a polynomial time algorithm for $\textsc{HomInd}(\mathcal{T}_k)$, for every fixed $k \in \mathbb{N}$. In this paper, we show that there is a polynomial-time-decidable class $\mathcal{F}$ of undirected graphs of *bounded* treewidth such that $\textsc{HomInd}(\mathcal{F})$ is undecidable.

Our second hardness result concerns the class $\mathcal{K}$ of complete graphs. We show that $\textsc{HomInd}(\mathcal{K})$ is co-NP-hard, and in fact, we show completeness for the class $\mathsf{C}_{=}\mathsf{P}$ (under P-time Turing reductions). On the algorithmic side, we show that $\textsc{HomInd}(\mathcal{P})$ can be solved in polynomial time for the class $\mathcal{P}$ of directed paths. We end with a brief study of two variants of the $\textsc{HomInd}(\mathcal{F})$ problem: (a) the problem of lexographic-comparison of homomorphism numbers of two graphs, and (b) the problem of computing certain distance-measures (defined via homomorphism numbers) between two graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory

Keywords and phrases graph homomorphism numbers, counting complexity, treewidth

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.54

Funding *Gaurav Rattan*: Supported by DFG grant RA 3242/1-1.

1 Introduction

A classic theorem due to Lovász [12] states that two graphs G, H are isomorphic if and only if for every graph F the number $\mathsf{hom}(F, G)$ of homomorphisms from F to G is equal to the number $\mathsf{hom}(F, H)$ of homomorphisms from F to H. Jointly with Dell, the last two authors of this paper recently proved [6] that two graphs are fractionally isomorphic, or equivalently, can be distinguished by the colour refinement algorithm (see e.g. [8]), if and only if for every tree T it holds that $\mathsf{hom}(T, G) = \mathsf{hom}(T, H)$. Another well-known fact (see e.g. [20]) is that

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 54; pp. 54:1–54:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

two graphs are co-spectral, that is, their adjacency matrices have the same sequence of eigenvalues, if and only if for every cycle C it holds that $\mathsf{hom}(C,G) = \mathsf{hom}(C,H)$. Thus counting homomorphisms from all graphs in some class gives us very interesting equivalence relations, which we call homomorphism indistinguishability: for every class $\mathcal{F}$ of graphs, two graphs G, H are *homomorphism-indistinguishable over* $\mathcal{F}$ (for short: $\mathcal{F}$*-HI*) if and only if for every $F \in \mathcal{F}$ it holds that $\mathsf{hom}(F,G) = \mathsf{hom}(F,H)$.[1] Thus two graphs are $\mathcal{G}$-HI for the class $\mathcal{G}$ of all graphs if and only if they are isomorphic. They are $\mathcal{T}$-HI for the class $\mathcal{T}$ of trees if and only if they are fractionally isomorphic, and they are $\mathcal{C}$-HI for the class $\mathcal{C}$ of cycles if and only if they are co-spectral. We have also proved in [6] that for every k, two graphs are $\mathcal{T}_k$-HI for the class $\mathcal{T}_k$ of all graphs of tree width at most k if and only if the k-dimensional Weisfeiler-Leman algorithm does not distinguish them, and we gave a characterisation of $\mathcal{P}$-HI for the class $\mathcal{P}$ of paths in terms of a natural system of linear equations. Furthermore, the first author of this paper recently proved that two directed graphs are $\mathcal{A}$-HI for the class $\mathcal{A}$ of all directed acyclic graphs if and only if they are isomorphic [3] (also see [13]).

It follows from these results that the problem of deciding whether two graphs are homomorphism-indistinguishable over some class has a quite interesting complexity theoretic behaviour. For every class $\mathcal{F}$ of graphs, let $\textsc{HomInd}(\mathcal{F})$ be the problem of deciding whether two given graphs are $\mathcal{F}$-HI. Since the graph isomorphism problem is in solvable in quasipolynomial time [2], $\textsc{HomInd}(\mathcal{G})$ is in quasipolynomial time. Furthermore, it is an immediate consequence of the characterisations above that $\textsc{HomInd}(\mathcal{T})$, $\textsc{HomInd}(\mathcal{C})$, $\textsc{HomInd}(\mathcal{P})$, and $\textsc{HomInd}(\mathcal{T}_k)$ for all $k \geq 1$ are in polynomial time. This prompts the question whether $\textsc{HomInd}(\mathcal{F})$ is in quasipolynomial time, or even in polynomial time, for every class $\mathcal{F}$ of graphs. While this seems unlikely, it is not obvious what complexities one might expect for $\textsc{HomInd}(\mathcal{F})$ as $\mathcal{F}$ ranges over all classes of graphs. We study these and related questions here.

Homomorphism counts give us embeddings of graphs into an infinite dimensional vector space: for a class $\mathcal{F}$, we can associate with every graph G a *homomorphism vector* $\mathsf{Hom}_{\mathcal{F}}(G) := \big(\mathsf{hom}(F,G) \bigm| F \in \mathcal{F}\big) \in \mathbb{R}^{\mathcal{F}}$. Then $\textsc{HomInd}(\mathcal{F})$ is simply the problem of deciding whether two graphs have the same homomorphism vector. Defining a metric or, even better, an inner product on the range vector space, we obtain a (pseudo-)metric on the class of all graphs (where two non-isomorphic graphs may have distance 0). We will also study the problem of approximately computing the distance between two graphs with respect to metrics obtained this way. Such metrics are important for machine learning tasks such as clustering and classification on graphs. In fact, homomorphism vectors are closely related to the so-called graph kernels (see, e.g. [16, 17, 21]): almost all such graph kernels are defined as the inner products of homomorphism vectors for natural graph classes. Thus our results shed some new light on the complexity of computing these kernels.

At first sight, $\textsc{HomInd}(\mathcal{F})$ looks like a problem in $\mathsf{co\text{-}NP}$: to witness that G and H are not $\mathcal{F}$-HI we just have to nondeterministically guess one $F \in \mathcal{F}$ and verify that $\mathsf{hom}(F,G) \neq \mathsf{hom}(F,H)$. But this argument is flawed for several reasons. First of all, it assumes that for given F, G we can compute $\mathsf{hom}(F,G)$ in polynomial time, which in general is not the case. Let $\#\textsc{Hom}(\mathcal{F})$ be the problem of computing $\mathsf{hom}(F,G)$, given $F \in \mathcal{F}$ and G. Then for the class $\mathcal{G}$ of all graphs, $\#\textsc{Hom}(\mathcal{G})$ is $\#\mathsf{P}$-complete. Under the complexity assumption that $\#\mathsf{W[1]} \neq \mathsf{FPT}$, it is known that $\#\textsc{Hom}(\mathcal{F})$ is in P if and only if $\mathcal{F}$ has bounded tree

[1] We note that homomorphism indistinguishability is incomparable to homomorphic equivalence; remember that two graphs G, H are *homomorphically equivalent* if there is a homomorphism from G to H and a homomorphism from H to G.

width [5]. This result extends to directed graphs and in fact all classes of relational structures of bounded arity. But even for $\mathcal{F}$ with tractable $\#\textsc{Hom}(\mathcal{F})$, the problem $\textsc{HomInd}(\mathcal{F})$ is not necessarily in co-NP, because the witness F may be very large compared to G and H.

▶ **Theorem 1.** *There is a polynomial-time-decidable class $\mathcal{F}$ of undirected graphs of bounded tree width such that $\textsc{HomInd}(\mathcal{F})$ is undecidable.*

We also prove a version of the above theorem for directed graphs (Theorem 15): the graphs in the corresponding class $\mathcal{F}$ are just directed paths padded by isolated vertices. And we cannot only prove undecidability, but also use similar arguments to obtain all kinds of complexities. So the complexity landscape for problems $\textsc{HomInd}(\mathcal{F})$, which started out in quasi-polynomial time, looks fairly complicated. But admittedly the classes $\mathcal{F}$ we can use to prove Theorem 1 are quite esoteric from a graph theoretic point of view. What about "natural" graph classes? For the class $\mathcal{K}$ of all complete graphs, the corresponding problem $\textsc{HomInd}(\mathcal{K})$ turns out to be hard.

▶ **Theorem 2.** $\textsc{HomInd}(\mathcal{K})$ *is* co-NP*-hard.*

For an upper bound, note that $\textsc{HomInd}(\mathcal{K}) \in \mathsf{P}^{\#\mathsf{P}}$, because to decide whether for two n-vertex graphs G, H it holds that $\mathsf{hom}(K, G) = \mathsf{hom}(K, H)$ for all complete graphs K, we only need to check the equality for all K of size at most n. Actually, we pinpoint the exact complexity of $\textsc{HomInd}(\mathcal{K})$ by proving it to be complete for the complexity class $\mathsf{C}_=\mathsf{P}$ (see Theorem 18). This implies that $\textsc{HomInd}(\mathcal{K})$ is not in the polynomial hierarchy PH unless PH collapses (see Corollary 22).

We also look at tractable cases of the homomorphism indistinguishability problem. In particular, we prove that $\textsc{HomInd}(\mathcal{P}_\rightarrow)$ is in P for the class $\mathcal{P}_\rightarrow$ of directed paths (see Theorem 23).

In the last section we look at variants of $\textsc{HomInd}(\mathcal{F})$. A first such variant is the problem $\textsc{HomLex}(\mathcal{F})$ of lexicographically comparing the homomorphism vectors over $\mathcal{F}$ of two input graphs. Of course the lexicographical order depends on some order of the graphs in $\mathcal{F}$; the simple classes $\mathcal{F}$ we consider (directed paths and cycles as well as complete graphs) have only one graph per size, and we can use the natural order by size. We prove that $\textsc{HomLex}(\mathcal{P}_\rightarrow)$ and $\textsc{HomLex}(\mathcal{C}_\rightarrow)$ are in polynomial time for the classes $\mathcal{P}_\rightarrow$ and $\mathcal{C}_\rightarrow$ of directed paths and cycles and that $\textsc{HomLex}(\mathcal{K})$ is #P-complete.

Finally, we study the problem of computing the distance between two graphs with respect to various metrics defined on the homomorphism vectors $\mathsf{Hom}_{\mathcal{F}}(G)$. We prove that if $\mathcal{F}$ is a polynomial time enumerable class of graphs for which $\#\textsc{Hom}(\mathcal{F})$ is in polynomial time then the distance between two graphs can be approximated up to an arbitrarily small additive error ε in polynomial time. This is not a deep result, but we believe it is quite relevant, because computing distances between graphs (even approximately) with respect to various distance measures tends to be a very hard algorithmic problem (e.g. [1, 9, 11, 14, 15]). Here we have a family of natural metrics for which it is tractable.

2 Preliminaries

Directed Graphs. A directed graph (or digraph) G consists of a finite set of vertices $V(G)$ (or V_G) and a set of edges $E(G) \subseteq V \times V$. A subgraph H of a directed graph G is a graph satisfying $V_H \subseteq V_G$ and $E_H \subseteq E_G \cap (V_H \times V_H)$. We assume familiarity with the basic terminology from graph theory, e.g., path, cycle etc., which can be found in e.g., [7]. Given two digraphs G and H, a homomorphism from G to H is an edge-preserving mapping from $V(G)$ to $V(H)$, i.e., a mapping $\varphi : V_G \to V_H$ such that its natural extension to $V_G \times V_G$ satisfies $\varphi(E_G) \subseteq E_H$.

A set X of vertices of a directed graph G is a *clique* if for all distinct $x, y \in X$, either $(x, y) \in E(G)$ or $(y, x) \in E(G)$. For every $k \geq 1$, let $c_k(G)$ be the number of cliques of size k in G. For every directed graph G and every $k \geq 1$, let $d_k(G)$ be the number of directed paths of length $k - 1$ in G. We call a directed graph P a *partial order* if it is acyclic and the edge relation is transitive. Observe that if P is a partial order then the edge relation induces a linear order on every clique. Furthermore, a subgraph Q of P is a directed path if and only if $V(Q)$ is a clique in P. Thus for every $k \geq 1$ we have $c_k(P) = d_k(P)$.

Denote $\mathsf{hom}(G, H)$ to be the number of homomorphisms from G to H. For a class $\mathcal{F}$ of graphs and a graph G, the *homomorphism vector of G over $\mathcal{F}$* is

$$\mathsf{Hom}_{\mathcal{F}}(G) := \big(\mathsf{hom}(F, G) \bigm| F \in \mathcal{F}\big).$$

Undirected Graphs. The notions of subgraph, homomorphisms and homomorphism vector have analogous standard definitions in the undirected case. Denote the edge $\{u, v\}$ of an undirected graph as uv. Let K_k denote the undirected k-clique, i.e. the graph defined by $V(G) = [k]$ and $E(G) = \binom{[k]}{2}$.

▶ **Definition 3.** *Let G, H be two undirected graphs. The product graph $G \otimes H$ is defined by*

$$V(G \otimes H) := V(G) \times V(H),$$
$$E(G \otimes H) := \big\{\{(u, v), (u', v')\} \bigm| uu' \in E(G) \text{ and } vv' \in E(H)\big\}.$$

Let G, H be two graphs, and $k \geq 1$. The following folklore lemma states that the homomorphism numbers are multiplicative for the above-mentioned product operation.

▶ **Lemma 4.** *For every graph F, $\mathsf{hom}(F, G \otimes H) = \mathsf{hom}(F, G) \cdot \mathsf{hom}(F, H)$.*

It is well-known that for every $k \in \mathbb{N}$ and every graph G, $c_k(G) = \frac{\mathsf{hom}(K_k, G)}{k!}$.

▶ **Corollary 5.** *$c_k(G \otimes H) = k! \cdot c_k(G) \cdot c_k(H)$.*

The following construction will be useful later.

▶ **Proposition 6.** *Let n, ℓ, k be fixed positive integers such that $k \leq n$ and $\ell \leq \binom{n}{k}$. In time poly(n), we can construct a graph P satisfying $c_k(P) = \ell$.*

Proof. Let

$$n_1 := \max\left\{n' \,\middle|\, n' \geq k \text{ and } \binom{n'}{k} \leq \ell\right\}, \quad i_1 := \max\left\{i \,\middle|\, i \cdot \binom{n_1}{k} \leq \ell\right\}.$$

Clearly, $k \leq n_1 \leq n$ and $1 \leq i_1$, and both can be computed in time poly(n). Moreover, the remainder

$$\ell' := \ell - i_1 \cdot \binom{n_1}{k} < \ell$$

satisfies $0 \leq \ell' < \binom{n_1}{k}$. An exhaustive application of this rule ensures the existence of numbers $1 \leq s \leq n$, $1 \leq i_1, \ldots, i_s$, and $n_s < n_{s-1} < \cdots < n_1 \leq n$ such that

$$\ell = i_1 \cdot \binom{n_1}{k} + \cdots + i_s \cdot \binom{n_s}{k}.$$

Moreover, all $i_1, \ldots, i_s$ and $n_1, \ldots, n_s$ can be computed in time poly(n). Finally, the desired graph P consists of i_1 disjoint copies of K_{n_1}, i_2 disjoint copies of K_{n_2}, and so on. Clearly, $c_k(P) = \ell$ by our construction. ◀

Cayley-Hamilton. The famous Cayley-Hamilton theorem states that the substitution of a matrix in its characteristic polynomial results in the zero matrix. Formally, let $A \in \mathbb{F}^{n \times n}$ be a square matrix over a field $\mathbb{F}$.

▶ **Theorem 7** (Cayley-Hamilton). *Let $p(\lambda) = \det(\lambda I_n - A)$ denote the characteristic polynomial of A. Then, $p(A) = 0$.*

Let $\operatorname{sum}(A)$ denote the sum of all entries of A. Recall that the trace of a matrix A, denoted by $\operatorname{tr}(A)$, is the sum of diagonal entries of A. The following lemma is standard.

▶ **Lemma 8.** *Let $A = A(G)$ be the adjacency matrix of a digraph G and $k \geq 1$. The number of walks of length k in A is equal to $\operatorname{sum}(A^k)$. Equivalently, $\operatorname{sum}(A^k) = \mathsf{hom}(P_k, G)$.*

Similarly, the number of closed walks of length k in G is equal to $\operatorname{tr}(A^k)$. Equivalently, $\operatorname{tr}(A^k) = \mathsf{hom}(C_k, G)$.

The following is a consequence of the Cayley-Hamilton theorem and is utilised in Section 6 for our tractability result regarding $\textsc{HomInd}(\mathcal{P}_{\rightarrow})$ for the class $\mathcal{P}_{\rightarrow}$ of directed paths.

▶ **Corollary 9.** *Let $A \in \mathbb{F}^{n \times n}$. There exist $a_1, \dots, a_n \in \mathbb{R}$ such that for every $\ell \geq 0$*

$$A^{n+\ell} = \sum_{i \in [n]} a_i A^{i+\ell-1}.$$

Thus

$$\operatorname{sum}(A^{n+\ell}) = \sum_{i \in [n]} a_i \cdot \operatorname{sum}(A^{i+\ell-1}) \quad \textit{and} \quad \operatorname{tr}(A^{n+\ell}) = \sum_{i \in [n]} a_i \cdot \operatorname{tr}(A^{i+\ell-1}).$$

3 Combinatorial Constructions

Theorem 10 presents our main construction, which is at the heart of both our undecidability results in Section 4 and our hardness results in Section 5. Essentially, for an arbitrary $k \geq 1$, we construct two partial orders that have the same number of directed paths on i vertices for every $i \geq 1$ except for $i = k$. Hence, these partial orders are distinguished by the directed path of length $k - 1$ but not by any other directed path.

▶ **Theorem 10.** *For every $k \geq 1$, there are partial orders P_k, Q_k of order $|P_k|, |Q_k| \leq \max\{1, 2(k-1)\}$ such that*

$$\begin{aligned} d_i(P_k) &= d_i(Q_k) && \textit{for } 1 \leq i \leq k-1, && (1)\\ d_k(P_k) &= 1, \quad d_k(Q_k) = 0, && && (2)\\ d_i(P_k) &= d_i(Q_k) = 0 && \textit{for all } i > k. && (3) \end{aligned}$$

Proof. For a directed graph G, a set $X \subseteq V(G)$, and a fresh vertex $x \notin V(G)$, let $G \triangleright_X x$ be the graph obtained from G by adding vertex x and edges from all vertices in X to x. Formally, $V(G \triangleright_X x) := V(G) \cup \{x\}$ and $E(G \triangleright_X x) := E(G) \cup \{(x', x) \mid x' \in X\}$.

Let us describe the construction of the directed graphs P_k, Q_k. Let v_i for $i \geq 1$ and w_i for $i \geq 2$ be fresh vertices. Let

$$\begin{aligned} V_1 &:= \{v_1\}, & P_1 &:= (V_1, \emptyset),\\ W_1 &:= \emptyset, & Q_1 &:= (W_1, \emptyset),\\ V_2 &:= \{v_1, v_2\}, & P_2 &:= \big(\{V_2, \{(v_1, v_2)\}\big),\\ W_2 &:= \{v_1, w_2\} & Q_2 &:= \big(W_2, \emptyset\big), \end{aligned}$$

and for $k \geq 2$, noting that $W_{k-1} \subseteq V_k$ and $V_{k-1} \subseteq W_k$, let

$$\begin{aligned} V_{k+1} &:= V_k \cup \{v_{k+1}, w_k\}, & P_{k+1} &:= (P_k \triangleright_{V_k} v_{k+1}) \triangleright_{W_{k-1}} w_k, \\ W_{k+1} &:= W_k \cup \{v_k, w_{k+1}\}, & Q_{k+1} &:= (Q_k \triangleright_{W_k} w_{k+1}) \triangleright_{V_{k-1}} v_k. \end{aligned}$$

Figure 1 illustrates the construction for $1 \leq k \leq 4$.

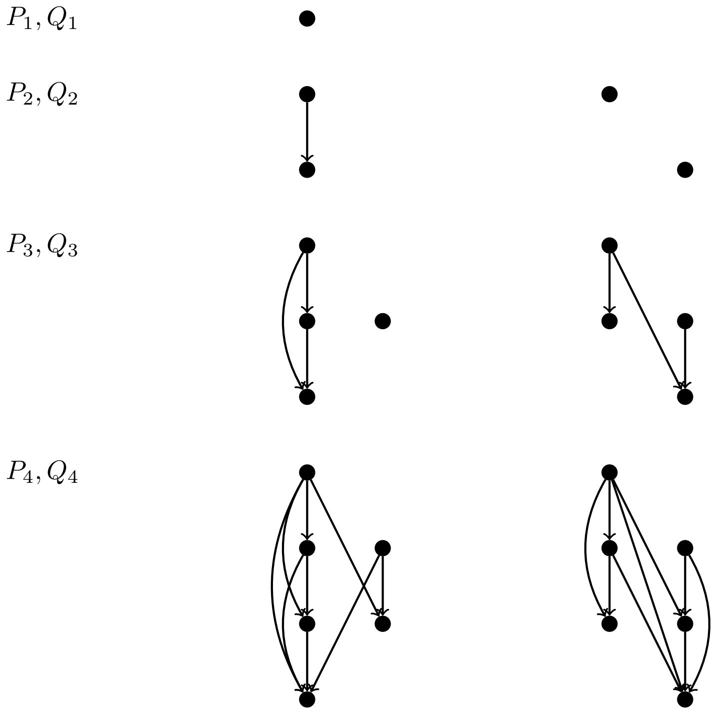


Figure 1 The graphs P_k, Q_k for $k = 1, \dots, 4$.

Note that both P_k and Q_k are directed acyclic graphs. Obviously, we have $|V_k|, |W_k| \leq \max\{1, 2(k-1)\}$ for all k. By $G[X]$, we denote the induced subgraph of a set X in a graph G. Observe that for all $\ell \geq k \geq 1$ we have $P_\ell[X] = P_k[X]$ for all $X \subseteq V_k$ and $Q_\ell[Y] = Q_k[Y]$ for all $Y \subseteq W_k$.

▷ Claim 11. For all $\ell > k \geq 1$ it holds that $P_\ell[W_k] = Q_k$ and $Q_\ell[V_k] = P_k$.

Proof. It suffices to prove the claim for $\ell = k+1$. The proof is by induction on k. For $k = 1, 2$, the assertion is immediate from Figure 1. For the inductive step, let $k \geq 2$. By the induction hypothesis, we have $P_k[W_{k-1}] = Q_{k-1}$. To construct Q_k from Q_{k-1}, we add the vertex w_k and edges from all vertices in W_{k-1} to w_k. Moreover, we add v_{k-1} and edges from all vertices in V_{k-2} to v_{k-1}. Both of these vertices and all these edges are present in P_{k+1}:

- v_{k-1} and edges from all vertices in V_{k-2} to v_{k-1} have already been added in the transition from P_{k-2} to P_{k-1};
- w_k and edges from all vertices in W_{k-1} to w_k are added in the transition from P_k to P_{k+1}.

Thus $P_{k+1}[W_k] = Q_k$. The argument that $Q_{k+1}[V_k] = P_k$ is similar. ◁

Since $V_{k+1} = V_k \cup W_k \cup \{v_{k+1}\}$ and $W_{k+1} = V_k \cup W_k \cup \{w_{k+1}\}$, a consequence of this claim is that

$$P_{k+1} \cap Q_{k+1} = P_{k+1} \setminus \{v_{k+1}\} = Q_{k+1} \setminus \{w_{k+1}\} = P_k \cup Q_k. \tag{4}$$

▷ Claim 12. The edge relations of P_k and Q_k are transitive.

Proof. This follows by induction from (4), because v_k and w_k have out-degree 0 in P_k, Q_k, respectively. ◁

Thus P_k and Q_k are partial orders.

▷ Claim 13. For all graphs G and all subsets $X \subseteq G$ we have

$$c_1\big(G \triangleright_X x\big) = c_1(G) + 1, \tag{5}$$

$$c_i\big(G \triangleright_X x\big) = c_i(G) + c_{i-1}\big(G[X]\big) \qquad \text{for } i \geq 2. \tag{6}$$

Proof. Straightforward. ◁

Thus for $k \geq 2$ and $i \geq 2$ we have

$$c_i(P_{k+1}) = c_i(P_k) + c_{i-1}(P_k) + c_{i-1}(Q_{k-1}) \tag{7}$$

and similarly

$$c_i(Q_{k+1}) = c_i(Q_k) + c_{i-1}(Q_k) + c_{i-1}(P_{k-1}). \tag{8}$$

Now we prove (1)-(3) by induction on k. We have already noted that it holds for $k = 1, 2$. For the inductive step, let $k \geq 2$ and $i \geq 1$.

Case 1: $i = 1$. We have $c_1(P_{k+1}) = |V_{k+1}| = 2k = |W_{k+1}| = c_1(Q_{k+1})$.

Case 2: $2 \leq i \leq k-1$. We have $c_i(P_k) = c_i(Q_k)$, $c_{i-1}(P_k) = c_{i-1}(Q_k)$ and $c_{i-1}(P_{k-1}) = c_{i-1}(Q_{k-1})$ by the induction hypothesis, and by (7) and (8) this implies $c_i(P_{k+1}) = c_i(P_{k+1})$.

Case 3: $i = k$. We have $c_i(P_k) = c_i(Q_k) + 1$ and $c_{i-1}(P_k) = c_{i-1}(Q_k)$ and $c_{i-1}(P_{k-1}) = c_{i-1}(Q_{k-1}) + 1$ by the induction hypothesis. Thus by (7) and (8),

$$\begin{aligned} c_i(P_{k+1}) &= c_i(P_k) + c_{i-1}(P_k) + c_{i-1}(Q_{k-1}) \\ &= c_i(Q_k) + 1 + c_{i-1}(Q_k) + c_{i-1}(P_{k-1}) - 1 = c_i(Q_{k+1}), \end{aligned}$$

which completes the proof of (1).

Case 4: $i = k+1$. We have $c_i(P_k) = c_i(Q_k) = 0$ and $c_{i-1}(P_k) = 1, c_{i-1}(Q_k) = 0$ and $c_{i-1}(P_{k-1}) = c_{i-1}(Q_{k-1}) = 0$ by the induction hypothesis. Thus by (7) and (8),

$$\begin{aligned} c_i(P_{k+1}) &= c_i(P_k) + c_{i-1}(P_k) + c_{i-1}(Q_{k-1}) = 1, \\ c_i(Q_{k+1}) &= c_i(Q_k) + c_{i-1}(Q_k) + c_{i-1}(P_{k-1}) = 0. \end{aligned}$$

This proves (2).

Case 5: $i \geq k+2$. We have $c_i(P_k) = c_i(Q_k) = c_{i-1}(P_k) = c_{i-1}(Q_k) = c_{i-1}(P_{k-1}) = c_{i-1}(Q_{k-1}) = 0$ by the induction hypothesis. Thus by (7) and (8), $c_i(P_{k+1}) = c_i(Q_{k+1}) = 0$. This proves (3).

Now assertions (1)–(3) follow, because for partial orders G we have $c_k(G) = d_k(G)$ for all $k \geq 1$. ◀

As the construction of Theorem 10 yields partial orders, for which directed path counts and clique counts are the same, we also obtain an undirected version for clique counts with Corollary 14.

▶ **Corollary 14.** *For every $k \geq 1$ there are undirected graphs G_k, H_k of order $|G_k|, |H_k| \leq \max\{1, 2(k-1)\}$ such that*

$$c_i(G_k) = c_i(H_k) \qquad \textit{for } 1 \leq i \leq k-1, \tag{9}$$

$$c_k(G_k) = 1, \quad c_k(H_k) = 0, \tag{10}$$

$$c_i(G_k) = c_i(H_k) = 0 \qquad \textit{for all } i > k. \tag{11}$$

Proof. Let G_k, H_k be the undirected graphs underlying the partial orders P_k, Q_k of Theorem 10. ◀

We also show how to generalise Theorem 10 and Corollary 14 such that number of directed paths of length $k-1$ and the number of cliques of size k in these graphs, respectively, can freely be chosen. The exact statements and their proofs can be found in the full version of the paper.

4 Undecidability Results

We proceed to derive undecidability results for HomInd($\mathcal{F}$) using the combinatorial constructions of Section 3. Before we prove our main theorem (Theorem 1), the following version for the case of directed graphs will be necessary.

▶ **Theorem 15.** *There is a polynomial time decidable class $\mathcal{F}$ of directed graphs of tree width 1 such that* HomInd($\mathcal{F}$) *is undecidable.*

Proof. Let us fix some Gödel numbering of deterministic Turing machines such that M_j denotes the machine with Gödel number j. For every pair $(j, t) \in \mathbb{N}^2$, let $F_{j,t}$ be the graph that is the disjoint union of a directed path $P^{\to}_{j+1}$ of length $(j+1)$ and an independent set of size t. Let $\mathcal{F}$ be the class of all graphs $F_{j,t}$ such that M_j halts in t steps when started on the empty input word. Clearly, $\mathcal{F}$ is decidable in polynomial time.

Observe that for every graph G and all $j, t \in \mathbb{N}$ we have

$$\mathsf{hom}(F_{j,t}, G) = \mathsf{hom}(P_{j+1}, G) \cdot |G|^t. \tag{12}$$

Let G_k, H_k be the graphs constructed in Theorem 10. It is easy to note that $\mathsf{hom}(F_{0,t}, G_1) = 1 \neq 0 = \mathsf{hom}(F_{0,t}, H_1)$ and $\mathsf{hom}(F_{j,t}, G_1) = \mathsf{hom}(F_{j,t}, H_1) = 0$ for $j \geq 1$. For $k \geq 2$, recall that $|G_k| = |H_k| = 2(k-1)$ and that $\mathsf{hom}(P^{\to}_j, G_k) = \mathsf{hom}(P^{\to}_j, H_k)$ if and only if $j \neq k$. Thus it follows from (12) that for all j, t we have

$$\mathsf{hom}(F_{j,t}, G_k) \neq \mathsf{hom}(F_{j,t}, H_k) \iff j = k-1. \tag{13}$$

This implies

$$\begin{aligned} \mathsf{Hom}_{\mathcal{F}}(G_k) \neq \mathsf{Hom}_{\mathcal{F}}(H_k) &\iff F_{(k-1),t} \in \mathcal{F} \text{ for some } t \in \mathbb{N} \\ &\iff M_{k-1} \text{ halts on the empty input word.} \end{aligned}$$

This proves that HomInd($\mathcal{F}$) is undecidable. Since every graph in $\mathcal{F}$ is a directed path with some padded isolated vertices, the treewidth of $\mathcal{F}$ is 1. Hence, proved. ◀

The above proof can be easily modified to invoke Corollary 14 instead of Theorem 10, which yields the following corollary.

▶ **Corollary 16.** *There is a polynomial time decidable class $\mathcal{F}$ of undirected graphs such that* HomInd($\mathcal{F}$) *is undecidable.*

In particular, every graph in this class of undirected graphs is a clique (padded with isolated vertices), and therefore, this class has unbounded treewidth. Our main theorem, Theorem 1, is a sharper version of the above corollary.

4.1 Proof of Theorem 1

The proof of Theorem 1 is obtained by replacing the directed graphs $F_{j,t}, G_k, H_k$ (constructed in the proof of Theorem 15) by their suitably-defined undirected versions $\widetilde{F}_{j,t}, \widetilde{G}_k, \widetilde{H}_k$. The two key requirements of this transformation are (a) the homomorphism numbers are preserved, i.e., $\mathsf{hom}(F_{j,t}, G_k) = \mathsf{hom}(\widetilde{F}_{j,t}, \widetilde{G}_k)$ and $\mathsf{hom}(F_{j,t}, H_k) = \mathsf{hom}(\widetilde{F}_{j,t}, \widetilde{H}_k)$, and (b) the treewidth of the transformed graphs is bounded. To satisfy these properties, we devise suitable gadgets to encode the direction of an edge: our gadgets employ homomorphically incomparable Kneser graphs in order to control the homomorphism numbers of the resulting undirected graphs. The end result of our gadget construction is the following technical lemma (note that our construction there does not exploit any special properties of the present graphs and works for arbitrary directed graphs). The proof is deferred to the full version of the paper.

Let $\mathcal{F}$ be the class of all directed graphs $F_{j,t}$ arising in the proof of Theorem 15, such that $\textsc{HomInd}(\mathcal{F})$ is undecidable.

▶ **Lemma 17.** *Given graphs $F_{j,t}, G_k, H_k$ (constructed in the proof of Theorem 15), we can construct corresponding undirected graphs $\widetilde{F}_{j,t}, \widetilde{G}_k, \widetilde{H}_k$ in polynomial time, satisfying the following properties.*

1. *For all $j, t \in \mathbb{N}$ there exists a positive integer $C_{j,t}$ such that for every $k \in \mathbb{N}$,*

$$\mathsf{hom}(\widetilde{F}_{j,t}, \widetilde{G}_k) = C_{j,t} \cdot \mathsf{hom}(F_{j,t}, G_k),$$
$$\mathsf{hom}(\widetilde{F}_{j,t}, \widetilde{H}_k) = C_{j,t} \cdot \mathsf{hom}(F_{j,t}, H_k).$$

2. *There exists a fixed positive integer ℓ such that for every $j, t \in \mathbb{N}$, the graph $\widetilde{F}_{j,t}$ has treewidth at most ℓ.*
3. *Let $\widetilde{\mathcal{F}}$ be the class of all undirected graphs $\widetilde{F}_{j,t}$ such that $F_{j,t} \in \mathcal{F}$. Then $\widetilde{\mathcal{F}}$ is polynomial time decidable.*

From the above lemma, we can deduce Theorem 1 as follows.

Proof of Theorem 1. The proof follows the proof of Theorem 15 closely. By Lemma 17 and Equation 13,

$$\mathsf{hom}(\widetilde{F}_{j,t}, \widetilde{G}_k) \neq \mathsf{hom}(\widetilde{F}_{j,t}, \widetilde{H}_k) \iff j = k - 1. \tag{14}$$

This implies

$$\begin{aligned}\mathsf{Hom}_{\widetilde{\mathcal{F}}}(\widetilde{G}_k) \neq \mathsf{Hom}_{\widetilde{\mathcal{F}}}(\widetilde{H}_k) &\iff \widetilde{F}_{(k-1),t} \in \widetilde{\mathcal{F}} \text{ for some } t \in \mathbb{N}\\ &\iff F_{(k-1),t} \in \mathcal{F} \text{ for some } t \in \mathbb{N}\\ &\iff M_{k-1} \text{ halts on the empty input word.}\end{aligned}$$

This proves that $\textsc{HomInd}(\widetilde{\mathcal{F}})$ is undecidable. By Lemma 17, $\widetilde{\mathcal{F}}$ has bounded treewidth and is polynomial-time-decidable. Hence, proved. ◀

5 Hardness for Cliques

The construction of Corollary 14 allows for a simple proof of the co-NP-hardness of $\textsc{HomInd}(\mathcal{K})$.

Proof of Theorem 2. We reduce the co-NP-hard problem

$$\overline{\textsc{Clique}} = \{(F,k) \mid F \text{ graph without a } k\text{-clique}\}$$

to $\textsc{HomInd}(\mathcal{K})$. To this end, we use Corollary 14 to construct two graphs G_k and H_k in time polynomial in $k \leq n$. Then, the graphs

$$F_1 := F \otimes G_k \quad \text{and} \quad F_2 := F \otimes H_k.$$

can be constructed in time $\text{poly}(n)$. Corollary 5, together with (9)-(11), implies that

$$c_i(F_1) = c_i(F_2) \qquad \text{for } 1 \leq i \leq k-1, \tag{15}$$

$$c_k(F_1) = k! \cdot c_k(F), \quad c_k(F_2) = 0, \tag{16}$$

$$c_i(F_1) = c_i(F_2) = 0 \qquad \text{for all } i > k. \tag{17}$$

Hence, the mapping $F \mapsto (F_1, F_2)$ is the desired polynomial-time many-one reduction as $c_k(F) = 0$ if and only if $c_i(F_1) = c_i(F_2)$ for every $i \geq 1$. ◀

A more refined argument gives us a more precise classification of the complexity of $\textsc{HomInd}(\mathcal{K})$.

▶ **Theorem 18.** $\textsc{HomInd}(\mathcal{K})$ *is complete for* $\mathsf{C_{=}P}$ *under polynomial time Turing reductions.*

The complexity class $\mathsf{C_{=}P}$ was introduced in [18, 22]. Here we use the following equivalent definition from [10].

▶ **Definition 19.** *Let L be a decision problem. Then $L \in \mathsf{C_{=}P}$ if and only if there is a function f in $\#\mathrm{P}$ and a function g computable in polynomial time such that for every instance x of L*

$$x \in L \iff f(x) = g(x).$$

Before giving the proof of Theorem 18, we derive some of its consequences, which shows that $\textsc{HomInd}(\mathcal{K})$ is apparently much harder than co-NP as stated in Theorem 2, and in fact, unlikely to be in the polynomial hierarchy (PH).

It is clear that every problem in $\mathsf{C_{=}P}$ can be decided in polynomial time with an oracle to a problem in #P. The following slightly weak converse is also easy to see (cf. [4], Section 1.2).

▶ **Proposition 20.** $\mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{NP}^{\mathsf{C_{=}P}}$.

Thus Theorem 18 implies that:

▶ **Corollary 21.** $\mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{NP}^{\textsc{HomInd}(\mathcal{K})}$.

Combined with renowned Toda's Theorem [19], we conclude that $\textsc{HomInd}(\mathcal{K})$ is above the polynomial hierarchy.

▶ **Corollary 22.** $\textsc{HomInd}(\mathcal{K})$ *is not in* PH, *unless* PH *collapses.*

Now we are ready to prove Theorem 18: the proof is deferred to the full version of the paper.

6 Tractable Cases

▶ **Theorem 23.** $\textsc{HomInd}(\mathcal{F})$ *can be solved in polynomial time for the class* $\mathcal{F}$ *of directed paths.*

The proof of the theorem is deferred to the full version of the paper.

7 Related Problems

For a class of graphs $\mathcal{F}$, let $\mathcal{F}_k$ denote the class of all graphs $F \in \mathcal{F}$ of order k. For classes $\mathcal{F}$ where there is at most one graph $F \in \mathcal{F}$ of order k for every $k \geq 1$, it is only natural to consider the entries $\mathsf{hom}(F, G)$ of a homomorphism vector $\mathsf{Hom}_{\mathcal{F}}(G)$ as sorted by the order of F. Then it becomes possible to compare these vectors using the lexicographical ordering $\leq_\ell$, and for such a class $\mathcal{F}$, we let $\textsc{HomLex}(\mathcal{F})$ denote the following decision problem: given graphs G and H, decide whether $\mathsf{Hom}_{\mathcal{F}}(G) \leq_\ell \mathsf{Hom}_{\mathcal{F}}(H)$.

For the classes of directed paths and directed cycles, the proof of Theorem 23 immediately yields that the corresponding decision problems can be decided in polynomial time as, given graphs G and H with $|V(G)| = |V(H)| = n$, it suffices to consider the first $2n$ paths and n cycles, respectively.

▶ **Theorem 24.** $\textsc{HomLex}(\mathcal{F})$ *can be solved in polynomial time for the class of directed paths and the class of directed cycles.*

For the class of all complete graphs, the hardness of the $\textsc{HomLex}(\mathcal{K})$ is expected.

▶ **Theorem 25.** $\textsc{HomLex}(\mathcal{K})$ *is complete for* $\#\mathrm{P}$ *under polynomial time Turing reductions.*

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

A more interesting direction is to consider the similarity of homomorphism vectors as this induces a measure of similarity on the graphs themselves (see e.g. [13], Lemma 10.22 and Lemma 10.32). It is not clear how exactly the distance of homomorphism vectors should be defined, but some natural candidates are the following:

(a) $$d^1_{\mathcal{F}}(G, H) := \sum_{\substack{k \geq 1 \\ \mathcal{F}_k \neq \emptyset}} \frac{1}{k^k |\mathcal{F}_k|} \sum_{F \in \mathcal{F}_k} |\mathsf{hom}(F, G) - \mathsf{hom}(F, H)|$$

(b) $$d^\infty_{\mathcal{F}}(G, H) := \sum_{\substack{k \geq 1 \\ \mathcal{F}_k \neq \emptyset}} \frac{1}{k^k} \max_{F \in \mathcal{F}_k} |\mathsf{hom}(F, G) - \mathsf{hom}(F, H)|$$

(c) $$d^2_{\mathcal{F}}(G, H) := \sqrt{\sum_{\substack{k \geq 1 \\ \mathcal{F}_k \neq \emptyset}} \frac{1}{k^k |\mathcal{F}_k|} \sum_{F \in \mathcal{F}_k} (\mathsf{hom}(F, G) - \mathsf{hom}(F, H))^2}$$

The scaling by k^k in the definitions is quite arbitrary and ensures that the sums converge: Let G and H be graphs with $|V(G)| \geq |V(H)|$. As the number of homomorphisms from a k-vertex graph to an n-vertex graph is at most n^k, we for example have

$$\sqrt[k]{\frac{1}{k^k |\mathcal{F}_k|} \sum_{F \in \mathcal{F}_k} |\mathsf{hom}(F, G) - \mathsf{hom}(F, H)|} \leq \sqrt[k]{\frac{|V(G)|^k}{k^k}} = \frac{|V(G)|}{k} \xrightarrow{k \to \infty} 0,$$

which implies convergence of the sum in the definition of $d^1_{\mathcal{F}}$ by the root test. Of course, one may also scale by a different factor, e.g., $k!$, and possibly even make it depend on the orders of G and H. It is not hard to see that $d^1_{\mathcal{F}}$, $d^\infty_{\mathcal{F}}$, and $d^2_{\mathcal{F}}$ are pseudometrics on graphs. For $d^2_{\mathcal{F}}$, this follows directly from the fact that it is the pseudometric induced by an inner product on homomorphism vectors proposed by Dell, Grohe, and Rattan [6].

For "simple-enough" classes of graphs, these pseudometrics can be computed up to an arbitrarily small additive error ε in polynomial time in the straightforward way: compute "enough" terms by computing the first graphs in $\mathcal{F}$ and then counting homomorphisms from them one by one. To this end, we call a class of graphs $\mathcal{F}$ polynomial-time enumerable if there is a polynomial-time algorithm that, on input 1^k, outputs all graphs in $\mathcal{F}_k$. Theorem 26 only considers $d^1_{\mathcal{F}}$ for simplicity, but the calculations can directly be adapted to $d^\infty_{\mathcal{F}}$ and $d^2_{\mathcal{F}}$. The proof is deferred to the full version of the paper.

▶ **Theorem 26.** *Let $\mathcal{F}$ be a polynomial-time enumerable class of graphs for which $\#\textsc{Hom}(\mathcal{F})$ is in polynomial time. Then, for every $\varepsilon > 0$, there is a polynomial-time algorithm $D^\varepsilon_{\mathcal{F}}$ that takes two graphs G and H as input and outputs a real number $D^\varepsilon_{\mathcal{F}}(G, H)$ such that $|d^1_{\mathcal{F}}(G, H) - D^\varepsilon_{\mathcal{F}}(G, H)| \leq \varepsilon$ holds for all G and H.*

Among the classes to which Theorem 26 applies are the class $\mathcal{C}$ of all cycles, the class $\mathcal{P}$ of all paths, and for every fixed $d \geq 1$, the class of all complete d-ary trees.

8 Conclusion

We established the rich complexity-theoretic behaviour of the problem $\textsc{HomInd}(\mathcal{F})$ for a variety of graph classes. It was already known that this problem can be solved in polynomial time for the class of paths, trees and bounded treewidth graphs. Our results are complementary: there exist polynomial-time-decidable graph classes for which this problem is undecidable, even though these graph classes satisfy strong structural restrictions (such as bounded treewidth). For the class of complete graphs and directed paths, we also provide tight upper and lower bounds for the complexity of $\textsc{HomInd}(\mathcal{F})$. Our techniques rely on combinatorial constructions of graphs with almost-identical homomorphism vectors: these constructions might be of independent interest.

Perhaps the most interesting direction of further work is the study of graph metrics induced by homomorphism vectors. These metrics induce natural measures of similarity between graphs: such measures serve as an important black-box component for the design of practical graph learning algorithms. Therefore, a better understanding of these metrics will enable a theoretical analysis of practical graph learning tools.

References

1 V. Arvind, J. Köbler, S. Kuhnert, and Y. Vasudev. Approximate Graph Isomorphism. In B. Rovan, V. Sassone, and P. Widmayer, editors, *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science*, volume 7464 of *Lecture Notes in Computer Science*, pages 100–111. Springer Verlag, 2012.

2 L. Babai. Graph Isomorphism in Quasipolynomial Time. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC '16)*, pages 684–697, 2016.

3 J. Böker. Color Refinement, Homomorphisms, and Hypergraphs. *arXiv e-prints*, page arXiv:1903.12432, March 2019.

4 R. Curticapean. Parity Separation: A Scientifically Proven Method for Permanent Weight Loss. In I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, editors, *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (Track A)*, volume 55 of *LIPIcs*, pages 47:1–47:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

5 V. Dalmau and P. Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1-3):315–323, 2004.

6 H. Dell, M. Grohe, and G. Rattan. Lovász Meets Weisfeiler and Leman. In I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, editors, *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (Track A)*, volume 107 of *LIPIcs*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

7 R. Diestel. *Graph Theory*. Springer Verlag, 4th edition, 2010.

8 M. Grohe, K. Kersting, M. Mladenov, and P. Schweitzer. Color Refinement and its Applications. In G. Van den Broeck, K. Kersting, S. Natarajan, and D. Poole, editors, *An Introduction to Lifted Probabilistic Inference*. Cambridge University Press, 2017. To appear. URL: `https://lii.rwth-aachen.de/images/Mitarbeiter/pub/grohe/cr.pdf`.

9 M. Grohe, G. Rattan, and G. Woeginger. Graph Similarity and Approximate Isomorphism. In I. Potapov, P.G. Spirakis, and J. Worrell, editors, *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science*, volume 117 of *LIPIcs*, pages 20:1–20:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

10 L. A. Hemaspaandra and H. Vollmer. The satanic notations: counting classes beyond #P and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.

11 A. Kolla, I. Koutis, V. Madan, and A.K. Sinop. Spectrally Robust Graph Isomorphism. In I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, editors, *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs*, pages 84:1–84:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

12 L. Lovász. Operations with Structures. *Acta Mathematica Hungarica*, 18:321–328, 1967.

13 L. Lovász. *Large Networks and Graph Limits*. American Mathematical Society, 2012.

14 V. Nagarajan and M. Sviridenko. On the maximum quadratic assignment problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 516–524, 2009.

15 R. O'Donnell, J. Wright, C. Wu, and Y. Zhou. Hardness of Robust Graph Isomorphism, Lasserre Gaps, and Asymmetry of Random Graphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1659–1677, 2014.

16 N. Shervashidze, P. Schweitzer, E.J. van Leeuwen, K. Mehlhorn, and K.M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

17 N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009.

18 J. Simon. *On Some Central Problems in Computational Complexity*. PhD thesis, Cornell University, 1975.

19 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

20 E. R. Van Dam and W. H. Haemers. Which graphs are determined by their spectrum? *Linear Algebra and its applications*, 373:241–272, 2003. `doi:10.1016/S0024-3795(03)00483-X`.

21 S.V.N. Vishwanathan, N.N. Schraudolph, R. Kondor, and K.M. Borgwardt;. Graph Kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

22 K. W. Wagner. Some Observations on the Connection Between Counting an Recursion. *Theoretical Computuer Science*, 47(3):131–147, 1986.

SZX-Calculus: Scalable Graphical Quantum Reasoning

Titouan Carette
CNRS, LORIA, Inria Mocqua, Université de Lorraine, F 54000 Nancy, France
titouan.carette@loria.fr

Dominic Horsman
LIG, Université Grenoble Alpes, France
dominic.horsman@univ-grenoble-alpes.fr

Simon Perdrix
CNRS, LORIA, Inria Mocqua, Université de Lorraine, F 54000 Nancy, France
https://members.loria.fr/SPerdrix/
simon.perdrix@loria.fr

Abstract

We introduce the Scalable ZX-calculus (SZX-calculus for short), a formal and compact graphical language for the design and verification of quantum computations. The SZX-calculus is an extension of the ZX-calculus, a powerful framework that captures graphically the fundamental properties of quantum mechanics through its complete set of rewrite rules. The ZX-calculus is, however, a low level language, with each wire representing a single qubit. This limits its ability to handle large and elaborate quantum evolutions. We extend the ZX-calculus to registers of qubits and allow compact representation of sub-diagrams via binary matrices. We show soundness and completeness of the SZX-calculus and provide two examples of applications, for graph states and error correcting codes.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Quantum computing, categorical quantum mechanics, completeness, scalability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.55

Related Version Full proofs for this paper can be found at arXiv:1905.00041 [7].

Funding This work is funded by ANR-17-CE25-0009 SoftQPro, PIA-GDN/Quantex, and LUE project UOQ.

1 Introduction

The ZX-calculus is an intuitive and powerful graphical language for quantum computing, introduced by Coecke and Duncan [11]. Quantum processes can be represented by ZX-diagrams, which can be seen intuitively as a generalisation of quantum circuits. The language is also equipped with a set of rewrite rules which preserves the represented quantum evolution. Unlike quantum circuits, the ZX-calculus has been proved to be complete for various universal fragments of pure quantum mechanics [26, 23, 27, 28, 43], and also mixed states quantum mechanics [8]. Completeness means that any equality can be derived in this language: if two diagrams represent the same quantum process then they can be transformed one into the other using the rewriting rules of the language. Completeness opens avenues for various applications of the ZX-calculus in quantum information processing, including circuit optimisation [16, 30] – which out-performs all other technics for T-count reductions [34] – error correcting codes [17, 21, 9], lattice surgery [14], measurement-based quantum computing [19, 15, 32] *etc.* Automated tools for quantum reasoning, e.g. Quantomatic [35] and PyZX [33], are also based on the ZX-calculus. The ZX-calculus is also used as intermediate representation in a commercial quantum compiler [12].

The cornerstone of the ZX-calculus is that fundamental properties of quantum mechanics can be captured graphically. The language remains, however, relatively low level: each wire represents a single qubit, a feature that limits the design of larger-scale and more complex quantum procedures. We address in this paper the problem of scalability of the ZX-calculus. In [9], the authors – including one of the present paper – demonstrated that the ZX-calculus can be used in practice to design and verify quantum error correcting codes. They introduced various shortcuts to deal with the scalability of the language: mainly the use of thick wires to represent registers of qubits and matrices to represent sub-diagrams, and hence reason about families of diagrams in a compact way. However, the approach lacked a general theory and fundamental properties like soundness and completeness.

Contributions. We introduce the Scalable ZX-calculus, SZX calculus for short, to provide theoretical foundations to this approach. We extend the ZX-calculus to deal with registers of qubits by introducing some new generators and rewrite rules. We show soundness – i.e. the new generators can be used in a consistent way – as well as completeness of the SZX-calculus. A simple but key ingredient is the introduction of two generators, not present in [9], for dividing and gathering registers of qubits. A wire representing a register of $(n{+}m)$-qubits can be divided into two wires representing respectively n and m qubits. Similarly two registers can be gathered into a single larger one. We also extend the generators of the ZX-calculus so that they can act not only on a single qubit but on a register of qubits. The SZX-calculus is then constructed as a combination of the ZX-calculus and the sub-language made of the divider and the gatherer, by adding the necessary rewrite rules describing how these two sub-languages interact. We show that the SZX-calculus is universal, sound, and complete, providing an intuitive and formal language to represent quantum operations on an arbitrarily large finite number of qubits. The use of the divider and the gatherer allows one to derive inductive (graphical) proofs.

Furthermore, the SZX-calculus provides the fundamental structures – namely the (co)comutative Hopf algebras – to develop a graphical theory of binary matrices, following work on graphical linear algebra [5]. As a consequence, we introduce an additional generator parametrized by a binary matrix together with four simple rewrite rules. Note that, while matrices were also used in [9], we introduce here a more elementary generator acting on a single register (1 input/1 output) rather than two registers (2 inputs/2 outputs). We prove completeness of the SZX-calculus augmented with these matrices. The use of matrices allows a compact representation where subdiagrams can be replaced by matrices. Moreover, basic matrix arithmetic can be done graphically. It makes the SZX-calculus with matrices a powerful tool for formal and compact quantum reasoning.

In section 5, we show the SZX-calculus in action. The main application of the SZX-calculus we consider in this paper is the graph state formalism [24]. We show how graph states can be represented using SZX-diagrams and how some fundamental properties like fixpoint properties, local complementation, and pivoting can be derived in the calculus. We also consider error correcting code examples in order to show that the techniques for the design and verification of codes developed in [9] can be performed smoothly in the SZX-calculus.

Related works. Scalability is crucial in the development of the ZX-calculus and more generally for graphical languages. We review here some contributions in this domain that we briefly compare to our approach.

The !-boxes formalism [31] is a meta language for graphical languages, which has been extensively used in the development of the automated tool Quantomatic. A !-box is a region (subdiagram) of a diagram which can be discarded or duplicated. There is also a first order logic handling families of equations between concrete (i.e. !-box free) diagrams. In contrast, the scalable ZX is not a meta-language but an actual graphical language equipped with an equational theory (namely a coloured PROP). There is no obvious way to compare these two approaches (even in terms of expressive power).

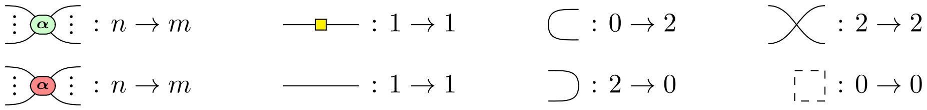
Monoidal multiplexing [10] corresponds to two categorical constructions which allow representing n diagrams in parallel. Roughly speaking, one of the two constructions would be equivalent to the use of big wires for the subclass of SZX-diagrams which are matrix, divider and gatherer-free. It is worth noticing that, to our knowledge, monoidal multiplexing has never been combined with the matrix approach, even though both were developed in the same line of research on graphical linear algebra.

Recently, Miatto [39] has independently introduced a graphical calculus involving matrices, and the equivalent of green spiders, dividers and gatherers. This graphical calculus has been developed in the context of the tensor networks, and the author mainly shows that 6 kinds of matrix products can be represented graphically. We note that the represented matrices do not coincide with the ones we are axiomatising: the matrices represented in Miatto's language correspond to $\mathbb{C}^{2^m\times 2^n}$ matrices whereas ours are in $\mathbb{F}_2^{m\times n}$, hence the equations differ. It is however worth noting that equation Fig.6 in [39] essentially corresponds to the equation governing the interaction between green spiders and the divider given in section 3.3.

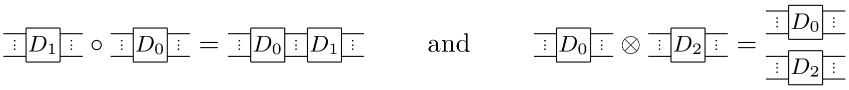
Structure of the paper. We first present the ZX-calculus in section 2, then we introduce the SZX-calculus in section 3, and an axiomatisation of binary matrices for compressing diagrams in section 4. Finally, in section 5, we use the SZX-calculus for error correcting codes and graph states. Full proofs for this paper can be found at arXiv:1905.00041 [7].

2 Background: the ZX-calculus

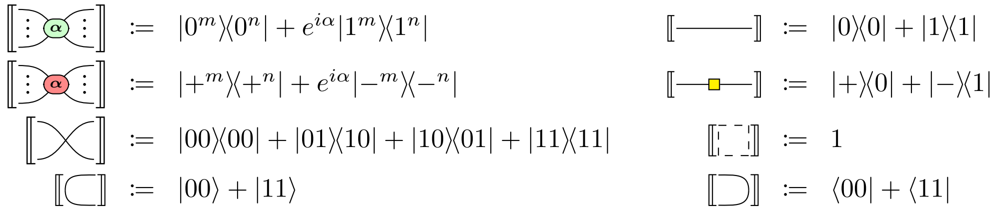
A ZX-diagram $D : k \to \ell$ with k inputs and ℓ outputs is generated by: $\forall n, m \in \mathbb{N}$, $\forall \alpha \in \mathbb{R}$,



and the two compositions: for any ZX-diagrams $D_0 : a \to b$, $D_1 : b \to c$, and $D_2 : c \to d$:

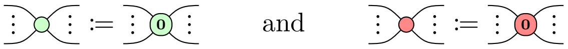


For any n, m, $ZX[n, m]$ is the set of all ZX-diagrams of type $n \to m$. The ZX-diagrams are representing quantum processes: for any ZX-diagram $D : n \to m$ its interpretation $[\![D]\!] \in \mathcal{M}_{2^m\times 2^n}(\mathbb{C})$ is inductively defined as: $[\![D_1 \circ D_0]\!] = [\![D_1]\!] \circ [\![D_0]\!]$, $[\![D_0 \otimes D_2]\!] = [\![D_0]\!] \otimes [\![D_2]\!]$, and


α
$:= \quad |0^m\rangle\langle 0^n| + e^{i\alpha}|1^m\rangle\langle 1^n|$
$:= \quad |0\rangle\langle 0| + |1\rangle\langle 1|$
α
$:= \quad |+^m\rangle\langle +^n| + e^{i\alpha}|-^m\rangle\langle -^n|$
$:= \quad |+\rangle\langle 0| + |-\rangle\langle 1|$
$:= \quad |00\rangle\langle 00| + |01\rangle\langle 10| + |10\rangle\langle 01| + |11\rangle\langle 11|$
$:= \quad 1$
$:= \quad |00\rangle + |11\rangle$
$:= \quad \langle 00| + \langle 11|$

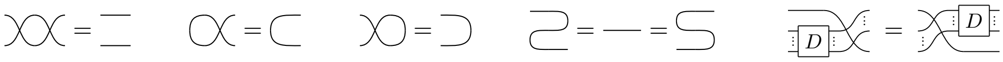
Where $|0\rangle{:=}\binom{1}{0}$, $|1\rangle{:=}\binom{0}{1}$, $|+\rangle{:=}\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, $|-\rangle{:=}\frac{|0\rangle-|1\rangle}{\sqrt{2}}$, $|a^{k+1}\rangle{:=}|a\rangle\otimes|a^k\rangle$, $|a^0\rangle{:=}1$, and $\langle a|{:=}|a\rangle^\dagger$, moreover n and m are respectively the number of inputs and outputs of the spiders.

When equal to zero, the angle of the green or red spider is omitted:



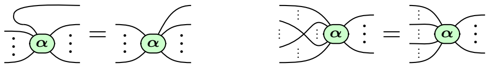
ZX-diagrams are *universal* for pure qubit quantum mechanics: $\forall n, m \in \mathbb{N}$, and $\forall M \in \mathcal{M}_{2^n \times 2^m}(\mathbb{C})$, there exists a ZX-diagram $D : n \to m$ such that $[\![D]\!] = M$.

ZX-diagrams also come with a set of graphical rewrite rules, or axioms, which allows one to transform a diagram preserving its interpretation. Some of them are gathered under the *Only Topology Matters* paradigm. When using these we label the equality by top. Two diagrams that can be transformed into each other by moving around the wires are equal. This can be derived from the following rules:



The last set of rule expresses the naturality of the swap; in other words, that all the generators can be passed through wires.

The legs of the spiders of ZX-calculus can be exchanged and bent. This implies that diagrams are essentially graphs with inputs and outputs.



Finally, the rules that are not purely topological are given in Figure 1.

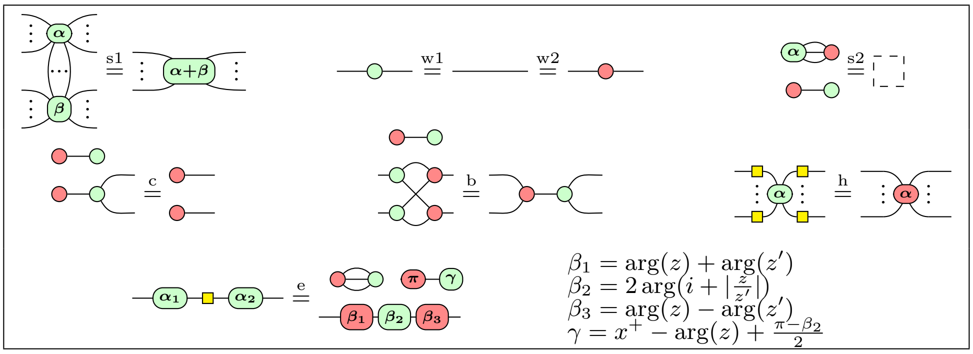


Figure 1 Axioms of the ZX-calculus, where $x^+ := \frac{\alpha_1+\alpha_2}{2}$, $x^- := x^+ - \alpha_2$, $z := -\sin(x^+) + i\cos(x^-)$, $z' := \cos(x^+) - i\sin(x^-)$ and $z' = 0 \Rightarrow \beta_2 = 0$. In the upper left rule, there must be at least one wire between the spiders annotated by α and β. The colour-swapped version of those rules also holds. A label is given to each axiom, above the equals sign, for later reference.

We write $ZX \vdash D = D'$ when D can be transformed into D' using the rules of the ZX-calculus. The rules of the ZX-calculus are sound: for any ZX-diagrams D, D', $ZX \vdash D = D' \Rightarrow [\![D]\!] = [\![D']\!]$ i.e., the rules preserve the interpretation of the language. The language is also complete: for any ZX-diagrams D, D', $[\![D]\!] = [\![D']\!] \Rightarrow ZX \vdash D = D'$ i.e., whenever two diagrams represent the same quantum evolution, we can transform one into the other using the rules of the language [43].

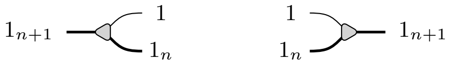
3 The scalable ZX-calculus

In the ZX-calculus, each wire represents a single qubit. Therefore, a system acting on n qubits will be represented by an n-input diagram. This quickly leads to intractable diagrams when it comes to big systems. The extension to the SZX-calculus presented here provides a more compact notation.

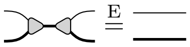
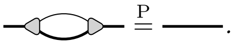
3.1 Divide and gather, a calculus for big wires

The input (resp. output) type of a ZX-diagram is its number of input wires, and hence number of input qubits. In the SZX-calculus, wires represent registers of qubits. A wire of type 1_n represents a register of n qubits. A type of the SZX-calculus is then a formal sum of the form $\sum_i 1_{n_i}$, the empty sum being denoted by 0. In other words, the set of types of SZX-calculus is the free monoid over $\mathbb{N}^*$, the set of positive integers. We denote it $\langle \mathbb{N}^* \rangle$. Graphically, we represent the wire of type 1_n by an bold font wires labelled by n[1], a label that is omitted when it is not ambiguous. A normal font wire always denotes a single qubit register of type 1_1. By convention the sum of m wires of type 1_n is denoted m_n with $0_n = m_0 = 0$. n_1 is simply written n. Given a type $a = \sum_i 1_{n_i}$, its size is defined as $S(a) := \sum_i n_i$.

Big wires can be divided into smaller ones and, conversely, can be gathered to form bigger ones. For any $n \in \mathbb{N}$, we introduce two new generators: the *divider* and *gatherer* of size n. They are depicted as follows:



We take the convention that the divider and the gatherer of size 0 are the identity. We define a fragment of the SZX, the wire calculus $\mathbb{W}$.

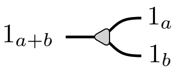
▶ **Definition 1** ($\mathbb{W}$-calculus). *The $\mathbb{W}$-calculus is defined as the graphical language generated by identity wires, the dividers, and the gatherers of any size, and satisfying the elimination rule*  *and the expansion rule* .

The roles of the dividers and gatherers in the equations are perfectly symmetric, so each time something is shown for dividers it also holds for gatherers by symmetry.

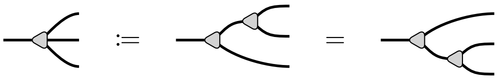
We now show a coherence theorem for scalable calculi: the rewiring theorem. It states that two diagrams of the $\mathbb{W}$-calculus with the same type are equal.

▶ **Theorem 2.** *Let $\omega \in \mathbb{W}[a,b]$ and $\omega' \in \mathbb{W}[c,d]$:* $\quad \mathbb{W} \vdash \omega = \omega' \quad \Leftrightarrow \quad a = c$ *and* $b = d$

This theorem has strong consequences. We can define generalized dividers able to divide any wire of size 1_{a+b} into a wire of size 1_a and a wire of size 1_b.



Those generalized dividers have a unique possible interpretation as diagrams of $\mathbb{W}$-calculus given by their types, and we know exactly the equations they verify: all the well typed ones. In particular, an associativity-like law holds for generalized wires allowing us to define n-ary generalized dividers.



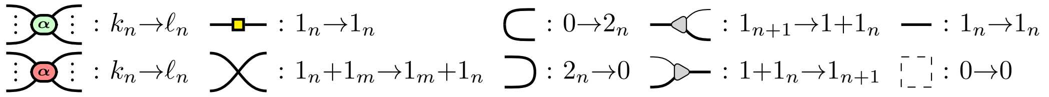
Each time we use the property that any well typed equation in $\mathbb{W}$ is true, we will label the equality by R.

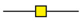
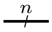
[1] On a blackboard the bold font might be advantageously replaced by struck-out wires.

3.2 The SZX-diagrams

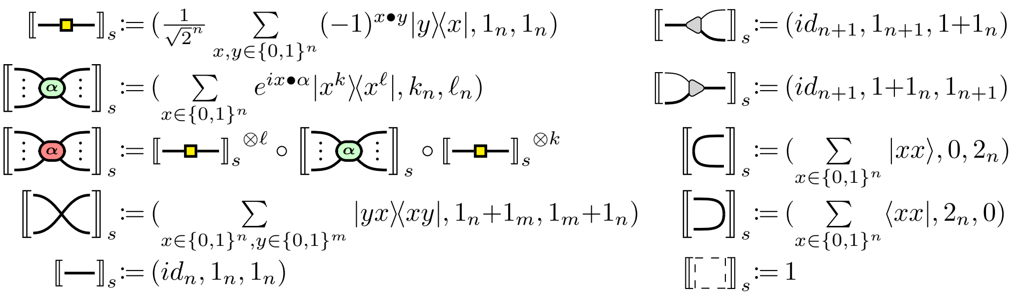
We now fuse the $\mathbb{W}$-calculus and the ZX-calculus into one language: the full SZX-calculus.

The generators of SZX-diagrams are: $\forall n, m \in \mathbb{N}^*, \forall k, \ell \in \mathbb{N}$, $\forall \alpha \in \mathbb{R}^n$,



SZX-generators can be combined using the usual sequential and spacial compositions to form SZX-diagrams. Note that for $n = m = 1$ we recover all the generators of the ZX-calculus. We denote them, as in the ZX-calculus, using thin wires e.g.  for : $1_1 \to 1_1$. Any big wire can be labelled by its size  : $1_n \to 1_n$ to avoid ambiguity. Such labels will be used mainly for scalars i.e. diagrams with no input/output. Each green or red spider is parametrised by a vector $\alpha \in \mathbb{R}^n$ of angles. With slight abuse of notation we use a single angle $\alpha_0 \in \mathbb{R}$ to denote the vector $(\alpha_0, \ldots, \alpha_0) \in \mathbb{R}^n$ when the spider has at least one leg $(k + \ell > 0)$ so that this leg can be labelled by n to avoid a potential ambiguity. Like in the ZX-calculus, the angle α_0 is omitted when $\alpha_0 = 0$.

The interpretation of ZX-diagrams is extended to SZX-diagrams as follows: for any SZX-diagram $D : a \to b$, its interpretation $\llbracket D \rrbracket_s$ is a triplet (M, a, b) where $M \in \mathcal{M}_{2^{S(b)} \times 2^{S(a)}}(\mathbb{C})$. $\llbracket D \rrbracket_s$ is inductively defined as: $\llbracket D_1 \circ D_0 \rrbracket_s = (M_1 \circ M_0, a, c)$, $\llbracket D_0 \otimes D_2 \rrbracket_s = (M_0 \otimes M_2, a + c, b + d)$ where $\llbracket D_0 \rrbracket_s = (M_0, a, b)$, $\llbracket D_1 \rrbracket_s = (M_1, b, c)$, and $\llbracket D_2 \rrbracket_s = (M_2, c, d)$. Moreover:

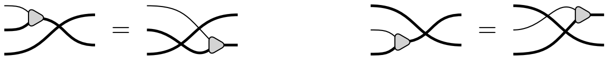

$:= (\frac{1}{\sqrt{2^n}} \sum_{x,y \in \{0,1\}^n} (-1)^{x \bullet y} |y\rangle\langle x|, 1_n, 1_n)$
$:= (id_{n+1}, 1_{n+1}, 1{+}1_n)$
$:= (\sum_{x \in \{0,1\}^n} e^{ix \bullet \alpha} |x^k\rangle\langle x^\ell|, k_n, \ell_n)$
$:= (id_{n+1}, 1{+}1_n, 1_{n+1})$
$:= \llbracket\ \rrbracket_s^{\otimes \ell} \circ \llbracket\ \rrbracket_s \circ \llbracket\ \rrbracket_s^{\otimes k}$
$:= (\sum_{x \in \{0,1\}^n} |xx\rangle, 0, 2_n)$
$:= (\sum_{x \in \{0,1\}^n, y \in \{0,1\}^m} |yx\rangle\langle xy|, 1_n{+}1_m, 1_m{+}1_n)$
$:= (\sum_{x \in \{0,1\}^n} \langle xx|, 2_n, 0)$
$:= (id_n, 1_n, 1_n)$
$:= 1$

Where $\forall u, v \in \mathbb{R}^m$, $u \bullet v = \sum_{i=1}^m u_i v_i$, $M^{\otimes 0} = 1$, and $M^{\otimes k+1} = M \otimes M^{\otimes k}$.

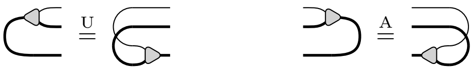
▶ **Theorem 3** (Universality). *SZX-diagrams are universal for pure qubit quantum mechanics:* $\forall a, b \in \langle \mathbb{N}^* \rangle, \forall M \in \mathcal{M}_{2^{S(b)} \times 2^{S(a)}}(\mathbb{C})$, $\exists D : a \to b$ *such that* $\llbracket D \rrbracket_s = (M, a, b)$.

3.3 The calculus

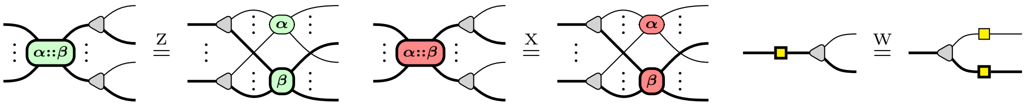
The SZX-calculus is based on distribution rules that allow dividers and gatherers to go through the big generators. For this to work we need first to ensure that the swap behaves naturally with respect to dividers and gatherers. This is given by the following two rules:



Then the rules governing the interaction between dividers, gatherers and the so-called *cups* and *caps* are:



We put labels over the equals signs to allow subsequent reference to the rules. These rules are sufficient to fully describe possible interactions between wires of any size, gatherers and dividers. It remains to specify how dividers and gatherers interact with big generators:



Where α::β means that we append the phase $\alpha \in \mathbb{R}$ to the (generalized) phase $\beta \in \mathbb{R}^n$.

This completes the set of rules of the SZX-calculus. Note that all rules agree with the interpretation, ensuring soundness of the SZX-calculus.

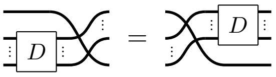
We see that any big generator s_n is in fact just n copies of the corresponding size one generator s acting in parallel. That is, a parallel composition but with a particular permutation of the inputs and outputs. Such constructions are called multiplexed diagrams in [10]. Multiplexed diagrams are shown to satisfy the same equations as size 1 diagrams. The following lemma states the same results for big generators:

▶ **Lemma 4.** *For any rule of the ZX-calculus, and any $n \in \mathbb{N}^*$, the equation obtained by replacing each generator by its big version of size n is provable in the SZX-calculus.*

We can go even further than Lemma 4. In fact, the SZX-calculus is complete:

▶ **Theorem 5.** $\forall a, b \in \langle\mathbb{N}^*\rangle, \forall D, D' \in \mathcal{SZX}[a,b],\ \llbracket D \rrbracket_s = \llbracket D' \rrbracket_s \Rightarrow SZX \vdash D = D'$.

Theorem 5 has interesting graphical consequences, ensuring that the *Only Topology Matters* paradigm applies to the SZX-calculus. In particular, swaps of any size behave naturally with respect to any diagram:



This suggests a more compact presentation close to the one of the ZX-calculus, given in the next subsection.

3.4 Compact axiomatisation

Assuming that *Only Topology Matters*, the SZX-calculus enjoys a more compact axiomatisation:

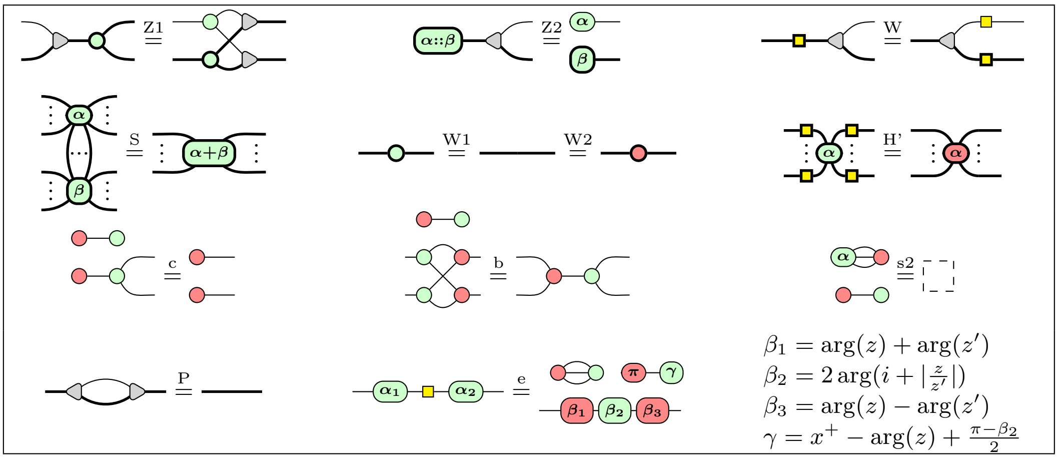
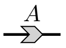
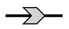
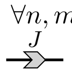

Z1
α::β
Z2
α
β
W
α
S
α+β
β
W1
W2
H'
α
α
c
b
α
s2
P
α1
α2
e
π
γ
β1
β2
β3
$\beta_1 = \arg(z) + \arg(z')$
$\beta_2 = 2\arg(i + |\frac{z}{z'}|)$
$\beta_3 = \arg(z) - \arg(z')$
$\gamma = x^+ - \arg(z) + \frac{\pi - \beta_2}{2}$

Figure 2 Axioms of the SZX-calculus, where $x^+ := \frac{\alpha_1+\alpha_2}{2}$, $x^- := x^+ - \alpha_2$, $z := -\sin(x^+) + i\cos(x^-)$, $z' := \cos(x^+) - i\sin(x^-)$ and $z' = 0 \Rightarrow \beta_2 = 0$. In the spider fusion rule, there must be at least one wire between the spiders annotated by α and β. The colour-swapped versions of those rules also hold. The bold font wires stand for wires of any size $n \geq 1$.

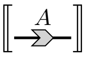
▶ **Lemma 6.** *All the rules of the SZX-calculus can be derived from the compact axioms of Figure 2 together with the Only Topology Matters paradigm.*

4 Axiomatising binary matrices for compressing diagrams

In this section, we introduce a new generator for the SZX-calculus, parametrized by a binary matrix, allowing us to represent large graphical structures in a compact way: $\forall n, m \in \mathbb{N}^*$, $\forall A \in \mathbb{F}_2^{m\times n}$, 

$: 1_n \to 1_m$. All-ones matrices will be omitted:  := 

where $\forall i,j, J_{i,j} = 1$. The new generator is interpreted as follows:

$\forall A \in \mathbb{F}_2^{m\times n}$, ⟦ 

⟧$_s$ $= (|x\rangle \mapsto |Ax\rangle, 1_n, 1_m)$

where the matrix product Ax is in $\mathbb{F}_2$ and x is seen as a column vector i.e. $(Ax)_i = \sum_{k=1}^{n} A_{i,k}x_k \bmod 2$.

▶ Remark 7. Note that, compared to [9], the matrix is not necessarily connected to green and red spiders. It is therefore a more elementary generator.

Those matrices are required to satisfy the four axioms given in Figure 3, which are sound.

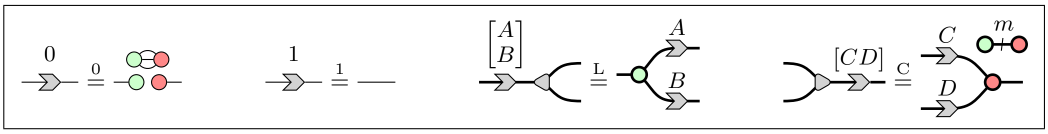
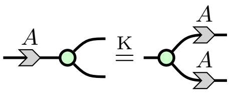


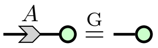
Figure 3 Axioms for matrices, where $A \in \mathbb{F}_2^{a\times n}$, $B \in \mathbb{F}_2^{b\times n}$, $C \in \mathbb{F}_2^{m\times c}$ and $D \in \mathbb{F}_2^{m\times d}$. $\begin{bmatrix}A\\B\end{bmatrix}$ and $[CD]$ are block matrices.

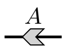
▶ Remark 8. The rules of the ZX-calculus define a scaled Hopf algebra between the green and red structure. This algebra is commutative and cocommutative with a trivial antipode. Thus, following the work of [44], the notion of $\{0,1\}$-matrices naturally emerges. It is worth noticing that it coincides with the matrices we are introducing in this section. Notice however that our axiomatisation of the matrices strongly relies on their interaction with the divider and the gatherer, which are not present in [44].

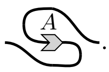
In the following, the SZX-calculus refers to the SZX-calculus augmented with the matrix generators and the axioms of Figure 3.

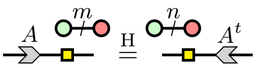
Useful equations can be derived. First, matrices are copied and erased by green nodes.

▶ **Lemma 9.** *For any $A \in \mathbb{F}_2^{m\times n}$, SZX ⊢* 

and SZX ⊢ 

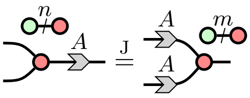
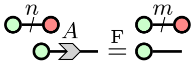
We define backward matrices as follows: 

:= 

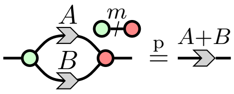
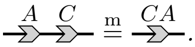
▶ **Lemma 10.** $\forall A \in \mathbb{F}_2^{m\times n}$, *SZX* ⊢ 

where A^t is the transpose of A.

As a consequence, conjugating by Hadamard ( ) reverses the orientation and transposes the matrix (up to scalars). Since conjugating by Hadamard colour-swaps the spiders and preserves the other generators of the language, one can derive from any equation a new one (up to scalars) which consists in colour-swapping the spiders, transposing the matrices and then changing their orientation. For instance Lemma 9 gives that matrices are cocopied and coerased by red nodes:

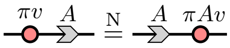
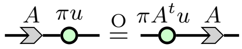
▶ **Lemma 11.** *For any $A \in \mathbb{F}_2^{m\times n}$, SZX ⊢*  *and SZX ⊢* 

Basic matrix operations like addition and multiplication (in $\mathbb{F}_2$) can be implemented graphically:

▶ **Lemma 12.** *For any $A, B \in \mathbb{F}_2^{m\times n}$, and any $C \in \mathbb{F}_2^{k\times m}$, SZX ⊢*  *and SZX ⊢*  *.*

Whereas all the previous properties about matrices are angle-free, some spiders whose angles are multiple of π can be pushed through matrices as follows:

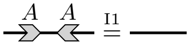
▶ **Lemma 13.** *For any $A \in \mathbb{F}_2^{m\times n}$, any $v \in \mathbb{F}_2^n$ and any $u \in \mathbb{F}_2^m$,*

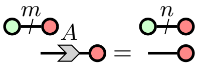
SZX ⊢  *and SZX ⊢* 

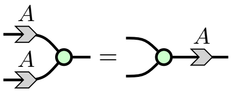
Injective matrices enjoy some specific properties:

▶ **Lemma 14.** *For any $A \in \mathbb{F}_2^{m\times n}$, the following properties are equivalent:*

(1) *A is injective.*

(3) *SZX ⊢* 

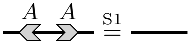
(2) *SZX ⊢* 

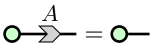
(4) *SZX ⊢* 

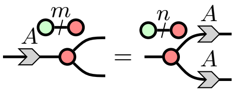
By Hadamard conjugation, we obtain some dual properties for surjective matrices:

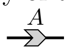
▶ **Lemma 15.** *For any $A \in \mathbb{F}_2^{m\times n}$, the following properties are equivalent:*

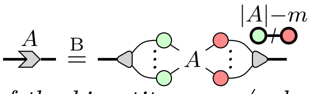
(1) *A is surjective.*

(3) *SZX ⊢* 

(2) *SZX ⊢* 

(4) *SZX ⊢* 

Due to the universality of the SZX-calculus, matrices are expressible as SZX-diagrams, and the matrix generator  is actually a compact representation of a green/red bipartite graphs whose biadjacency matrix is A:

▶ **Lemma 16.** *For any $A \in \mathbb{F}_2^{m\times n}$, SZX ⊢*  *where A represents in the RHS diagram the adjacency matrix of the bipartite green/red graph, and |A| is the number of 1 in A.*

Lemma 16 and 5 imply the completeness of the SZX-calculus with matrices:

▶ **Theorem 17.** *SZX-calculus with matrices is complete.*

5 Applications

This section provides two examples of the SZX-calculus in action.

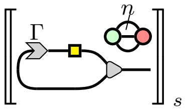
5.1 Application to graph states

Graph states [24] form a subclass of quantum states that can be represented by simple undirected graphs where each vertex represents a qubit and the edges represent intuitively the entanglement between qubits. The graph state formalism is widely used in quantum information processing, providing combinatorial characterisations of quantum properties in measurement-based quantum computing [40, 13, 6], secret sharing [37, 29, 25], error correcting codes [41, 4] etc. Graph states are also strongly related to the ZX-calculus [18] where they have been used for instance in proving the completeness of some fragments [1, 20]. A graph state is a particular kind of stabilizer state and thus can be defined as a fixpoint: given a graph G of order n, the corresponding graph state $|G\rangle$ is a the unique state (up to a global phase) such that for any vertex u, applying $X = [\![-\pi-]\!]$ on u and $Z = [\![-\pi-]\!]$ on its neighbours leaves the state unchanged. The global phase is fixed by the extra condition $\langle 0^n|G\rangle = \frac{1}{\sqrt{n}}$.

A graph state admits a simple representation as a ZX-diagram: each vertex is represented by a green spider connected to an output, and each edge is represented by a Hadamard (─□─) connecting the corresponding green dots. In the following, we provide two alternative, scalable, representations of graph states: the first is a compact matrix-based representation of bipartite graph states, the second is an inductive definition of arbitrary graph state, allowing inductive proofs. In both representations, we provide diagrammatic proofs of some key properties of the graph states.

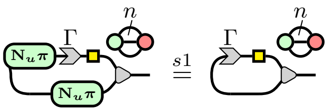
First, any bipartite graph state can be depicted with a SZX-diagram via its biadjacency matrix:

▶ **Lemma 18.** *For any bipartite graph G with biadjacency matrix $\Gamma \in \mathbb{F}_2^{m\times n}$,*

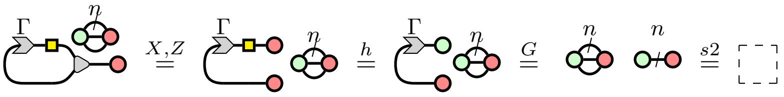


$$= (|G\rangle, 0, 1_{n+m})$$

Proof of Lemma 18. The last two components are straightforward typing, for the first one we use the characterization of $|G\rangle$ by its stabilizer [24]. $|G\rangle$ is the unique (up to a scalar) common fix point of $X_u Z_{N_u}$ for all vertices u of G. Each subset of vertices is identified with its characteristic vector, e.g. $\mathbf{N}_u\pi$ is a vector with a π at the position i if the i-th vertex is a neighbour of u, and 0 otherwise. The following proof uses the fact that $\Gamma^t u = \mathbf{N}_u$. We assume u is in the first part of the bipartite graph (the other case is similar):



It remains to take care of the scalar. We see that $\sqrt{2^{n+m}}\langle 0^{n+m}|G\rangle = 1$:

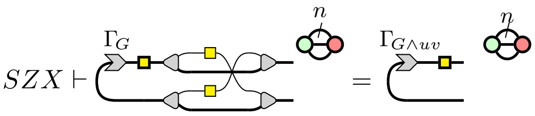


◀

A fundamental property of graph states is that graph transformations (like pivoting and local complementation) can be performed on graph states using local operations. Given a bipartite graph G, pivoting according to an edge (u, v) produces a graph denoted $G \wedge uv$ where the labels u and v are exchanged and their neighbourhood is complemented: for any $w \in \mathbf{N}_u \setminus v$ and $t \in \mathbf{N}_v \setminus u$, w and t are connected in $G \wedge uv$ iff they are not connected in G. Pivoting can be implemented on bipartite graph states by simply applying Hadamard on u

and v: $H_{u,v}|G\rangle = |G \wedge uv\rangle$ [42, 38]. This property can be derived in the SZX-calculus, and its proof, given in arXiv:1905.00041 due to space limits, provides an interesting example of the SZX-calculus in action:

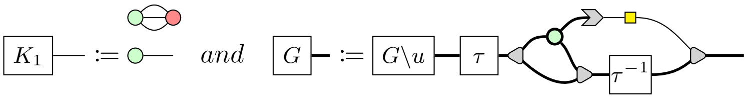
▶ **Lemma 19.** *Given a bipartite graph G and an edge (u, v),*



where Γ_G (resp. $\Gamma_{G\wedge uv}$) is the biadjacency matrix of G (resp. $G\wedge uv$) such that u corresponds to the first row (resp. column) and v to the first column (resp. row).

Now we introduce a general inductive definition of graph state boxes, which associates a SZX-diagram with any (not necessarily bipartite) graph.

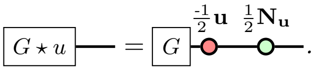
▶ **Definition 20.** *Given a graph G with ordered vertices, the corresponding graph state box is defined by:*



where K_1 is the graph of order 1, u is the first vertex of G, τ is a permutation on the list of vertices of $G\backslash u$ which puts the neighbourhood of u first and then the other vertices.

▶ **Lemma 21.** $\left[\!\left[\,\boxed{G}\!-\,\right]\!\right]_s = (|G\rangle, 0, 1_n).$

We will now use the SZX-calculus to show the property known as local complementation. Given a vertex u of a graph G the local complementation of G according to u is the graph $G \star u$ which is G where all edges between neighbours of u have been complemented, that is, edges became non-edges and non-edges became edges.

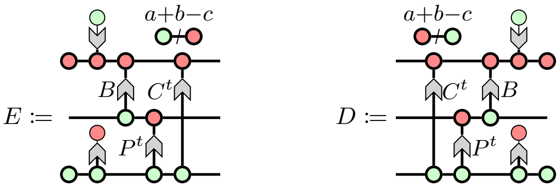
▶ **Theorem 22.** *For any graph G and vertex u, $SZX \vdash$* .

5.2 Application to error correcting codes

The original motivation for the development of a scalable ZX-calculus was the design of tripartite Coherent Parity Checking (CPC) error correcting codes [9]. We reformulate here in the SZX-calculus the definition of those codes and the proof of some elementary properties.

The idea is to spread the information of some logical qubits over a bigger number of physical qubits. In our example the code is parametrized by three matrices $B \in \mathbb{F}_2^{a\times b}$, $P \in \mathbb{F}_2^{c\times b}$ and $C \in \mathbb{F}_2^{c\times a}$. The aim is to encode b logical qubits into $a+b+c$ physical ones.

▶ **Definition 23.** *The tripartite CPC encoder $E : 1_b \to 1_a+1_b+1_c$ and decoder $D : 1_a+1_b+1_c \to 1_b$ defined by the matrices $B \in \mathbb{F}_2^{a\times b}$, $P \in \mathbb{F}_2^{c\times b}$ and $C \in \mathbb{F}_2^{c\times a}$ are:*

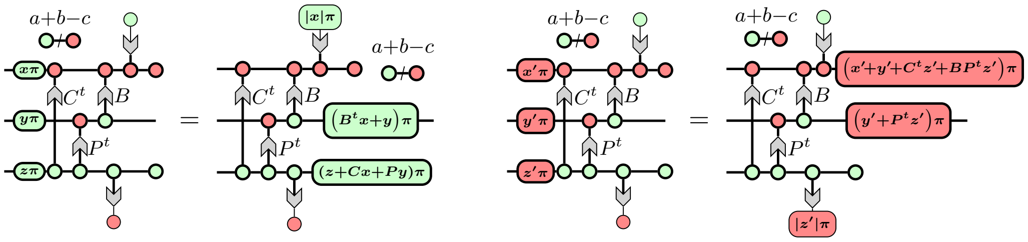


We can prove that the code is correct when there are no errors, in other words:

▶ **Lemma 24.** *The encoder is an isometry that is $SZX \vdash D \circ E = $ —.*

We now end by showing what happens when errors go through the decoder: x, y and z (resp. x', y', z') are vectors of phase flip errors (resp. bit flip errors). The implementation of the decoder involves some measurements, which according to Lemma 25, produce some syndromes ($|x| = \sum_i x_i \bmod 2$, $z{+}Cx{+}Py$, $x'{+}y'{+}C^tz'{+}BP^tz'$, and $|z'|$) which guide us to correct the middle wire. Of course the exact protocol and its efficiency depend on clever choices of B, P and C, see [9] for details.

▶ **Lemma 25.** *The following equalities hold in the SZX-calculus:*



6 Conclusion and further work

We have introduced the SZX-calculus, a formal and compact graphical language for quantum reasoning, and proved its universality, soundness, and completeness. This work is addressing two main objectives. First, to demonstrate that some of the ingredients for scalability which were sketched out in [9] – like the thick wires and the use of matrices – together with some new ingredients – like the divider and the gatherer – can be axiomatised to provide a complete scalable graphical language. Our second objective was to provide a sufficiently precise definition of the language to consider an implementation in a graphical proof assistant like Quantomatic. This last point would pave the way towards the formal verification of large scale quantum protocols and algorithms.

We aim to provide a language ready for applications and available to most of the quantum computing community. For this reason, we have deliberately avoided a categorical presentation. A fully categorical description of the scalable construction will be the subject of further work. We nevertheless provide here a sketch of how our construction can be generalised in a categorical setting. Graphical languages can be defined as props, see [3] and [44], that is symmetric strict monoidal categories whose set of objects is freely generated by one object we denote 1. In fact it is possible to define a scalable construction for any coloured prop. Given a set C of colours we can define two $\langle\langle C\rangle - \varepsilon\rangle$-coloured props $\mathbb{D}_C$ and $\mathbb{G}_C$ whose objects are formal sums of 1_{n_c} and morphisms are respectively generated by dividers and gatherers for each pair (n, c). The elimination rule is a distribution rule as in [36], which allows us to define the composed prop $\mathbb{D}_C ; \mathbb{G}_C$. The prop of wires $\mathbb{W}_C$ is then defined as this composition quotiented by the expansion rule. This last pro satisfies a rewire theorem similar to Theorem 2. Then given a C-coloured prop $\mathbf{P}$, we define the $\langle\langle C\rangle - \varepsilon\rangle$-coloured prop $\overline{\mathbf{P}}$ which has the same generators and equations as those of $\mathbf{P}$ on wires of size 1. Finally the scalable prop $\mathcal{S}\mathbf{P}$ is defined as the composition of prop $\mathbb{D}_C ; \overline{\mathbf{P}} ; \mathbb{G}_C$ quotiented by the expansion rule. The corresponding distribution rules follows the same pattern as in 3.3. Such a generalization gives scalable versions of any graphical language based on props such as the ZW–calculus [22], the ZH-calculus [2] or $\mathbb{IH}$ [5].

References

1 Miriam Backens. The ZX-Calculus is Complete for Stabilizer Quantum Mechanics. *New Journal of Physics*, 16(9):093021, September 2014. `doi:10.1088/1367-2630/16/9/093021`.

2 Miriam Backens and Aleks Kissinger. ZH: A complete graphical calculus for quantum computations involving classical non-linearity. In Peter Selinger and Giulio Chiribella, editors, Proceedings of the 15th International Conference on *Quantum Physics and Logic,* Halifax, Canada, 3-7th June 2018, volume 287 of *Electronic Proceedings in Theoretical Computer Science*, pages 23–42. Open Publishing Association, 2019. `doi:10.4204/EPTCS.287.2`.

3 John C Baez, Brandon Coya, and Franciscus Rebro. Props in network theory. *arXiv preprint arXiv:1707.08321*, 2017. `arXiv:1707.08321`.

4 BA Bell, DA Herrera-Martí, MS Tame, D Markham, WJ Wadsworth, and JG Rarity. Experimental demonstration of a graph state quantum error-correction code. *Nature communications*, 5:3658, 2014.

5 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. Interacting Hopf algebras. *Journal of Pure and Applied Algebra*, 221(1):144–184, 2017.

6 Daniel E Browne, Elham Kashefi, Mehdi Mhalla, and Simon Perdrix. Generalized flow and determinism in measurement-based quantum computation. *New Journal of Physics*, 9(8):250, 2007. URL: `http://stacks.iop.org/1367-2630/9/i=8/a=250`.

7 Titouan Carette, Dominic Horsman, and Simon Perdrix. SZX-calculus: Scalable graphical quantum reasoning, 2019. `arXiv:1905.00041`.

8 Titouan Carette, Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Completeness of Graphical Languages for Mixed States Quantum Mechanics. In *International Colloquium on Automata, Languages, and Programming (ICALP'19)*, 2019.

9 Nicholas Chancellor, Aleks Kissinger, Joschka Roffe, Stefan Zohren, and Dominic Horsman. Graphical structures for design and verification of quantum error correction. *arXiv preprint arXiv:1611.08012*, 2016. `arXiv:1611.08012`.

10 Apiwat Chantawibul and Paweł Sobociński. Monoidal Multiplexing. In *International Colloquium on Theoretical Aspects of Computing*, pages 116–131. Springer, 2018.

11 Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011.

12 Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the qubit routing problem. *arXiv preprint arXiv:1902.08091*, 2019. `arXiv:1902.08091`.

13 Vincent Danos, Elham Kashefi, Prakash Panangaden, and Simon Perdrix. Extended Measurement Calculus. *Semantic Techniques in Quantum Computation*, 2010.

14 Niel de Beaudrap and Dominic Horsman. The ZX calculus is a language for surface code lattice surgery, 2017. `arXiv:1704.08670`.

15 Ross Duncan. A graphical approach to measurement-based quantum computing. *arXiv preprint arXiv:1203.6242*, 2012. `arXiv:1203.6242`.

16 Ross Duncan, Aleks Kissinger, Simon Pedrix, and John van de Wetering. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *arXiv preprint arXiv:1902.03178*, 2019. `arXiv:1902.03178`.

17 Ross Duncan and Maxime Lucas. Verifying the Steane code with Quantomatic. *Electronic Proceedings in Theoretical Computer Science*, 171:33–49, 2014. arXiv preprint arXiv:1902.03178. `arXiv:1902.03178`.

18 Ross Duncan and Simon Perdrix. Graph states and the necessity of Euler decomposition. In *Conference on Computability in Europe (CiE)*, pages 167–177. Springer, 2009.

19 Ross Duncan and Simon Perdrix. Rewriting Measurement-Based Quantum Computations with Generalised Flow. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming*, pages 285–296, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

20 Ross Duncan and Simon Perdrix. Pivoting Makes the ZX-Calculus Complete for Real Stabilizers. In *QPL 2013*, Electronic Proceedings in Theoretical Computer Science, pages 50–62, 2013. doi:10.4204/EPTCS.171.5.

21 Craig Gidney and Austin G Fowler. Efficient magic state factories with a catalyzed |CCZ> to 2|T> transformation. *arXiv preprint arXiv:1812.01238*, 2018. arXiv:1812.01238.

22 Amar Hadzihasanovic. The algebra of entanglement and the geometry of composition. *arXiv preprint arXiv:1709.08086*, 2017. arXiv:1709.08086.

23 Amar Hadzihasanovic, Kang Feng Ng, and Quanlong Wang. Two Complete Axiomatisations of Pure-state Qubit Quantum Computing. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 502–511, New York, NY, USA, 2018. ACM. doi:10.1145/3209108.3209128.

24 Marc Hein, Wolfgang Dür, Jens Eisert, Robert Raussendorf, M Nest, and H-J Briegel. Entanglement in graph states and its applications. *Proceedings of the International School of Physics "Enrico Fermi" on "Quantum Computers, Algorithms and Chaos"*, 2006. arXiv:quantph/0602096.

25 Jérôme Javelle, Mehdi Mhalla, and Simon Perdrix. New Protocols and Lower Bounds for Quantum Secret Sharing with Graph States. In Kazuo Iwama, Yasuhito Kawano, and Mio Murao, editors, *Theory of Quantum Computation, Communication, and Cryptography*, volume 7582 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-35656-8_1.

26 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A complete axiomatisation of the ZX-calculus for Clifford+ T quantum mechanics. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 559–568. ACM, 2018.

27 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A Generic Normal Form for ZX-Diagrams and Application to the Rational Angle Completeness. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2019. arXiv:1805.05296.

28 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Completeness of the ZX-Calculus. *arXiv preprint arXiv:1903.06035*, 2019. arXiv:1903.06035.

29 Elham Kashefi, Damian Markham, Mehdi Mhalla, and Simon Perdrix. Information flow in secret sharing protocols. In *Proc. 5th Workshop on Developments in Computational Models (DCM).*, volume 9, pages 87–97. EPTCS, 2009.

30 Aleks Kissinger and Arianne Meijer-van de Griend. CNOT circuit extraction for topologically-constrained quantum memories. *arXiv preprint arXiv:1904.00633*, 2019. arXiv:1904.00633.

31 Aleks Kissinger and David Quick. A first-order logic for string diagrams. *arXiv preprint arXiv:1505.00343*, 2015. arXiv:1505.00343.

32 Aleks Kissinger and John van de Wetering. Universal MBQC with generalised parity-phase interactions and Pauli measurements. *arXiv:1704.06504*, 2017. arXiv:1704.06504.

33 Aleks Kissinger and John van de Wetering. PyZX, 2018. URL: https://github.com/Quantomatic/pyzx.

34 Aleks Kissinger and John van de Wetering. Reducing T-count with the ZX-calculus. *arXiv preprint arXiv:1903.10477*, 2019. arXiv:1903.10477.

35 Aleks Kissinger and Vladimir Zamdzhiev. Quantomatic: A proof assistant for diagrammatic reasoning. In *International Conference on Automated Deduction*, pages 326–336. Springer, 2015.

36 Stephen Lack. Composing props. *Theory and Applications of Categories*, 13(9):147–163, 2004.

37 Damian Markham and Barry C. Sanders. Graph States for Quantum Secret Sharing. *Physical Review A*, 78:042309, 2008. URL: doi:10.1103/PhysRevA.78.042309.

38 Mehdi Mhalla and Simon Perdrix. Graph states, pivot minor, and universality of (X, Z)-measurements. *arXiv preprint arXiv:1202.6551*, 2012. arXiv:1202.6551.

39 Filippo M Miatto. Graphical Calculus for products and convolutions. *arXiv preprint arXiv:1903.01366*, 2019. arXiv:1903.01366.

40 Robert Raussendorf, Daniel E. Browne, and Hans J. Briegel. Measurement-based quantum computation on cluster states. *Physical Review A*, 68(2), August 2003. doi:10.1103/physreva.68.022312.

41 D. Schlingemann and R. F. Werner. Quantum error-correcting codes associated with graphs. *Physical Review A*, 65, 2001. doi:10.1103/PhysRevA.65.012308.

42 Maarten Van den Nest. *Local equivalence of stabilizer states and codes*. PhD thesis, Faculty of Engineering, K. U. Leuven, Belgium, May 2005.

43 Renaud Vilmart. A Near-Optimal Axiomatisation of ZX-Calculus for Pure Qubit Quantum Mechanics. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2019. arXiv:1812.09114.

44 Fabio Zanasi. Interacting Hopf Algebras: the theory of linear systems. *arXiv preprint arXiv:1805.03032*, 2018. arXiv:1805.03032.

On the Stretch Factor of Polygonal Chains

Ke Chen
Department of Computer Science, University of Wisconsin–Milwaukee, USA
kechen@uwm.edu

Adrian Dumitrescu
Department of Computer Science, University of Wisconsin–Milwaukee, USA
dumitres@uwm.edu

Wolfgang Mulzer
Institut für Informatik, Freie Universität Berlin, Germany
mulzer@inf.fu-berlin.de

Csaba D. Tóth
Department of Mathematics, California State University Northridge, Los Angeles, CA
Department of Computer Science, Tufts University, Medford, MA, USA
csaba.toth@csun.edu

Abstract

Let $P = (p_1, p_2, \ldots, p_n)$ be a polygonal chain. The *stretch factor* of P is the ratio between the total length of P and the distance of its endpoints, $\sum_{i=1}^{n-1} |p_i p_{i+1}|/|p_1 p_n|$. For a parameter $c \geq 1$, we call P a *c-chain* if $|p_i p_j| + |p_j p_k| \leq c|p_i p_k|$, for every triple (i, j, k), $1 \leq i < j < k \leq n$. The stretch factor is a global property: it measures how close P is to a straight line, and it involves all the vertices of P; being a c-chain, on the other hand, is a *fingerprint*-property: it only depends on subsets of $O(1)$ vertices of the chain.

We investigate how the c-chain property influences the stretch factor in the plane: (i) we show that for every $\varepsilon > 0$, there is a noncrossing c-chain that has stretch factor $\Omega(n^{1/2-\varepsilon})$, for sufficiently large constant $c = c(\varepsilon)$; (ii) on the other hand, the stretch factor of a c-chain P is $O\left(n^{1/2}\right)$, for every constant $c \geq 1$, regardless of whether P is crossing or noncrossing; and (iii) we give a randomized algorithm that can determine, for a polygonal chain P in $\mathbb{R}^2$ with n vertices, the minimum $c \geq 1$ for which P is a c-chain in $O\left(n^{2.5} \text{ polylog } n\right)$ expected time and $O(n \log n)$ space.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Theory of computation → Computational geometry

Keywords and phrases polygonal chain, vertex dilation, Koch curve, recursive construction

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.56

Related Version A full version is available on the arXiv under `http://arxiv.org/abs/1906.10217`.

Funding *Wolfgang Mulzer*: Partially supported by ERC STG 757609.
Csaba D. Tóth: Supported in part by NSF CCF-1422311 and CCF-1423615.

Acknowledgements This work was initiated at the Fields Workshop on Discrete and Computational Geometry, held July 31–August 4, 2017, at Carleton University. The authors thank the organizers and all participants of the workshop for inspiring discussions and for providing a great research atmosphere. This problem was initially posed by Rolf Klein in 2005. We would like to thank Rolf Klein and Christian Knauer for interesting discussions on the stretch factor and related topics.

1 Introduction

Given a set S of n point sites in the plane, what is the best way to connect S into a *geometric network (graph)*? This question has motivated researchers for a long time, going back as far as the 1940s, and beyond [19,35]. Numerous possible criteria for a good geometric network

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 56; pp. 56:1–56:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

have been proposed, perhaps the most basic being the *length*. In 1955, Few [20] showed that for any set of n points in a unit square, there is a traveling salesman tour of length at most $\sqrt{2n} + 7/4$. This was improved to at most $0.984\sqrt{2n} + 11$ by Karloff [23]. Similar bounds also hold for the shortest spanning tree and the shortest rectilinear spanning tree [13, 16, 21]. Besides length, two further key factors in the quality of a geometric network are the *vertex dilation* and the *geometric dilation* [31], both of which measure how closely shortest paths in a network approximate the Euclidean distances between their endpoints.

The *dilation* (also called *stretch factor* [29] or *detour* [1]) between two points p and q in a geometric graph G is defined as the ratio between the length of a shortest path from p to q and the Euclidean distance $|pq|$. The *dilation* of the graph G is the maximum dilation over all pairs of vertices in G. A graph in which the dilation is bounded above by $t \geq 1$ is also called a *t-spanner* (or simply a *spanner* if t is a constant). A complete graph in Euclidean space is clearly a 1-spanner. Therefore, researchers focused on the dilation of graphs with certain additional constraints, for example, noncrossing (i.e., plane) graphs. In 1989, Das and Joseph [15] identified a large class of plane spanners (characterized by two simple local properties). Bose et al. [6] gave an algorithm that constructs for any set of planar sites a plane 11-spanner with bounded degree. On the other hand, Eppstein [18] analyzed a fractal construction showing that *β-skeletons*, a natural class of geometric networks, can have arbitrarily large dilation.

The study of dilation also raises algorithmic questions. Agarwal et al. [1] described randomized algorithms for computing the dilation of a given path (on n vertices) in $\mathbb{R}^2$ in $O(n \log n)$ expected time. They also presented randomized algorithms for computing the dilation of a given tree, or cycle, in $\mathbb{R}^2$ in $O(n \log^2 n)$ expected time. Previously, Narasimhan and Smid [30] showed that an $(1 + \varepsilon)$-approximation of the stretch factor of any path, cycle, or tree can be computed in $O(n \log n)$ time. Klein et al. [24] gave randomized algorithms for a path, tree, or cycle in $\mathbb{R}^2$ to count the number of vertex pairs whose dilation is below a given threshold in $O(n^{3/2+\varepsilon})$ expected time. Cheong et al. [12] showed that it is NP-hard to determine the existence of a spanning tree on a planar point set whose dilation is at most a given value. More results on plane spanners can be found in the monograph dedicated to this subject [31] or in several surveys [8, 17, 29].

We investigate a basic question about the dilation of polygonal chains. More precisely, we ask how the dilation between the endpoints of a polygonal chain (which we will call the *stretch factor*, to distinguish it from the more general notion of dilation) is influenced by *fingerprint* properties of the chain, i.e., by properties that are defined on $O(1)$-size subsets of the vertex set. Such fingerprint properties play an important role in geometry, where classic examples include the *Carathéodory property*[1] [26, Theorem 1.2.3] or the *Helly property*[2] [26, Theorem 1.3.2]. In general, determining the effect of a fingerprint property may prove elusive: given n points in the plane, consider the simple property that every 3 points determine 3 distinct distances. It is unknown [9, p. 203] whether this property implies that the total number of distinct distances grows superlinearly in n.

Furthermore, fingerprint properties appear in the general study of *local versus global properties of metric spaces* that is highly relevant to combinatorial approximation algorithms that are based on mathematical programming relaxations [5]. In the study of dilation,

[1] Given a finite set S of points in d dimensions, if every $d + 2$ points in S are in convex position, then S is in convex position.

[2] Given a finite collection of convex sets in d dimensions, if every $d + 1$ sets have nonempty intersection, then all sets have nonempty intersection.

interesting fingerprint properties have also been found. For example, a (continuous) curve C is said to have the *increasing chord property* [14, 25] if for any points a, b, c, d that appear on C in this order, we have $|ad| \geq |bc|$. The increasing chord property implies that C has (geometric) dilation at most $2\pi/3$ [33]. A weaker property is the *self-approaching property*: a (continuous) curve C is self-approaching if for any points a, b, c that appear on C in this order, we have $|ac| \geq |bc|$. Self-approaching curves have dilation at most 5.332 [22] (see also [3]), and they have found interesting applications in the field of graph drawing [4, 7, 32].

We introduce a new natural fingerprint property and see that it can constrain the stretch factor of a polygonal chain, but only in a weaker sense than one may expect; we also provide algorithmic results on this property. Before providing details, we give a few basic definitions.

Definitions. A *polygonal chain* P in the Euclidean plane is specified by a sequence of n points $(p_1, p_2, \ldots, p_n)$, called its *vertices*. The chain P consists of $n-1$ line segments between consecutive vertices. We say P is *simple* if only consecutive line segments intersect and they only intersect at their endpoints. Given a polygonal chain P in the plane with n vertices and a parameter $c \geq 1$, we call P a *c-chain* if for all $1 \leq i < j < k \leq n$, we have

$$|p_i p_j| + |p_j p_k| \leq c|p_i p_k|. \tag{1}$$

Observe that the c-chain condition is a fingerprint condition that is not really a local dilation condition – it is more a combination between the local chain substructure and the distribution of the points in the subchains.

The *stretch factor* δ_P of P is defined as the dilation between the two end points p_1 and p_n of the chain:

$$\delta_P = \frac{\sum_{i=1}^{n-1} |p_i p_{i+1}|}{|p_1 p_n|}.$$

Note that this definition is different from the more general notion of dilation (also called *stretch factor* [29]) of a graph which is the maximum dilation over all pairs of vertices. Since there is no ambiguity in this paper, we will just call δ_P the stretch factor of P.

For example, the polygonal chain $P = ((0,0), (1,0), \ldots, (n,0))$ is a 1-chain with stretch factor 1; and $Q = ((0,0), (0,1), (1,1), (1,0))$ is a $(\sqrt{2}+1)$-chain with stretch factor 3.

Without affecting the results, the floor and ceiling functions are omitted in our calculations. For a positive integer t, let $[t] = \{1, 2, \ldots, t\}$. For a point set S, let $\operatorname{conv}(S)$ denote the convex hull of S. All logarithms are in base 2, unless stated otherwise.

Our results. We deduce three upper bounds on the stretch factor of a c-chain P with n vertices (Section 2). In particular, we have (i) $\delta_P \leq c(n-1)^{\log c}$, (ii) $\delta_P \leq c(n-2)+1$, and (iii) $\delta_P = O\left(c^2\sqrt{n-1}\right)$.

From the other direction, we obtain the following lower bound (Section 3): For every $c \geq 4$, there is a family $\mathcal{P}_c = \{P^k\}_{k\in\mathbb{N}}$ of simple c-chains, so that P^k has $n = 4^k + 1$ vertices and stretch factor $(n-1)^{\frac{1+\log(c-2)-\log c}{2}}$, where the exponent converges to 1/2 as c tends to infinity. The lower bound construction does not extend to the case of $1 < c < 4$, which remains open.

Finally, we present two algorithmic results (Section 4): (i) A randomized algorithm that decides, given a polygonal chain P in $\mathbb{R}^2$ with n vertices and a threshold $c > 1$, whether P is a c-chain in $O\left(n^{2.5} \text{ polylog } n\right)$ expected time and $O(n \log n)$ space. (ii) As a corollary, there is a randomized algorithm that finds, for a polygonal chain P with n vertices, the minimum $c \geq 1$ for which P is a c-chain in $O\left(n^{2.5} \text{ polylog } n\right)$ expected time and $O(n \log n)$ space.

2 Upper Bounds

At first glance, one might expect the stretch factor of a c-chain, for $c \geq 1$, to be bounded by some function of c. For example, the stretch factor of a 1-chain is necessarily 1. We derive three upper bounds on the stretch factor of a c-chain with n vertices in terms of c and n (cf. Theorems 1–3); see Fig. 1 for a visual comparison between the bounds. For large n, the bound in Theorem 1 is the best for $1 \leq c \leq 2^{1/2}$, while the bound in Theorem 3 is the best for $c > 2^{1/2}$. In particular, the bound in Theorem 1 is tight for $c = 1$. The bound in Theorem 2 is the best for $c \geq 2$ and $n \leq 111c^2$.

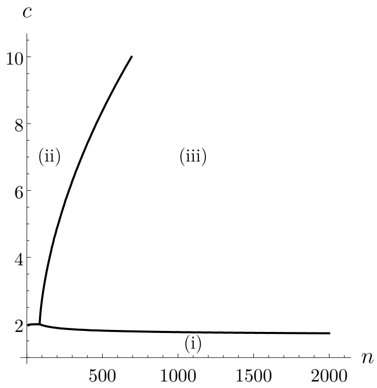


Figure 1 The values of n and c for which (i) Theorem 1, (ii) Theorem 2, and (iii) Theorem 3 give the current best upper bound.

Our first upper bound is obtained by a recursive application of the c-chain property. It holds for any positive distance function that may not even satisfy the triangle inequality.

▶ **Theorem 1.** *For a c-chain P with n vertices, we have $\delta_P \leq c(n-1)^{\log c}$.*

Proof. We prove, by induction on n, that

$$\delta_P \leq c^{\lceil \log(n-1) \rceil}, \tag{2}$$

for every c-chain P with $n \geq 2$ vertices. In the base case, $n = 2$, we have $\delta_P = 1$ and $c^{\lceil \log(2-1) \rceil} = 1$. Now let $n \geq 3$, and assume that (2) holds for every c-chain with fewer than n vertices. Let $P = (p_1, \ldots, p_n)$ be a c-chain with n vertices. Then, applying (2) to the first and second half of P, followed by the c-chain property for the first, middle, and last vertex of P, we get

$$\begin{aligned}
\sum_{i=1}^{n-1} |p_i p_{i+1}| &\leq \sum_{i=1}^{\lceil n/2 \rceil - 1} |p_i p_{i+1}| + \sum_{i=\lceil n/2 \rceil}^{n-1} |p_i p_{i+1}| \\
&\leq c^{\lceil \log(\lceil n/2 \rceil - 1) \rceil} \left(|p_1 p_{\lceil n/2 \rceil}| + |p_{\lceil n/2 \rceil} p_n| \right) \\
&\leq c^{\lceil \log(\lceil n/2 \rceil - 1) \rceil} \cdot c |p_1 p_n| \\
&\leq c^{\lceil \log(n-1) \rceil} |p_1 p_n|,
\end{aligned}$$

so (2) holds also for P. Consequently,

$$\delta_P \leq c^{\lceil \log(n-1) \rceil} \leq c^{\log(n-1)+1} = c \cdot c^{\log(n-1)} = c\,(n-1)^{\log c},$$

as required. ◀

Our second bound interprets the c-chain property geometrically and makes use of the fact that P resides in the Euclidean plane.

▶ **Theorem 2.** *For a c-chain P with n vertices, we have $\delta_P \le c(n-2)+1$.*

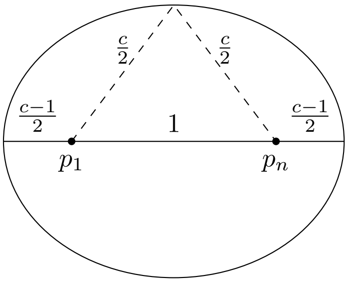


Figure 2 The entire chain P lies in an ellipse with foci p_1 and p_n.

Proof. Without loss of generality, assume that $|p_1p_n| = 1$. Since P is a c-chain, for every $1 < j < n$, we have $|p_1p_j| + |p_jp_n| \le c|p_1p_n| = c$. If we fix the points p_1 and p_n, then every p_j lies in an ellipse E with foci p_1 and p_n, for $1 < j < n$, see Figure 2. The diameter of E is its major axis, whose length is c. Since E contains all vertices of the chain P, we have $|p_1p_2|, |p_{n-1}p_n| \le \frac{c+1}{2} \le c$ and $|p_jp_{j+1}| \le c$ for all $1 < j < n-1$. Therefore the stretch factor of P is bounded above by

$$\delta_P = \frac{\sum_{j=1}^{n-1} |p_jp_{j+1}|}{|p_1p_n|} = |p_1p_2| + |p_{n-1}p_n| + \sum_{j=2}^{n-2} |p_jp_{j+1}|$$
$$\le \frac{c+1}{2} + \frac{c+1}{2} + c(n-3) = c(n-2)+1,$$

as required. ◀

Our third upper bound uses a volume argument to bound the number of long edges in P.

▶ **Theorem 3.** *Let $P = (p_1, \ldots, p_n)$ be a c-chain, for some constant $c \ge 1$, and let $L = \sum_{i=1}^{n-1} |p_ip_{i+1}|$ be its length. Then $L = O\left(c^2\sqrt{n-1}\right)|p_1p_n|$, hence $\delta_P = O\left(c^2\sqrt{n-1}\right)$.*

Proof. We may assume that p_1p_n is a horizontal segment of unit length. By the argument in the proof of Theorem 2, all points p_i ($i = 1, \ldots, n$) are contained in an ellipse E with foci p_1 and p_n, where the major axis of E has length c. Let U be the minimal axis-aligned square containing E; its side is of length c.

We set $x = 8c^2/\sqrt{n-1}$; and let L_0 and L_1 be the sum of lengths of all edges in P of length at most x and more than x, respectively. By definition, we have $L = L_0 + L_1$ and

$$L_0 \le (n-1)x = (n-1)\cdot 8c^2/\sqrt{n-1} = 8c^2\sqrt{n-1}. \tag{3}$$

We shall prove that $L_1 \le 8c^2\sqrt{n-1}$, implying $L \le 2x(n-1) = O\left(c^2\sqrt{n-1}\right)$. For this, we further classify the edges in L_1 according to their lengths: For $\ell = 0, 1, \ldots, \infty$, let

$$P_\ell = \left\{p_i : 2^\ell x < |p_ip_{i+1}| \le 2^{\ell+1}x\right\}. \tag{4}$$

Since all points lie in an ellipse of diameter c, we have $|p_ip_{i+1}| \le c$, for all $i = 0, \ldots, n-1$. Consequently, $P_\ell = \emptyset$ when $c \le 2^\ell x$, or equivalently $\log(c/x) \le \ell$.

We use a volume argument to derive an upper bound on the cardinality of P_ℓ, for $\ell = 0, 1, \dots, \lfloor \log(c/x) \rfloor$. Assume that $p_i, p_k \in P_\ell$, and w.l.o.g., $i < k$. If $k = i + 1$, then by (4), $2^\ell x < |p_i p_k|$. Otherwise,

$$2^\ell x < |p_i p_{i+1}| < |p_i p_{i+1}| + |p_{i+1} p_k| \le c|p_i p_k|, \text{ or } \frac{2^\ell x}{c} < |p_i p_k|.$$

Consequently, the disks of radius

$$R = \frac{2^\ell x}{2c} = \frac{4 \cdot 2^\ell c}{\sqrt{n-1}} \tag{5}$$

centered at the points in P_ℓ are interior-disjoint. The area of each disk is πR^2. Since $P_\ell \subset U$, these disks are contained in the R-neighborhood U_R of the square U, i.e., the Minkowski sum $R + U$. For $\ell \le \log(c/x)$, we have $2^\ell x \le c$, hence $R = \frac{2^\ell x}{2c} \le \frac{c}{2c} = \frac{1}{2} \le \frac{c}{2}$. Then we can bound the area of U_R from above as follows:

$$\text{area}(U_R) < (c + 2R)^2 \le (2c)^2 = 4c^2. \tag{6}$$

Since U_R contains $|P_\ell|$ interior-disjoint disks of radius R, we obtain

$$|P_\ell| \le \frac{\text{area}(U_R)}{\pi R^2} < \frac{4c^2}{\pi R^2} = \frac{16c^4}{\pi 2^{2\ell} x^2}. \tag{7}$$

For every segment $p_{i-1}p_i$ with length more than x, we have that $p_i \in P_\ell$, for some $\ell \in \{0, 1, \dots, \lfloor \log(c/x) \rfloor\}$. The total length of these segments is

$$\begin{aligned} L_1 &\le \sum_{\ell=0}^{\lfloor \log(c/x) \rfloor} |P_\ell| \cdot 2^{\ell+1} x < \sum_{\ell=0}^{\lfloor \log(x/c) \rfloor} \frac{16c^4}{\pi 2^{2\ell} x^2} \cdot 2^{\ell+1} x = \sum_{\ell=0}^{\lfloor \log(x/c) \rfloor} \frac{32c^4}{\pi 2^\ell x} \\ &< \frac{32c^4}{\pi x} \sum_{\ell=0}^{\infty} \frac{1}{2^\ell} = \frac{64c^4}{\pi x} = \frac{8c^2}{\pi} \cdot \sqrt{n-1}, \end{aligned}$$

as required. Together with (3), this yields $L \le 8\left(1 + c^2/\pi\right) \cdot \sqrt{n-1}$. ◀

3 Lower Bounds

We now present our lower bound construction, showing that the dependence on n for the stretch factor of a c-chain cannot be avoided.

▶ **Theorem 4.** *For every constant $c \ge 4$, there is a set $\mathcal{P}_c = \{P^k\}_{k \in \mathbb{N}}$ of simple c-chains, so that P^k has $n = 4^k + 1$ vertices and stretch factor $(n-1)^{\frac{1+\log(c-2)-\log c}{2}}$.*

By Theorem 3, the stretch factor of a c-chain in the plane is $O\left((n-1)^{1/2}\right)$ for every constant $c \ge 1$. Since

$$\lim_{c \to \infty} \frac{1 + \log(c-2) - \log c}{2} = \frac{1}{2},$$

our lower bound construction shows that the limit of the exponent cannot be improved. Indeed, for every $\varepsilon > 0$, we can set $c = \frac{2^{2\varepsilon+1}}{2^{2\varepsilon}-1}$, and then the chains above have stretch factor $(n-1)^{\frac{1+\log(c-2)-\log c}{2}} = (n-1)^{1/2-\varepsilon} = \Omega(n^{1/2-\varepsilon})$.

We first construct a family $\mathcal{P}_c = \{P^k\}_{k \in \mathbb{N}}$ of polygonal chains. Then we show, in Lemmata 5 and 6, that every chain in $\mathcal{P}_c$ is simple and indeed a c-chain. The theorem follows since the claimed stretch factor is a consequence of the construction.

Construction of $\mathcal{P}_c$. The construction here is a generalization of the iterative construction of the *Koch curve*; when $c = 6$, the result is the original Cesàro fractal (which is a variant of the Koch curve) [10]. We start with a unit line segment P^0, and for $k = 0, 1, \dots,$ we construct P^{k+1} by replacing each segment in P^k by four segments such that the middle three points achieve a stretch factor of $c_* = \frac{c-2}{2}$ (this choice will be justified in the proof of Lemma 6). Note that $c_* \geq 1$, since $c \geq 4$.

We continue with the details. Let P^0 be the unit line segment from $(0, 0)$ to $(1, 0)$; see Figure 3 (left). Given the polygonal chain P^k ($k = 0, 1, \dots$), we construct P^{k+1} by replacing each segment of P^k by four segments as follows. Consider a segment of P^k, and denote its length by ℓ. Subdivide this segment into three segments of lengths $(\frac{1}{2} - \frac{a}{c_*})\ell$, $\frac{2a}{c_*}\ell$, and $(\frac{1}{2} - \frac{a}{c_*})\ell$, respectively, where $0 < a < \frac{c_*}{2}$ is a parameter to be determined later. Replace the middle segment with the top part of an isosceles triangle of side length $a\ell$. The chains P^0, P^1, P^2, and P^4 are depicted in Figures 3 and 4.

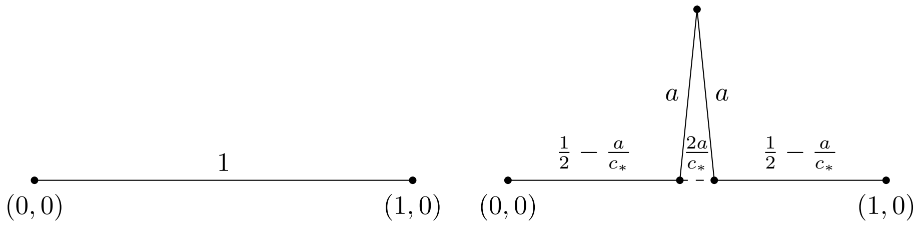


Figure 3 The chains P^0 (left) and P^1 (right).

Note that each segment of length ℓ in P^k is replaced by four segments of total length $(1 + \frac{2a(c_*-1)}{c_*})\ell$. After k iterations, the chain P^k consists of 4^k line segments of total length $\left(1 + \frac{2a(c_*-1)}{c_*}\right)^k$.

By construction, the chain P^k (for $k \geq 1$) consists of four scaled copies of P^{k-1}. For $i = 1, 2, 3, 4$, let the *ith subchain of* P^k be the subchain of P^k consisting of 4^{k-1} segments starting from the $((i-1)4^{k-1} + 1)$th segment. By construction, the ith subchain of P^k is similar to the chain P^{k-1}, for $i = 1, 2, 3, 4$.[3] The following functions allow us to refer to these subchains formally. For $i = 1, 2, 3, 4$, define a function $f_i^k : P^k \to P^k$ as the identity on the ith subchain of P^k that sends the remaining part(s) of P^k to the closest endpoint(s) along this subchain. So $f_i^k(P^k)$ is similar to P^{k-1}. Let $g_i : \mathcal{P}_c \setminus \{P^0\} \to \mathcal{P}_c$ be a piecewise defined function such that $g_i(C) = \sigma^{-1} \circ f_i^k \circ \sigma(C)$ if C is similar to P^k, where $\sigma : C \to P^k$ is a similarity transformation. Applying the function g_i on a chain P^k can be thought of as "cutting out" its ith subchain.

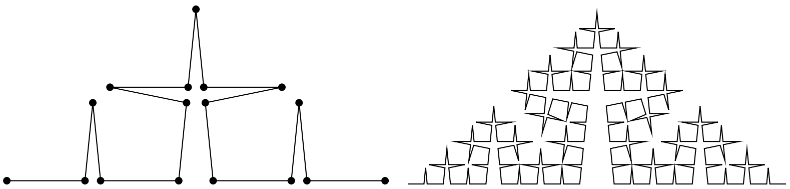


Figure 4 The chains P^2 (left) and P^4 (right).

[3] Two geometric shapes are *similar* if one can be obtained from the other by translation, rotation, and scaling; and are *congruent* if one can be obtained from the other by translation and rotation.

Clearly, the stretch factor of the chain monotonically increases with the parameter a. However, if a is too large, the chain is no longer simple. The following lemma gives a sufficient condition for the constructed chains to avoid self-crossings.

▶ **Lemma 5.** *For every constant $c \geq 4$, if $a \leq \frac{c-2}{2c}$, then every chain in $\mathcal{P}_c$ is simple.*

Proof. Let $T = \text{conv}(P^1)$. Observe that T is an isosceles triangle; see Figure 5 (left). We first show the following:

▷ Claim. If $a \leq \frac{c-2}{2c}$, then $\text{conv}(P^k) = T$ for all $k \geq 1$.

Proof. We prove the claim by induction on k. It holds for $k = 1$ by definition. For the induction step, assume that $k \geq 2$ and that the claim holds for $k-1$. Consider the chain P^k. Since it contains all the vertices of P^1, $T \subset \text{conv}(P^k)$. So we only need to show that $\text{conv}(P^k) \subset T$.

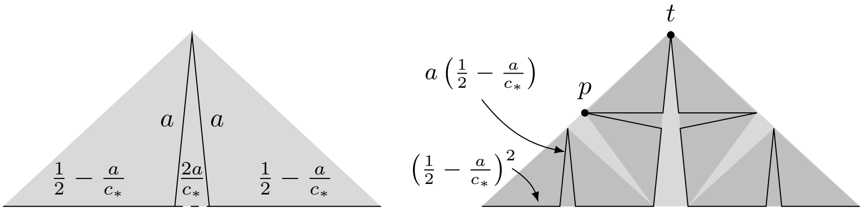

a
a
$\frac{1}{2} - \frac{a}{c_*}$
$\frac{2a}{c_*}$
$\frac{1}{2} - \frac{a}{c_*}$
t
$a\left(\frac{1}{2} - \frac{a}{c_*}\right)$
p
$\left(\frac{1}{2} - \frac{a}{c_*}\right)^2$

Figure 5 Left: Convex hull T of P^1 in light gray; Right: Convex hulls of $g_i(P^2)$, $i = 1, 2, 3, 4$, in dark gray, are contained in T.

By construction, $P^k \subset \bigcup_{i=1}^{4} \text{conv}(g_i(P^k))$; see Figure 5 (right). By the inductive hypothesis, $\text{conv}(g_i(P^k))$ is an isosceles triangle similar to T, for $i = 1, 2, 3, 4$. Since the bases of $\text{conv}(g_1(P^k))$ and $\text{conv}(g_4(P^k))$ are collinear with the base of T by construction, due to similarity, they are contained in T. The base of $\text{conv}(g_2(P^k))$ is contained in T. In order to show $\text{conv}(g_2(P^k)) \subset T$, by convexity, it suffices to ensure that its apex p is also in T. Note that the coordinates of the top point is $t = \left(1/2, a\sqrt{c_*^2 - 1}/c_*\right)$, so the supporting line ℓ of the left side of T is

$$y = \frac{2a\sqrt{c_*^2 - 1}}{c_*}x, \text{ and}$$

$$p = \left(\frac{1}{2} - \frac{a}{2c_*} - \frac{a^2\left(c_*^2 - 1\right)}{c_*^2}, \left(\frac{a}{2c_*} + \frac{a^2}{c_*^2}\right)\sqrt{c_*^2 - 1}\right).$$

By the condition of $a \leq \frac{c-2}{2c} = \frac{c_*}{2(c_*+1)}$ in the lemma, p lies on or below ℓ. Under the same condition, we have $\text{conv}(g_3(P^k)) \subset T$ by symmetry. Then $P^k \subset \bigcup_{i=1}^{4} \text{conv}(g_i(P^k)) \subset T$. Since T is convex, $\text{conv}(P^k) \subset T$. So $\text{conv}(P^k) = T$, as claimed. ◁

We can now finish the proof of Lemma 5 by induction. Clearly, P^0 and P^1 are simple. Assume that $k \geq 2$, and P^{k-1} is simple. Consider the chain P^k. For $i = 1, 2, 3, 4$, $g_i(P^k)$ is similar to P^{k-1}, hence simple by the inductive hypothesis. Since $P^k = \bigcup_{i=1}^{4} g_i(P^k)$, it is sufficient to show that for all $i, j \in \{1, 2, 3, 4\}$, where $i \neq j$, a segment in $g_i(P^k)$ does not intersect any segments in $g_j(P^k)$, unless they are consecutive in P^k and they intersect at a common endpoint. This follows from the above claim together with the observation that for $i \neq j$, the intersection $g_i(P^k) \cap g_j(P^k)$ is either empty or contains a single vertex which is the common endpoint of two consecutive segments in P^k. ◀

In the remainder of this section, we assume that

$$a = \frac{c-2}{2c} = \frac{c_*}{2(c_*+1)}. \tag{8}$$

Under this assumption, all segments in P^1 have the same length a. Therefore, by construction, all segments in P^k have the same length

$$a^k = \left(\frac{c_*}{2(c_*+1)}\right)^k.$$

There are 4^k segments in P^k, with 4^k+1 vertices, and its stretch factor is

$$\delta_{P^k} = 4^k \left(\frac{c_*}{2(c_*+1)}\right)^k = \left(\frac{2c_*}{c_*+1}\right)^k.$$

Consequently, $k = \log_4(n-1) = \frac{\log(n-1)}{2}$, and

$$\delta_{P^k} = \left(\frac{2c_*}{c_*+1}\right)^{\frac{\log(n-1)}{2}} = \left(\frac{2c-4}{c}\right)^{\frac{\log(n-1)}{2}} = (n-1)^{\frac{1+\log(c-2)-\log c}{2}},$$

as claimed. To finish the proof of Theorem 4, it remains to show the constructed polygonal chains are indeed c-chains.

▶ **Lemma 6.** *For every constant $c \geq 4$, $\mathcal{P}_c$ is a family of c-chains.*

We first prove a couple of facts that will be useful in the proof of Lemma 6. We defer an intuitive explanation until after the formal statement of the lemma.

▶ **Lemma 7.** *Let $k \geq 1$ and let $P^k = (p_1, p_2, \ldots, p_n)$, where $n = 4^k+1$. Then the following hold:*

(i) *There exists a sequence $(q_1, q_2, \ldots, q_m)$ of $m = 2 \cdot 4^{k-1}$ points in $\mathbb{R}^2$ such that the chain $R^k = (p_1, q_1, p_2, q_2, \ldots, p_m, q_m, p_{m+1})$ is similar to P^k.*

(ii) *For $k \geq 2$, define $g_5 : \mathcal{P}_c \setminus \{P^0, P^1\} \to \mathcal{P}_c$ by*

$$g_5(P^k) = \left(g_3 \circ g_2(P^k)\right) \cup \left(g_4 \circ g_2(P^k)\right) \cup \left(g_1 \circ g_3(P^k)\right) \cup \left(g_2 \circ g_3(P^k)\right).$$

Then $g_5(P^k)$ is similar to P^{k-1}.

Part (i) of Lemma 7 says that given P^k, we can construct a chain R^k similar to P^k by inserting one point between every two consecutive points of the left half of P^k, see Figure 6 (left). Part (ii) says that the "top" subchain of P^k that consists of the right half of $g_2(P^k)$ and the left half of $g_3(P^k)$, see Figure 6 (right), is similar to P^{k-1}.

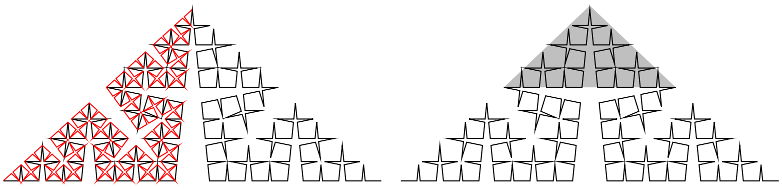


Figure 6 Left: Chain P^k with the scaled copy of itself R^k (in red); Right: Chain P^k with its subchain $g_5(P^k)$ marked by its convex hull.

Proof of Lemma 7. For (i), we review the construction of P^k, and show that R^k and P^k can be constructed in a coupled manner. In Figure 7 (left), consider $P^1 = (p_1, p_2, p_3, p_4, p_5)$. Recall that all segments in P^1 are of the same length $a = \frac{c_*}{2(c_*+1)}$. The isosceles triangles $\Delta p_1 p_2 p_3$ and $\Delta p_1 p_3 p_5$ are similar. Let $\sigma : \Delta p_1 p_3 p_5 \to \Delta p_1 p_2 p_3$ be the similarity transformation. Let $q_1 = \sigma(p_2)$ and $q_2 = \sigma(p_4)$. By construction, the chain $R^1 = (p_1, q_1, p_2, q_2, p_3)$ is similar to P^1. In particular, all of its segments have the same length. So the isosceles triangle $\Delta p_1 q_1 p_2$ is similar to $\Delta p_1 p_3 p_5$. Moreover, its base is the segment $p_1 p_2$, so $\Delta p_1 q_1 p_2$ is precisely $\mathrm{conv}(g_1(P^2))$, see Figure 7 (right).

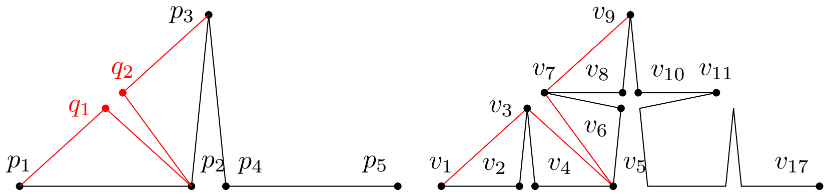


Figure 7 Left: the chains P^1 and R^1 (red); Right: the chains P^2 and R^1 (red).

Write $P^2 = (v_1, v_2, \ldots, v_{17})$, then $v_3 = q_1$ by the above argument and $v_7 = q_2$ by symmetry. Now $\Delta v_1 v_2 v_3$, $\Delta v_3 v_4 v_5$, $\Delta v_5 v_6 v_7$, and $\Delta v_7 v_8 v_9$ are four congruent isosceles triangles, all of which are similar to $\Delta v_1 v_9 v_{17}$, since the angles are the same. Repeat the above procedure on each of them to obtain $R^2 = (v_1, u_1, v_2, u_2, \ldots, v_8, u_8, v_9)$, which is similar to P^2. Continue this construction inductively to get the desired chain R^k for any $k \geq 1$.

For (ii), see Figure 7 (right). By definition, $g_5(P^2)$ is the subchain $(v_7, v_8, v_9, v_{10}, v_{11})$. Observe that the segments $v_7 v_8$ and $v_{10} v_{11}$ are collinear by symmetry. Moreover, they are parallel to $v_1 v_{17}$ since $\angle v_7 v_8 v_9 = \angle v_1 v_5 v_9$. So $g_5(P^2)$ is similar to P^1; see Figure 7 (left). Then for $k \geq 2$, $g_5(P^k)$ is the subchain of P^k starting at vertex v_7, ending at vertex v_{11}. By the construction of P^k, $g_5(P^k)$ is similar to P^{k-1}. ◀

Due to space constraints, the proof of Lemma 6 is deferred to the full version.

4 Algorithm for Recognizing c-Chains

In this section, we design a randomized Las Vegas algorithm to recognize c-chains. More precisely, given a polygonal chain $P = (p_1, \ldots, , p_n)$, and a parameter $c \geq 1$, the algorithm decides whether P is a c-chain, in $O\left(n^{2.5} \text{ polylog } n\right)$ expected time. By definition, $P = (p_1, \ldots, p_n)$ is a c-chain if $|p_i p_j| + |p_j p_k| \leq c\ |p_i p_k|$ for all $1 \leq i < j < k \leq n$; equivalently, p_j lies in the ellipse of major axis c with foci p_i and p_k. Consequently, it suffices to test, for every pair $1 \leq i < k \leq n$, whether the ellipse of major axis $c|p_i p_k|$ with foci p_i and p_k contains p_j, for all j, $i < j < k$. For this, we can apply recent results from geometric range searching.

▶ **Theorem 8.** *There is a randomized algorithm that can decide, for a polygonal chain $P = (p_1, \ldots, p_n)$ in $\mathbb{R}^2$ and a threshold $c > 1$, whether P is a c-chain in $O\left(n^{2.5} \text{ polylog } n\right)$ expected time and $O(n \log n)$ space.*

Agarwal, Matoušek and Sharir [2, Theorem 1.4] constructed, for a set S of n points in $\mathbb{R}^2$, a data structure that can answer ellipse range searching queries: it reports the number of points in S that are contained in a query ellipse. In particular, they showed that, for every $\varepsilon > 0$, there is a constant B and a data structure with $O(n)$ space, $O\left(n^{1+\varepsilon}\right)$ expected preprocessing time, and $O\left(n^{1/2}\log^B n\right)$ query time. The construction was later simplified by Matoušek and Patáková [27]. Using this data structure, we can quickly decide whether a given polygonal chain is a c-chain.

Proof of Theorem 8. Subdivide the polygonal chain $P = (p_1, \dots, p_n)$ into two subchains of equal or almost equal sizes, $P_1 = (p_1, \dots, p_{\lceil n/2\rceil})$ and $P_2 = (p_{\lceil n/2\rceil}, \dots, p_n)$; and recursively subdivide P_1 and P_2 until reaching 1-vertex chains. Denote by T the recursion tree. Then, T is a binary tree of depth $\lceil \log n\rceil$. There are at most 2^i nodes at level i; the nodes at level i correspond to edge-disjoint subchains of P, each of which has at most $n/2^i$ edges. Let W_i be the set of subchains on level i of T; and let $W = \bigcup_{i\geq 0} W_i$. We have $|W| \leq 2n$.

For each polygonal chain $Q \in W$, construct an ellipse range searching data structure $\mathrm{DS}(Q)$ described above [2] for the vertices of Q, with a suitable parameter $\varepsilon > 0$. Their overall expected preprocessing time is

$$\sum_{i=0}^{\lceil \log n\rceil} 2^i \cdot O\left(\left(\frac{n}{2^i}\right)^{1+\varepsilon}\right) = O\left(n^{1+\varepsilon}\sum_{i=0}^{\lceil \log n\rceil}\left(\frac{1}{2^i}\right)^{\varepsilon}\right) = O\left(n^{1+\varepsilon}\right),$$

their space requirement is $\sum_{i=0}^{\lceil \log n\rceil} 2^i \cdot O\left(n/2^i\right) = O(n\log n)$, and their query time at level i is $O\left(\left(n/2^i\right)^{1/2} \text{ polylog } \left(n/2^i\right)\right) = O\left(n^{1/2} \text{ polylog } n\right)$.

For each pair of indices $1 \leq i < k \leq n$, we do the following. Let $E_{i,k}$ denote the ellipse of major axis $c|p_ip_k|$ with foci p_i and p_k. The chain $(p_{i+1}, \dots, p_{k-1})$ is subdivided into $O(\log n)$ maximal subchains in W, using at most two subchains from each set W_i, $i = 0, \dots, \lceil \log n\rceil$. For each of these subchains $Q \in W$, query the data structure $\mathrm{DS}(Q)$ with the ellipse $E_{i,k}$. If all queries are positive (i.e., the count returned is $|Q|$ in *all* queries), then P is a c-chain; otherwise there exists j, $i < j < k$, such that $p_j \notin E_{i,k}$, hence $|p_ip_j| + |p_jp_k| > c|p_ip_k|$, witnessing that P is not a c-chain.

The query time over all pairs $1 \leq i < k \leq n$ is bounded above by

$$\begin{aligned}\binom{n}{2}\sum_{i=0}^{2\lceil \log n\rceil} O\left(\left(n/2^i\right)^{1/2} \text{ polylog } \left(n/2^i\right)\right) &= \binom{n}{2}\cdot O\left(n^{1/2} \text{ polylog } n\right)\\ &= O\left(n^{2.5} \text{ polylog } n\right).\end{aligned}$$

This subsumes the expected time needed for constructing the structures $\mathrm{DS}(Q)$, for all $Q \in W$. So the overall running time of the algorithm is $O\left(n^{2.5} \text{ polylog } n\right)$, as claimed. ◀

In the decision algorithm above, only the construction of the data structures $\mathrm{DS}(Q)$, $Q \in W$, uses randomization, which is independent of the value of c. The parameter c is used for defining the ellipses $E_{i,k}$, and the queries to the data structures; this part is deterministic. Hence, we can find the optimal value of c by Meggido's parametric search [28] in the second part of the algorithm.

Meggido's technique reduces an optimization problem to a corresponding decision problem at a polylogarithmic factor increase in the running time. An optimization problem is amenable to this technique if the following three conditions are met [34]: (1) the objective function is monotone in the given parameter; (2) the decision problem can be solved by evaluating bounded-degree polynomials, and (3) the decision problem admits an efficient parallel algorithm (with polylogarithmic running time using polynomial number of processors). All three conditions hold in our case: The area of each ellipse with foci in S monotonically increases with c; the data structure of [27] answers ellipse range counting queries by evaluating polynomials of bounded degree; and the $\binom{n}{2}$ queries can be performed in parallel. Alternatively, Chan's randomized optimization technique [11] is also applicable. Both techniques yield the following result.

▶ **Corollary 9.** *There is a randomized algorithm that can find, for a polygonal chain $P = (p_1, \dots, p_n)$ in $\mathbb{R}^2$, the minimum $c \geq 1$ for which P is a c-chain in $O\left(n^{2.5} \operatorname{polylog} n\right)$ expected time and $O(n \log n)$ space.*

We remark that, for $c = 1$, the test takes $O(n)$ time: it suffices to check whether points $p_3, \dots, p_n$ lie on the line spanned by p_1p_2, in that order.

5 Concluding Remarks

We end with some final observations and pointers for further research.

1. For $k \geq 1$, let $P_*^k = g_2(P^k) \cup g_3(P^k)$, see Figure 8 (right). It is easy to see that P_*^k is a c-chain with $n = 4^k/2 + 1$ vertices and has stretch factor $\sqrt{c(c-2)/8}(n-1)^{\frac{1+\log(c-2)-\log c}{2}}$. Since $\sqrt{c(c-2)/8} \geq 1$ for $c \geq 4$, this improves the result of Theorem 4 by a constant factor. Since this construction does not improve the exponent, and the analysis would be longer (requiring a case analysis without new insights), we omit the details.

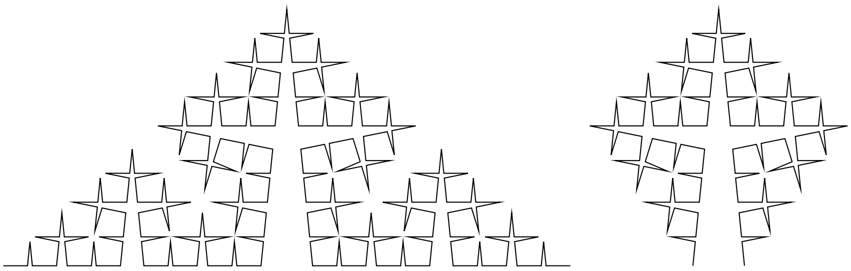


Figure 8 The chains P^4 (left) and P_*^4 (right).

2. If c is used instead of $c_* = (c-2)/2$ in the lower bound construction, then the condition $c \geq 4$ in Theorem 4 can be replaced by $c \geq 1$, and the bound can be improved from $(n-1)^{\frac{1+\log(c-2)-\log c}{2}}$ to $(n-1)^{\frac{1+\log c-\log(c+1)}{2}}$. However, we were unable to prove that the resulting P^k's, $k \in \mathbb{N}$, are c-chains, although a computer program has verified that the first few generations of them are indeed c-chains.
3. The volume argument in Theorem 3 easily generalizes to higher dimensions. If P be a c-chain in $\mathbb{R}^d$ for fixed $c \geq 1$ and $d \geq 2$, then $\delta_P = O\left(c^2(n-1)^{1-1/d}\right)$. It is interesting to find out whether extra dimension(s) allows one to achieve a larger stretch factor.
4. The upper bounds in Theorem 1–3 are valid regardless of whether the chain is crossing or not. On the other hand, the lower bound in Theorem 4 is given by noncrossing chains. A natural question is whether a sharper upper bound holds if the chains are required to be noncrossing. More specifically, can the exponent of n in the upper bound be reduced to $1/2 - \varepsilon$, where $\varepsilon > 0$ depends on c?
5. Our algorithm in Section 4 can recognize c-chains with n vertices in $O\left(n^{2.5} \operatorname{polylog} n\right)$ expected time and $O(n \log n)$ space, using ellipse range searching data structures. It is likely that the running time can be improved in the future, perhaps at the expense of increased space, when suitable time-space trade-offs for semi-algebraic range searching become available. The existence of such data structures is conjectured [2], but currently remains open.

References

1 Pankaj K. Agarwal, Rolf Klein, Christian Knauer, Stefan Langerman, Pat Morin, Micha Sharir, and Michael A. Soss. Computing the Detour and Spanning Ratio of Paths, Trees, and Cycles in 2D and 3D. *Discrete & Computational Geometry*, 39(1-3):17–37, 2008. doi:10.1007/s00454-007-9019-9.

2 Pankaj K. Agarwal, Jiří Matoušek, and Micha Sharir. On Range Searching with Semialgebraic Sets. II. *SIAM J. Computing*, 42(6):2039–2062, 2013. doi:10.1137/120890855.

3 Oswin Aichholzer, Franz Aurenhammer, Christian Icking, Rolf Klein, Elmar Langetepe, and Günter Rote. Generalized self-approaching curves. *Discrete Applied Mathematics*, 109(1-2):3–24, 2001. doi:10.1016/S0166-218X(00)00233-X.

4 Soroush Alamdari, Timothy M. Chan, Elyot Grant, Anna Lubiw, and Vinayak Pathak. Self-approaching Graphs. In Walter Didimo and Maurizio Patrignani, editors, *Proc. 20th Symposium on Graph Drawing (GD)*, volume 7704 of *LNCS*, pages 260–271, Berlin, 2012. Springer. doi:10.1007/978-3-642-36763-2_23.

5 Sanjeev Arora, László Lovász, Ilan Newman, Yuval Rabani, Yuri Rabinovich, and Santosh Vempala. Local Versus Global Properties of Metric Spaces. *SIAM J. Computing*, 41(1):250–271, 2012. doi:10.1137/090780304.

6 Prosenjit Bose, Joachim Gudmundsson, and Michiel H. M. Smid. Constructing Plane Spanners of Bounded Degree and Low Weight. *Algorithmica*, 42(3-4):249–264, 2005. doi:10.1007/s00453-005-1168-8.

7 Prosenjit Bose, Irina Kostitsyna, and Stefan Langerman. Self-Approaching Paths in Simple Polygons. In Boris Aronov and Matthew J. Katz, editors, *Proc. 33rd Symposium on Computational Geometry (SoCG)*, volume 77 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl, 2017. doi:10.4230/LIPIcs.SoCG.2017.21.

8 Prosenjit Bose and Michiel H. M. Smid. On plane geometric spanners: A survey and open problems. *Computational Geometry: Theory and Applications*, 46(7):818–830, 2013. doi:10.1016/j.comgeo.2013.04.002.

9 Peter Brass, William O. J. Moser, and János Pach. *Research Problems in Discrete Geometry*. Springer, New York, 2005.

10 Ernesto Cesàro. Remarques sur la courbe de von Koch. *Atti della R. Accad. della Scienze fisiche e matem. Napoli*, 12(15), 1905. Reprinted as §228 in *Opere scelte, a cura dell'Unione matematica italiana e col contributo del Consiglio nazionale delle ricerche*, Vol. 2: Geometria, analisi, fisica matematica, Rome, dizioni Cremonese, pp. 464–479, 1964.

11 Timothy M. Chan. Geometric Applications of a Randomized Optimization Technique. *Discrete & Computational Geometry*, 22(4):547–567, 1999. doi:10.1007/PL00009478.

12 Otfried Cheong, Herman J. Haverkort, and Mira Lee. Computing a minimum-dilation spanning tree is NP-hard. *Computational Geometry: Theory and Applications*, 41(3):188–205, 2008. doi:10.1016/j.comgeo.2007.12.001.

13 Fan R. K. Chung and Ron L. Graham. On Steiner trees for bounded point sets. *Geometriae Dedicata*, 11(3):353–361, 1981. doi:10.1007/BF00149359.

14 Hallard T. Croft, Kenneth J. Falconer, and Richard K. Guy. *Unsolved Problems in Geometry*, volume 2 of *Unsolved Problems in Intuitive Mathematics*. Springer, New York, 1991. doi:10.1007/978-1-4612-0963-8.

15 Gautam Das and Deborah Joseph. Which Triangulations Approximate the Complete Graph? In Hristo Djidjev, editor, *Proc. International Symposium on Optimal Algorithms*, volume 401 of *LNCS*, pages 168–192, Berlin, 1989. Springer. doi:10.1007/3-540-51859-2_15.

16 Adrian Dumitrescu and Minghui Jiang. Minimum rectilinear Steiner tree of n points in the unit square. *Computational Geometry: Theory and Applications*, 68:253–261, 2018. doi:10.1016/j.comgeo.2017.06.007.

17 David Eppstein. Spanning trees and spanners. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, chapter 9, pages 425–461. Elsevier, Amsterdam, 2000.

18 David Eppstein. Beta-skeletons have unbounded dilation. *Computational Geometry: Theory and Applications*, 23(1):43–52, 2002. doi:10.1016/S0925-7721(01)00055-4.

19 László Fejes Tóth. Über einen geometrischen Satz. *Mathematische Zeitschrift*, 46:83–85, 1940.

20 Leonard Few. The shortest path and the shortest road through n points. *Mathematika*, 2(2):141–144, 1955. doi:10.1112/S0025579300000784.

21 Edgar N. Gilbert and Henry O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968. doi:10.1137/0116001.

22 Christian Icking, Rolf Klein, and Elmar Langetepe. Self-approaching curves. *Mathematical Proceedings of the Cambridge Philosophical Society*, 125(3):441–453, 1999.

23 Howard J. Karloff. How Long can a Euclidean Traveling Salesman Tour Be? *SIAM Journal on Discrete Mathematics*, 2(1):91–99, 1989. doi:10.1137/0402010.

24 Rolf Klein, Christian Knauer, Giri Narasimhan, and Michiel H. M. Smid. On the dilation spectrum of paths, cycles, and trees. *Computational Geometry: Theory and Applications*, 42(9):923–933, 2009. doi:10.1016/j.comgeo.2009.03.004.

25 David G. Larman and Peter McMullen. Arcs with increasing chords. *Mathematical Proceedings of the Cambridge Philosophical Society*, 72(2):205–207, 1972. doi:10.1017/S0305004100047022.

26 Jiří Matoušek. *Lectures on Discrete Geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2002. doi:10.1007/978-1-4613-0039-7.

27 Jiří Matoušek and Zuzana Patáková. Multilevel Polynomial Partitions and Simplified Range Searching. *Discrete & Computational Geometry*, 54(1):22–41, 2015. doi:10.1007/s00454-015-9701-2.

28 Nimrod Megiddo. Linear-Time Algorithms for Linear Programming in $\mathbb{R}^3$ and Related Problems. *SIAM J. Computing*, 12(4):759–776, 1983. doi:10.1137/0212052.

29 Joseph S. B. Mitchell and Wolfgang Mulzer. Proximity algorithms. In Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, chapter 32, pages 849–874. CRC Press, Boca Raton, 3rd edition, 2017. doi:10.1201/9781315119601.

30 Giri Narasimhan and Michiel H. M. Smid. Approximating the Stretch Factor of Euclidean Graphs. *SIAM J. Comput.*, 30(3):978–989, 2000. doi:10.1137/S0097539799361671.

31 Giri Narasimhan and Michiel H. M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007. doi:10.1017/CBO9780511546884.

32 Martin Nöllenburg, Roman Prutkin, and Ignaz Rutter. On self-approaching and increasing-chord drawings of 3-connected planar graphs. *Journal of Computational Geometry*, 7(1):47–69, 2016. URL: http://jocg.org/index.php/jocg/article/view/223, doi:10.20382/jocg.v7i1a3.

33 Günter Rote. Curves with increasing chords. *Mathematical Proceedings of the Cambridge Philosophical Society*, 115(1):1–12, 1994. doi:10.1017/S0305004100071875.

34 Jeffrey S. Salowe. Parametric search. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 43, pages 969–982. CRC Press, Boca Raton, 2nd edition, 2004. doi:10.1201/9781420035315.

35 Samuel Verblunsky. On the shortest path through a number of points. *Proceedings of the American Mathematical Society*, 2:904–913, 1951. doi:10.1090/S0002-9939-1951-0045403-1.

Deleting Edges to Restrict the Size of an Epidemic in Temporal Networks

Jessica Enright
Global Academy of Agriculture and Food Security, University of Edinburgh, UK
jessica.enright@ed.ac.uk

Kitty Meeks
School of Computing Science, University of Glasgow, UK
kitty.meeks@glasgow.ac.uk

George B. Mertzios
Department of Computer Science, Durham University, UK
george.mertzios@durham.ac.uk

Viktor Zamaraev
Department of Computer Science, Durham University, UK
viktor.zamaraev@durham.ac.uk

Abstract

Spreading processes on graphs are a natural model for a wide variety of real-world phenomena, including information or behaviour spread over social networks, biological diseases spreading over contact or trade networks, and the potential flow of goods over logistical infrastructure. Often, the networks over which these processes spread are dynamic in nature, and can be modeled with graphs whose structure is subject to discrete changes over time, i.e. with *temporal graphs*. Here, we consider temporal graphs in which edges are available at specified timesteps, and study the problem of deleting edges from a given temporal graph in order to reduce the number of vertices (temporally) reachable from a given starting point. This could be used to control the spread of a disease, rumour, etc. in a temporal graph. In particular, our aim is to find a temporal subgraph in which a process starting at any single vertex can be transferred to only a limited number of other vertices using a temporally-feasible path (i.e. a path, along which the times of the edge availabilities increase). We introduce a natural deletion problem for temporal graphs and we provide positive and negative results on its computational complexity, both in the traditional and the parameterised sense (subject to various natural parameters), as well as addressing the approximability of this problem.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Temporal networks, spreading processes, graph modification, parameterised complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.57

Funding *Kitty Meeks*: supported by a Royal Society of Edinburgh Personal Research Fellowship, funded by the Scottish Government.
George B. Mertzios: partially supported by the EPSRC Grants EP/P020372/1.
Viktor Zamaraev: supported by the EPSRC Grant EP/P020372/1.

Acknowledgements The authors wish to thank Bruno Courcelle and Barnaby Martin for useful discussions and hints on monadic second order logic.

1 Introduction and motivation

A temporal graph is, loosely speaking, a graph that changes with time. A great variety of modern and traditional networks can be modeled as temporal graphs; social networks, wired or wireless networks which change dynamically, transportation networks, and several physical systems are only a few examples of networks that change over time [31, 38]. Due to

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 57; pp. 57:1–57:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

its vast applicability in many areas, this notion of temporal graphs has been studied from different perspectives under various names such as *time-varying* [1,24,44], *evolving* [11,15,22], *dynamic* [14, 27], and *graphs over time* [33]; for a recent attempt to integrate existing models, concepts, and results from the distributed computing perspective see the survey papers [12–14] and the references therein. Mainly motivated by the fact that, due to causality, entities and information in temporal graphs can "flow" only along sequences of edges whose time-labels are increasing, most temporal graph parameters and optimization problems that have been studied so far are based on the notion of temporal paths (see Definition 2 below) and other path-related notions, such as temporal analogues of distance, diameter, reachability, exploration, and centrality [2–4,19,21,35,37]. Recently, non-path temporal graph problems have also been addressed theoretically, including for example temporal variations of coloring [36], vertex cover [5], and maximal cliques [30,49,50].

Inspired by the foundational work of Kempe et al. [32], we adopt a simple model for such time-varying networks, in which the vertex set remains unchanged while each edge is equipped with a set of time-labels.

▶ **Definition 1** (temporal graph). *A* temporal graph *is a pair* (G, λ)*, where* $G = (V, E)$ *is an underlying (static) graph and* $\lambda : E \to 2^{\mathbb{N}}$ *is a* time-labelling *function which assigns to every edge of* G *a set of discrete-time labels.*

For every edge $e \in E$ in the underlying graph G of a temporal graph (G, λ), $\lambda(e)$ denotes the set of time slots at which e is *active* in (G, λ).

Unless stated otherwise, to simplify the presentation of our results we restrict our attention in this paper to temporal graphs in which each edge is assigned a singleton set by the time-labelling function, that is, in which each edge is active at exactly one time.

Spreading processes on networks or graphs are a topic of significant research across network science [7], and a variety of application areas [28, 29], as well as inspiring more theoretical algorithmic work [23]. Part of the motivation for this interest is the usefulness of spreading processes for modelling a variety of natural phenomena, including biological diseases spreading over contact networks, and rumours or news (both fake and real) spreading over information-passing networks. The rise of quantitative approaches in modelling these phenomena is supported by the increasing number and size of network datasets that can be used as denominator graphs on which processes can spread (e.g. human mobility and contact networks [42], agricultural trade networks [39], and social networks [34]). Typically, a vertex in one of these networks represents some entity that has a state in the process (for example, being infected with a disease, or holding a belief), and edges represent contacts over which the state can spread to other vertices.

Our work is partly motivated by the need to control contagion (be it biological or informational) that may spread over contact networks. Data specifying timed contacts that could spread an infectious disease are recorded in a variety of settings, including movements of humans via commuter patterns and airline flights [16], and fine-grained recording of livestock movements between farms in most European nations [40]. There is very strong evidence that these networks play a critical role in large and damaging epidemics, including the 2009 H1N1 influenza pandemic [10] and the 2001 British foot-and-mouth disease epidemic [28]. Because of the key importance of timing in these networks to their capacity to spread disease, methods to assess the susceptibility of temporal graphs and networks to disease incursion have recently become an active area of work within network epidemiology in general, and within livestock network epidemiology in particular [9,41,47,48].

Here, similarly to [20], we focus our attention on deleting edges from (G, λ) in order to limit the temporal connectivity of the remaining temporal subgraph. To this end, the following temporal extension of the notion of a path in a static graph is fundamental [32,35].

▶ **Definition 2** (temporal path). *A* temporal path *from u to v in a temporal graph (G, λ) is a path from u to v in G, composed of edges $e_0, e_1, \ldots, e_k$ such that each edge e_i is assigned a time $t(e_i) \in \lambda(e_i)$, where $t(e_i) < t(e_{i+1})$ for $0 \leq i < k$.*

In many applications, it may be more realistic to generalise our notion of temporal paths so that the time between arriving at and leaving any vertex must fall within some fixed range. For example, in the context of disease transmission, an upper bound on the permitted time between entering and leaving a vertex might represent the time within which an infection would be detected and eliminated (thus ensuring no further transmission). On the other hand, a lower bound might represent the time between individuals being exposed to an infection and becoming infectious themselves. We formalise this as follows:

▶ **Definition 3.** *Let (G, λ) be a temporal graph and let $\alpha \leq \beta \in \mathbb{N}$. An* (α, β)-temporal path *from u to v in (G, λ) is a path from u to v in G, composed of edges $e_0, e_1, \ldots, e_k$, such that each edge e_i, $0 \leq i < k$, is assigned a time $t(e_i)$ from its image in λ, where $\alpha \leq t(e_{i+1}) - t(e_i) \leq \beta$.*

Our contribution

We consider a natural deletion problem for temporal graphs, namely Temporal Reachability Edge Deletion (for short, TR Edge Deletion), as well as its optimisation version, and study its computational complexity, both in the traditional and the parameterised sense, subject to natural parameters. Given a temporal graph (G, λ) and two natural numbers k, h, the goal is to delete at most k edges from (G, λ) such that, for every vertex v of G, there exists a temporal path to at most $h - 1$ other vertices.

In Section 3, we show that TR Edge Deletion is NP-complete, even on very restricted classes of graphs. We give two different reductions. The first shows that, assuming the Exponential Time Hypothesis, it is unlikely that we can improve significantly on a brute-force approach when considering how the running-time depends on the input size and the number of permitted deletions. The second demonstrates that TR Edge Deletion is *para-NP-hard* (i.e. NP-hard even for constant-valued parameters) with respect to each one of the parameters h, maximum degree Δ_G, or lifetime of (G, λ) (i.e. the maximum label assigned by λ to any edge of G).

In Section 4, we turn our attention to approximation algorithms for the optimisation version of the problem, Min TR Edge Deletion, in which the goal is to find a minimum-size set of edges to delete. We begin by describing a polynomial-time algorithm to compute an h-approximation to Min TR Edge Deletion on arbitrary temporal graphs, then show how similar techniques can be applied to compute a c-approximation on inputs in which the underlying graph has cutwidth c. We conclude our consideration of approximation algorithms by showing that in general there is unlikely to be a polynomial-time algorithm to compute any constant-factor approximation, even on temporal graphs of lifetime two.

In Section 5, we consider exact FPT algorithms. Our hardness results show that the problem remains intractable when parameterised by h or Δ_G alone; here we obtain an FPT algorithm by parameterising simultaneously by h, Δ_G and the treewidth $tw(G)$ of the underlying (static) graph G. In doing so, we demonstrate a general framework in which a celebrated result by Courcelle, concerning relational structures with bounded treewidth (see Theorem 14) can be applied to solve problems in temporal graphs.

We note that all of our results can be applied, with minor modifications to the proofs, to the setting of (α, β)-temporal paths.

2 Preliminaries

Given a (static) graph G, we denote by $V(G)$ and $E(G)$ the sets of its vertices and edges, respectively. An edge between two vertices u and v of G is denoted by uv, and in this case u and v are said to be *adjacent* in G. Given a temporal graph (G, λ), where $G = (V, E)$, the maximum label assigned by λ to an edge of G, called the *lifetime* of (G, λ), is denoted by $T(G, \lambda)$, or simply by T when no confusion arises. That is, $T(G, \lambda) = \max\{t \in \lambda(e) : e \in E\}$. Throughout the paper we consider temporal graphs with *finite lifetime* T. Furthermore, we assume that the given labelling λ is arbitrary, i.e. (G, λ) is given with an explicit list of labels for every edge. Thus, the *size* of the input temporal graph (G, λ) is $O\left(|V| + T + \sum_{t=1}^{T} |E_t|\right) = O(n + mT)$: when we are considering temporal graphs in which edges are active at a single timestep, it suffices to only consider the space required to represent the single time assigned to each edge, and thus the size of the temporal graph is $O(n + m \log T)$. We say that an edge $e \in E$ *appears at time* t if $t \in \lambda(e)$, and in this case we call the pair (e, t) a *time-edge* in (G, λ). Given a subset $E' \subseteq E$, we denote by $(G, \lambda) \setminus E'$ the temporal graph (G', λ'), where $G' = (V, E \setminus E')$ and λ' is the restriction of λ to $E \setminus E'$.

We say that a vertex v is *temporally reachable* from u in (G, λ) if there exists a temporal path from u to v. Furthermore we adopt the convention that every vertex v is temporally reachable from itself. The *temporal reachability set* of a vertex u, denoted by $\text{reach}_{G,\lambda}(u)$, is the set of vertices which are temporally reachable from vertex u. The *temporal reachability* of u is the number of vertices in $\text{reach}_{G,\lambda}(u)$. Furthermore, the *maximum temporal reachability* of a temporal graph is the maximum of the temporal reachabilities of its vertices.

In this paper we mainly consider the following problem.

Temporal Reachability Edge Deletion (TR Edge Deletion)

Input: A temporal graph (G, λ), and $k, h \in \mathbb{N}$.
Output: Is there a set $E' \subseteq E(G)$, with $|E'| \leq k$, such that the maximum temporal reachability of $(G, \lambda) \setminus E'$ is at most h?

Note that the problem clearly belongs to NP as a set of edges acts as a certificate (the reachability set of any vertex in a given temporal graph can be computed in polynomial time [3, 32, 35]). It is worth noting here that the (similarly-flavored) deletion problem for finding small separators in temporal graphs was studied recently, namely the problem of removing a small number of vertices from a given temporal graph such that two fixed vertices become temporally disconnected [26, 51].

3 Computational hardness

The main results of this section demonstrate that TR Edge Deletion is NP-complete even under very strong restrictions on the input. Our first result shows that the trivial brute-force algorithm, running in time $n^{\mathcal{O}(k)}$, in which we consider all possible sets of k edges to delete, cannot be significantly improved in general.

▶ **Theorem 4.** TR Edge Deletion *is W[1]-hard when parameterised by the maximum number k of edges that can be removed, even when the input temporal graph has lifetime 2. Moreover, assuming ETH, there is no $f(k)\tau^{o(k)}$ time algorithm for* TR Edge Deletion*, where τ is the size of the input temporal graph.*

The W[1]-hardness reduction of Theorem 4 also implies that the problem TR Edge Deletion is NP-complete, even on temporal graphs with lifetime at most two. We note that, for temporal graphs of lifetime one, the problem is solvable in polynomial time: in this setting, the reachability set of each vertex is precisely its closed neighbourhood, so the problem reduces to that of deleting a set of at most k edges so that every vertex has degree at most $h-1$ which is solvable in polynomial time [43, Theorem 33.4].

We now demonstrate that TR Edge Deletion remains NP-complete on temporal graphs of lifetime two even if the underlying graph has bounded degree and the maximum permitted size of a temporal reachability set is bounded by a constant.

▶ **Theorem 5.** TR Edge Deletion *is NP-complete, even when the maximum temporal reachability h is at most 7 and the input temporal graph (G, λ) has:*

1. *maximum degree Δ_G of the underlying graph G at most 5, and*
2. *lifetime at most 2.*

Therefore TR Edge Deletion *is para-NP-hard with respect to each of the parameters h, Δ_G, and lifetime $T(G, \lambda)$.*

Proof. As we mentioned in Section 2, the problem trivially belongs to NP. Now we give a reduction from the following well-known NP-complete problem [46].

3,4-SAT

Input: A CNF formula Φ with exactly 3 variables per clause, such that each variable appears in at most 4 clauses.
Output: Does there exists a truth assignment satisfying Φ?

Let Φ be an instance of $3,4$-SAT with variables $x_1, \ldots, x_n$, and clauses $C_1, \ldots, C_m$. We may assume without loss of generality that every variable x_i appears at least once negated and at least once unnegated in Φ. Indeed, if a variable x_i appears only negated (resp. unnegated) in Φ, then we can trivially set $x_i = 0$ (resp. $x_i = 1$) and then remove from Φ all clauses where x_i appears; this process would provide an equivalent instance of 3,4-SAT of smaller size. Now we construct an instance $((G, \lambda), k, h)$ of TR Edge Deletion which is a yes-instance if and only if Φ is satisfiable.

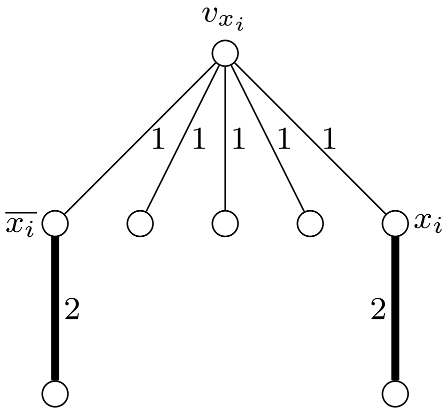


Figure 1 The gadget corresponding to variable x_i. The number beside an edge is the time step at which that edge appears. The bold edges are the ones we refer to as *literal edges*.

We construct (G, λ) as follows. For each variable x_i we introduce in G a copy of the subgraph shown in Figure 1, which we call an *x_i-gadget.* There are three special vertices in an x_i-gadget: x_i and $\overline{x_i}$, which we call *literal vertices*, and v_{x_i} which we call the *head vertex* of the x_i-gadget. All the edges incident to v_{x_i} appear in time step 1, the other two edges of x_i-gadget, which we call *literal edges*, appear in time step 2. Additionally, for every clause

C_s we introduce in G: 1) a *clause vertex* C_s that is adjacent to the three literal vertices corresponding to the literals of C_s, and 2) one more vertex adjacent only to C_s, which we call the *satellite vertex of* C_s. All the new edges incident to C_s appear in time step 1. See Figure 2 for an illustration. Finally, we set $k = n$ and $h = 7$.

First recall that, in Φ, every variable x_i appears at least once negated and at least once unnegated. Therefore, since every variable x_i appears in at most four clauses in Φ, it follows that each of the two vertices corresponding to the literals $x_i, \overline{x_i}$ is connected to at most three clause gadgets. Therefore the degree of each vertex corresponding to a literal in the constructed temporal graph (G, λ) (see Figure 2) is at most five. Moreover, it can be easily checked that the same also holds for every other vertex of (G, λ), and thus $\Delta_{G,\lambda} \leq 5$.

We continue by observing temporal reachabilities of the vertices of (G, λ). A literal vertex can temporally reach only the corresponding clause vertices, and the two neighbours in its gadget. Since every literal belongs to at most 4 clauses in Φ, the temporal reachability of the literal vertex in (G, λ) is at most 7 (including the vertex itself). The head vertex of a gadget temporally reaches only the vertices of the gadget, hence the temporal reachability of any head vertex in (G, λ) is 8. Any other vertex belonging to a gadget can temporally reach only its unique neighbour in G and so has temporal reachability 2. Every clause vertex can reach only the corresponding literal vertices, their neighbours incident to the literal edges, and its own satellite vertex. Hence the temporal reachability of every clause vertex in (G, λ) is 8. Finally, every satellite vertex reaches only its neighbour, and thus its temporal reachability is 2. Therefore in our instance of TR EDGE DELETION we only need to care about temporal reachabilities of the clause and head vertices.

Now we show that, if there is a set E of n edges such that the maximum temporal reachability of the modified graph $(G, \lambda) \setminus E$ is at most 7, then Φ is satisfiable. First, notice that since the temporal reachability of every head vertex is decreased in the modified graph and the number of gadgets is n, the set E contains exactly one edge from every gadget. Hence, as the temporal reachability of every clause vertex C_s is also decreased, set E must contain at least one literal edge that is incident to a literal neighbour of C_s. We now construct a truth assignment as follows: for every literal edge in E we set the corresponding literal to TRUE. If there are unassigned variables left we set them arbitrarily, say, to TRUE.

Since E has one edge in every gadget, every variable was assigned exactly once. Moreover, by the above discussion, every clause has a literal that is set to TRUE by the assignment. Hence the assignment is well-defined and satisfies Φ.

To show the converse, given a truth assignment $(\alpha_1, \ldots, \alpha_n)$ satisfying Φ we construct a set E of n edges such that the maximum temporal reachability of $(G, \lambda) \setminus E$ is at most 7. For every $i \in [n]$ we add to E the literal edge incident to x_i if $\alpha_i = 1$, and the literal edge incident to $\overline{x_i}$ otherwise. By the construction, E has exactly one edge from every gadget. Moreover, since the assignment satisfies Φ, for every clause C_s set E contains at least one literal edge corresponding to one of the literals of C_s. Hence, by removing E from (G, λ), we strictly decrease temporal reachability of every head and clause vertex. ◀

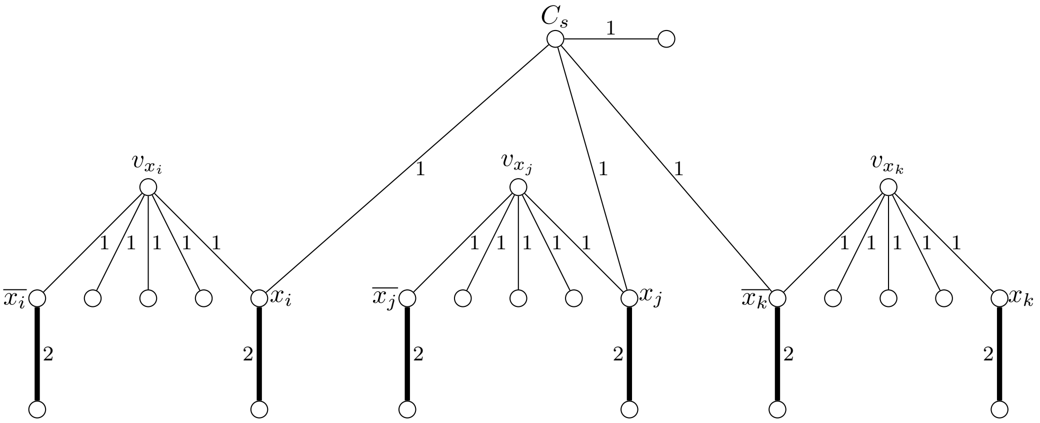


Figure 2 A subgraph of a temporal graph corresponding to an instance of 3,4-SAT.

4 Approximability

Given the strength of the hardness results proved in Section 3, it is natural to ask whether the problem admits efficient approximation algorithms for the following optimisation problem.

Minimum Temporal Reachability Edge Deletion (Min TR Edge Deletion)

Input: A temporal graph (G, λ) and $h \in \mathbb{N}$.
Output: A set X of edges of *minimum* size such that the maximum temporal reachability of $(G, \lambda) \setminus X$ is at most h?

We begin with some more notation. If (G, λ) is a temporal graph and $v \in V(G)$, we say that T is a *reachable subtree for* v if T is a subtree of G, $v \in V(T)$ and, for all $u \in V(T) \setminus \{v\}$, there is a temporal path from v to u in (T, λ'), where λ' is the restriction of λ to the edges of T. We first observe that, if a temporal graph has maximum reachability more than h, we can efficiently find a minimal reachable subtree witnessing this fact.

▶ **Lemma 6.** *Let (G, λ) be a temporal graph, and h a positive integer. There is an algorithm running in polynomial time which, on input $((G, \lambda), h)$,*

1. *if the maximum temporal reachability of (G, λ) is at most h, outputs "YES";*
2. *if the maximum temporal reachability of (G, λ) is greater than h, outputs a vertex $v \in V(G)$ and a reachable subtree T for v where T has exactly $h + 1$ vertices.*

Let h be a positive integer and $(G = (V, E), \lambda)$ be a temporal graph. We say that edge set $E' \subseteq E$ is a *valid deletion* of $(G = (V, E), \lambda)$ with respect to h if the maximum temporal reachability of $(G = (V, E), \lambda) \backslash E'$ is at most h. Where h is clear from the context, we may not refer to it explicitly. We now make a simple observation about valid deletions.

▶ **Lemma 7.** *Let (G, λ) be a temporal graph and h a positive integer. Suppose that T is a reachable subtree for some $v \in V(G)$ and that T has more than h vertices. If $E' \subseteq E(G)$ is a valid deletion with respect to h, then $|E' \cap E(T)| \geq 1$.*

Using these two observations, we now describe our first approximation algorithm.

▶ **Theorem 8.** *There exists a polynomial-time algorithm to compute an h-approximation to* Min TR Edge Deletion, *where h denotes the maximum permitted reachability.*

Proof. Let $((G,\lambda),h)$ be an input instance of Min TR Edge Deletion, and let $E_{opt} \subseteq E$ be a minimum-cardinality edge set such that $(G,\lambda) \setminus E_{\text{opt}}$ has temporal reachability at most h. It suffices to demonstrate that we can find in polynomial time a set $E' \subseteq E$ such that $(G,\lambda) \setminus E'$ has temporal reachability at most h, and $|E'| \leq h|E_{\text{opt}}|$. We claim that the following algorithm achieves this.

1. Initialise $E' := \emptyset$.
2. While (G,λ) has reachability greater than h:
   a. Find a pair (v,T) such that $v \in V(G)$, T is a reachable subtree for v and $|T| = h+1$.
   b. Add $E(T)$ to E', and update $(G,\lambda) \leftarrow (G,\lambda) \setminus E'$.
3. Return E'.

We begin by considering the running time of this algorithm. By Lemma 6 we can determine whether to execute the while loop and, if we do enter the loop, execute Step 2(a), all in polynomial time. Steps 1 and 2(b) can clearly both be carried out in linear time. Moreover, the total number of iterations of the while loop is bounded by the number of edges in G, so we see that the algorithm will terminate in polynomial time.

At every iteration, the algorithm removes exactly h edges, while the optimum deletion set E_{opt} must remove at least one of these h edges. Therefore, in total, we remove at most $h|E_{opt}|$ edges. To complete the proof, we observe that, by construction, the identified set E' is a valid deletion set. ◀

We now demonstrate that we can improve on this general approximation algorithm when the underlying graph has certain useful temporal properties, in particular when the cutwidth is bounded.

The *cutwidth* of a graph $G = (V,E)$ is the minimum integer c such that the vertices of G can be arranged in a linear order $v_1, \ldots, v_n$, called a *layout*, such that for every i with $1 \leq i < n$ the number of edges with one endpoint in $v_1, ..., v_i$ and one in $v_{i+1}, ..., v_n$ is at most c. Given a layout $v_1, v_2, \ldots, v_n$, we say that edges with one endpoint in $v_1, ..., v_i$ and one in $v_{i+1}, ..., v_n$ *span* v_i, v_{i+1}, and say that the maximum number of edges spanning a pair of consecutive vertices is the *cutwidth* of the layout. For any constant c, Thilikos et al. [45] give a linear-time algorithm to generate a layout of cutwidth at most c if one exists.

We can use a similar argument to that in Theorem 8 to give a polynomial-time algorithm to compute a c-approximation to Min TR Edge Deletion, where c is the cutwidth of the input temporal graph. In addition to Lemma 7, we will also make use of the following definition and observation:

Let $G = (V,E)$ be a graph. We say that an edge set $E_S \subseteq E$ is an *edge separator* that separates G into $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$ if, in $G_S = (V, E \backslash E_S)$ no vertex in V_A is reachable from V_B.

▶ **Lemma 9.** *Let h be a positive integer and $(G = (V,E), \lambda)$ be a temporal graph with an edge separator E_S that separates G into $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$. If, for the given h, E'_A and E'_B are valid deletion sets for $(G_A, \lambda|_{E_A})$, $(G_B, \lambda|_{E_B})$, respectively, then $E'_A \cup E'_B \cup E_S$ is a valid deletion set for $(G = (V,E), \lambda)$.*

We now describe a cutwidth approximation algorithm:

▶ **Theorem 10.** *There exists a polynomial-time algorithm to compute a c-approximation to* Min TR Edge Deletion *provided that a layout of cutwidth c is given.*

Proof (Sketch). Let $((G = (V,E), \lambda), h)$ be the input to Min TR Edge Deletion, and suppose that the layout $v_1, \ldots, v_n$ of V, with cutwidth c, is given. We claim that the following algorithm returns a c-approximation to Min TR Edge Deletion in polynomial time:

1. Initialise $E' := \emptyset$.
2. Initialise $i := 0$.
3. While (G, λ) has reachability greater than h:
   a. Find the maximum $j \in \{i, \dots, n\}$ such that the maximum reachability in the subgraph $(G[\{v_i, \dots, v_j\}], \lambda|_{E(G[\{v_i, \dots, v_j\}])})$ is at most h.
   b. Add all edges that span v_j, v_{j+1} to E', and and update $(G, \lambda) \leftarrow (G, \lambda) \setminus E'$.
   c. Update $i \leftarrow j + 1$
4. Return E'. ◀

For any fixed cutwidth c, using the layout generation algorithm given by Thilikos et al. [45] and the algorithm described above, we can give a cutwidth-approximation to Min TR Edge Deletion for graphs with cutwidth c.

▶ **Corollary 11.** *There exists a polynomial-time algorithm to compute a c-approximation to* Min TR Edge Deletion *whenever the cutwidth of the input graph is bounded above by c.*

Note that as paths have cutwidth one, Corollary 11 gives us an exact polynomial-time algorithm for Min TR Edge Deletion on paths.

We conclude this section by demonstrating that there is unlikely to be a polynomial-time algorithm to compute any constant factor approximation to Min TR Edge Deletion in general, even for temporal graphs of lifetime two.

▶ **Theorem 12.** *Unless $P = NP$,* Min TR Edge Deletion *cannot be approximated in polynomial time to within a factor of $(1 - o(1)) \ln \log_2 \sqrt{n}$, where n is the number of vertices in the input temporal graph, even if the input temporal graph has lifetime two.*

5 An exact FPT algorithm

In this section we show that TR Edge Deletion admits an FPT algorithm, when simultaneously parameterised by h, Δ_G, and $tw(G)$, where Δ_G is the maximum degree of G and $tw(G)$ is the treewidth of G. It is worth noting that, although the parameters h and Δ_G may at first seem to be large, parameterising only by these two parameters is not enough, as our results in the previous sections (see e.g. Theorem 5) imply that TR Edge Deletion is para-NP-hard, when simultaneously parameterised by h and Δ_G.

Our results in this section (see Theorem 16) illustrate how a celebrated theorem by Courcelle (see Theorem 14) can be applied to solve temporal graph problems, following a general framework that could potentially be applied to many other temporal problems as well: (i) we define a suitable family τ of relations (i.e. a suitable relational vocabulary) and a Monadic Second Order (MSO) formula ϕ (of length ℓ) that expresses our temporal graph problem at hand; (ii) we represent an arbitrary input temporal graph (G, λ) with an equivalent relational structure $\mathcal{A}$ (of treewidth at most t); (iii) we apply Courcelle's general theorem which solves our problem at hand in time linear to the size of the relational structure $\mathcal{A}$, whenever both ℓ and t are bounded; that is, in time $f(t, \ell) \cdot ||\mathcal{A}||$.

Here, we apply this general framework to the particular problem TR Edge Deletion (by appropriately defining τ, ϕ, and $\mathcal{A}$) such that ℓ only depends on our parameter h, while t only depends on $\mathrm{tw}(G)$ and Δ_G; this yields our FPT algorithm. Here, as it turns out, the size of $\mathcal{A}$ is quadratic on the size of the input temporal graph (G, λ). Before we present the main result of this section (see Section 5.2), we first present in Section 5.1 some necessary background on logic and on tree decompositions of graphs and relational structures. For any undefined notion in Section 5.1, we refer the reader to [25].

5.1 Preliminaries for the algorithm

Treewidth of graphs

Given any tree T, we will assume that it contains some distinguished vertex $r(T)$, which we will call the *root* of T. For any vertex $v \in V(T) \setminus \{r(T)\}$, the *parent* of v is the neighbour of v on the unique path from v to $r(T)$; the set of *children* of v is the set of all vertices $u \in V(T)$ such that v is the parent of u. The *leaves* of T are the vertices of T whose set of children is empty. We say that a vertex u is a *descendant* of the vertex v if v lies somewhere on the unique path from u to $r(T)$. In particular, a vertex is a descendant of itself, and every vertex is a descendant of the root. Additionally, for any vertex v, we will denote by T_v the subtree induced by the descendants of v.

We say that $(T, \mathcal{B})$ is a *tree decomposition* of G if T is a tree and $\mathcal{B} = \{\mathcal{B}_s : s \in V(T)\}$ is a collection of non-empty subsets of $V(G)$ (or *bags*), indexed by the nodes of T, satisfying:

(1) for all $v \in V(G)$, the set $\{s \in T : v \in \mathcal{B}_s\}$ is nonempty and induces a connected subgraph in T,

(2) for every $e = uv \in E(G)$, there exists $s \in V(T)$ such that $u, v \in \mathcal{B}_s$.

The *width* of the tree decomposition $(T, \mathcal{B})$ is defined to be $\max\{|\mathcal{B}_s| : s \in V(T)\} - 1$, and the *treewidth* of G is the minimum width over all tree decompositions of G.

Although it is NP-hard to determine the treewidth of an arbitrary graph [6], the problem of determining whether a graph has treewidth at most w (and constructing such a tree decomposition if it exists) can be solved in linear time for any constant w [8]; note that this running time depends exponentially on w.

▶ **Theorem 13** (Bodlaender [8]). *For each $w \in N$, there exists a linear-time algorithm, that tests whether a given graph $G = (V, E)$ has treewidth at most w, and if so, outputs a tree decomposition of G with treewidth at most w.*

Relational structures and monadic second order logic

A *relational vocabulary* τ is a set of relation symbols. Each relation symbol R has an *arity*, denoted $\text{arity}(R) \geq 1$. A *structure* $\mathcal{A}$ of vocabulary τ, or τ-structure, consists of a set A, called the *universe*, and an interpretation $R^{\mathcal{A}} \subseteq A^{\text{arity(R)}}$ of each relation symbol $R \in \tau$. We write $\overline{a} \in R^{\mathcal{A}}$ or $R^{\mathcal{A}}(\overline{a})$ to denote that the tuple $\overline{a} \in A^{\text{arity}(R)}$ belongs to the relation $R^{\mathcal{A}}$.

We briefly recall the syntax and semantics of first-order logic. We fix a countably infinite set of (*individual*) *variables*, for which we use small letters. *Atomic formulas of vocabulary τ* are of the form:

1. $x = y$ or
2. $R(x_1 \ldots x_r)$,

where $R \in \tau$ is r-ary and $x_1, \ldots, x_r, x, y$ are variables. *First-order formulas* of vocabulary τ are built from the atomic formulas using the Boolean connectives $\neg, \wedge, \vee$ and existential and universal quantifiers $\exists, \forall$.

The difference between first-order and second-order logic is that the latter allows quantification not only over elements of the universe of a structure, but also over subsets of the universe, and even over relations on the universe. In addition to the individual variables of first-order logic, formulas of second-order logic may also contain *relation variables*, each of which has a prescribed arity. Unary relation variables are also called *set variables.* We use capital letters to denote relation variables. To obtain second-order logic, the syntax of first-order logic is enhanced by new atomic formulas of the form $X(x_1 \ldots x_k)$, where X is k-ary relation variable. Quantification is allowed over both individual and relation variables.

A second-order formula is *monadic* if it only contains unary relation variables. Monadic second-order logic is the restriction of second-order logic to monadic formulas. The class of all monadic second-order formulas is denoted by MSO.

A *free variable* of a formula ϕ is a variable x with an occurrence in ϕ that is not in the scope of a quantifier binding x. A *sentence* is a formula without free variables. Informally, we say that a structure $\mathcal{A}$ *satisfies* a formula ϕ if there exists an assignment of the free variables under which ϕ becomes a true statement about $\mathcal{A}$. In this case we will write $\mathcal{A} \models \phi$.

Treewidth of relational structures

The definition of tree decompositions and treewidth generalizes from graphs to arbitrary relational structures in a straightforward way. A *tree decomposition* of a τ-structure $\mathcal{A}$ is a pair $(T, \mathcal{B})$, where T is a tree and $\mathcal{B}$ a family of subsets of the universe A of $\mathcal{A}$ such that:

(1) for all $a \in A$, the set $\{s \in V(T) : a \in \mathcal{B}_s\}$ is nonempty and induces a connected subgraph (i.e. subtree) in T,

(2) for every relation symbol $R \in \tau$ and every tuple $(a_1, \ldots, a_r) \in R^{\mathcal{A}}$, where $r := \text{arity}(R)$, there is a $s \in V(T)$ such that $a_1, \ldots, a_r \in \mathcal{B}_s$.

The *width* of the tree decomposition $(T, \mathcal{B})$ is the number $\max\{|\mathcal{B}_s| : s \in V(T)\} - 1$. The *treewidth* $tw(\mathcal{A})$ of $\mathcal{A}$ is the minimum width over all tree decompositions of $\mathcal{A}$.

We will make use of the version of Courcelle's celebrated theorem for relational structures of bounded treewidth, which, informally, says that the optimization problem definable by an MSO formula can be solved in FPT time with respect to the treewidth of a relational structure. The formal statement is an adaptation of an analogous theorem (see Theorem 9.21 in [18]) for the model-checking problem [17].

▶ **Theorem 14** ([18]). *Let ϕ be an MSO formula with a free set variable E, and let $\mathcal{A}$ be a relational structure on universe A, where $\text{tw}(\mathcal{A}) \leq t$. Then, given a width-$t$ tree decomposition of $\mathcal{A}$, a minimum-cardinality set $E \subseteq A$ such that $\mathcal{A}$ satisfies $\phi(E)$ can be computed in time*

$$f(t, \ell) \cdot ||\mathcal{A}||,$$

where f is a computable function, ℓ is the length of ϕ, and $||\mathcal{A}||$ is the size of $\mathcal{A}$.

5.2 The FPT algorithm

In this section we present an FPT algorithm for TR Edge Deletion when parameterised simultaneously by three parameters: h, $\text{tw}(G)$ and Δ_G. Our strategy is first, given an input temporal graph (G, λ), to construct a relational structure $\mathcal{A}_{G,\lambda}$ whose treewidth is bounded in terms of $\text{tw}(G)$ and Δ_G. Then we construct an MSO formula ϕ_h with a unique free set variable E, such that $\mathcal{A}_{G,\lambda}$ satisfies $\phi_h(E)$ for some $E \subseteq A$ if and only if the maximum reachability of $(G, \lambda) \setminus E$ is at most h. Finally, we apply Theorem 14 to find the minimum cardinality of such a set $E \subseteq A$. If the minimum cardinality is at most k, then $((G, \lambda), k, h)$ is a yes-instance of the problem, otherwise it is a no-instance.

We note that in the case we consider here in which each edge is active at a single timestep the construction below might be simplified slightly; however, in order to demonstrate the flexibility of this general framework, we choose to define a relational structure which would allow us to represent temporal graphs in which edges may be active at more than one timestep. Observe that Theorem 16 can immediately be adapted to this more general context if we replace Δ_G by the maximum temporal total degree of the input temporal graph (i.e. the maximum number of time-edges incident with any vertex).

Given a temporal graph (G, λ), we define a relational structure $\mathcal{A}_{G,\lambda}$ as follows. The ground set $A_{G,\lambda}$ consists of

- the set $V(G)$ of vertices in G,
- the set $E(G)$ of edges in G, and
- the set of all time-edges of (G, λ), i.e. the set $\Lambda(G, \lambda) = \{(e, t) \mid e \in E(G), t \in \lambda(e)\}$.

On this ground set $A_{G,\lambda}$, we define two binary relations $\mathcal{R}$ and $\mathcal{L}$ as follows:

1. $((e_1, t_1), (e_2, t_2)) \in \mathcal{R}$ if and only if the following conditions hold:
   a. $(e_1, t_1), (e_2, t_2) \in \Lambda(G, \lambda)$;
   b. e_1, e_2 share a vertex in G;
   c. $t_1 < t_2$.
2. $(e, (e, t)) \in \mathcal{L}$ if and only if $(e, t) \in \Lambda(G, \lambda)$.

First we show that the treewidth of $\mathcal{A}_{G,\lambda}$ is bounded by a function of $\mathrm{tw}(G)$ and Δ_G.

▶ **Lemma 15.** *The treewidth of $\mathcal{A}_{G,\lambda}$ is at most $(2\Delta_G + 1)(\mathrm{tw}(G) + 1) - 1$.*

Using this, we now provide the main result of this section.

▶ **Theorem 16.** TR Edge Deletion *admits an FPT algorithm with respect to the combined parameters h, $\mathrm{tw}(G)$, and Δ_G.*

6 Conclusions and open problems

In this paper we studied the problem of removing a small number of *edges* from a given *temporal graph* (i.e. a graph that changes over time) to ensure that every vertex has a temporal path to at most h other vertices. The main motivation for this problem comes from the need to limit spreading processes on dynamic graphs. Such a graph could, for example, capture potentially-infectious contacts between individuals, and removing an edge would correspond to restricting or prohibiting contact between two entities in order to limit the spread of an epidemic.

We show that our problem is W[1]-hard when parameterised by the maximum number k of edges that can be removed and, assuming the Exponential Time Hypothesis, we cannot significantly improve on the brute-force algorithm that considers all possible deletions sets of k edges. On the positive side, we prove that this problems admits a fixed-parameter tractable (FPT) algorithm with respect to the combination of three parameters: the treewidth $\mathrm{tw}(G)$ of the underlying graph G, the maximum allowed temporal reachability h, and the maximum degree Δ_G of (G, λ). Moreover, we show that the latter two parameters combined (i.e. without the treewidth $\mathrm{tw}(G)$) are not enough for deriving an FPT algorithm as the problem is para-NP-complete with respect to both of these parameters. On the other hand, it remains open whether this problem is FPT, when parameterised by treewidth $\mathrm{tw}(G)$, combined with only one of the other two parameters h and Δ_G. We also consider the approximability of this problem, and give two polynomial-time approximation algorithms. The first computes an h-approximation on an arbitrary input graph, where h denotes the maximum allowable temporal reachability, and the second computes a c-approximation on graphs of cutwidth c. We complement these positive results by showing that no constant-factor approximation algorithm exists for general input graphs unless $P = NP$. A natural open problem is whether we can improve these approximation algorithms. Our lower bound rules out a $(\log \log h)$-factor approximation, but a significant improvement on our factor h approximation may be possible.

References

1 Eric Aaron, Danny Krizanc, and Elliot Meyerson. DMVP: foremost waypoint coverage of time-varying graphs. In *Proceedings of the 40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 29–41, 2014.

2 E. Akrida, G.B. Mertzios, S. Nikoletseas, C. Raptopoulos, P.G. Spirakis, and V. Zamaraev. *How fast can we reach a target vertex in stochastic temporal graphs?*, 2019. Technical Report available at `https://arxiv.org/abs/1903.03636`.

3 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. Ephemeral networks with random availability of links: The case of fast networks. *Journal of Parallel and Distributed Computing*, 87:109–120, 2016.

4 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. The complexity of optimal design of temporally connected graphs. *Theory of Computing Systems*, 61(3):907–944, 2017.

5 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP)*, 2018. To appear. Technical Report available at `https://arxiv.org/abs/1802.07103`.

6 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277—-284, 1987.

7 Alain Barrat, Marc Barthlemy, and Alessandro Vespignani. *Dynamical Processes on Complex Networks*. Cambridge University Press, New York, NY, USA, 1st edition, 2008.

8 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 226–234, 1993.

9 Alfredo Braunstein and Alessandro Ingrosso. Inference of causality in epidemics on temporal contact networks. *Scientific Reports*, 6:27538, 2016. `doi:10.1038/srep27538`.

10 Dirk Brockmann and Dirk Helbing. The Hidden Geometry of Complex, Network-Driven Contagion Phenomena. *Science*, 342(6164):1337–1342, 2013. `doi:10.1126/science.1245200`.

11 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing Shortest, Fastest, and Foremost Journeys in Dynamic Networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.

12 Arnaud Casteigts and Paola Flocchini. Deterministic Algorithms in Dynamic Networks: Formal Models and Metrics. Technical report, Defence R&D Canada, April 2013. URL: `https://hal.archives-ouvertes.fr/hal-00865762`.

13 Arnaud Casteigts and Paola Flocchini. Deterministic Algorithms in Dynamic Networks: Problems, Analysis, and Algorithmic Tools. Technical report, Defence R&D Canada, April 2013. URL: `https://hal.archives-ouvertes.fr/hal-00865764`.

14 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)*, 27(5):387–408, 2012.

15 Andrea E. F. Clementi, Claudio Macci, Angelo Monti, Francesco Pasquale, and Riccardo Silvestri. Flooding Time of Edge-Markovian Evolving Graphs. *SIAM Journal on Discrete Mathematics (SIDMA)*, 24(4):1694–1712, 2010.

16 Vittoria Colizza, Alain Barrat, Marc Barthélemy, and Alessandro Vespignani. The role of the airline transportation network in the prediction and predictability of global epidemics. *Proceedings of the National Academy of Sciences*, 103(7):2015–2020, 2006. `doi:10.1073/pnas.0510525103`.

17 Bruno Courcelle, 2018. Personal communication.

18 Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*. Cambridge University Press, 2012.

19 Jessica Enright and Kitty Meeks. Changing times to optimise reachability in temporal graphs. Technical Report available at `https://arxiv.org/abs/1802.05905`.

20 Jessica Enright and Kitty Meeks. Deleting edges to restrict the size of an epidemic: a new application for treewidth. *Algorithmica*, 80(6):1857–1889, 2018.

21 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On Temporal Graph Exploration. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 444–455, 2015.

22 Afonso Ferreira. Building a Reference Combinatorial Model for MANETs. *IEEE Network*, 18(5):24–29, 2004.

23 Stephen Finbow and Gary MacGillivray. The Firefighter Problem: a survey of results, directions and questions. *Australasian J. Combinatorics*, 43:57–78, 2009. URL: `http://ajc.maths.uq.edu.au/pdf/43/ajc_v43_p057.pdf`.

24 Paola Flocchini, Bernard Mans, and Nicola Santoro. Exploration of Periodically Varying Graphs. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 534–543, 2009.

25 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

26 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. In *44th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 2018. To appear. Technical Report available at `https://arxiv.org/abs/1803.00882`.

27 George Giakkoupis, Thomas Sauerwald, and Alexandre Stauffer. Randomized Rumor Spreading in Dynamic Graphs. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 495–507, 2014.

28 Daniel T. Haydon, Rowland R. Kao, and Paul R. Kitching. The UK foot-and-mouth disease outbreak—the aftermath. *Nature Reviews Microbiology*, 2(8):675, 2004.

29 Itai Himelboim, Marc A. Smith, Lee Rainie, Ben Shneiderman, and Camila Espina. Classifying Twitter Topic-Networks Using Social Network Analysis. *Social Media + Society*, 3(1):2056305117691545, 2017. `doi:10.1177/2056305117691545`.

30 Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the Bron-Kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining*, 7(1):35:1–35:16, 2017.

31 Petter Holme and Jari Saramäki, editors. *Temporal Networks*. Springer, 2013.

32 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the 32nd annual ACM symposium on Theory of computing (STOC)*, pages 504–513, 2000.

33 Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 2007.

34 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

35 G.B. Mertzios, O. Michail, I. Chatzigiannakis, and P.G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, pages 1416–1449, 2019.

36 George B Mertzios, Hendrik Molter, and Viktor Zamaraev. Sliding window temporal graph coloring. In *Proceedings of the 33st AAAI Conference on Artificial Intelligence (AAAI)*, (to appear).

37 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016.

38 Othon Michail and Paul G. Spirakis. Elements of the Theory of Dynamic Networks. *Communications of the ACM*, 61(2):72–72, 2018.

39 A. Mitchell, D. Bourn, J. Mawdsley, W. Wint, R. Clifton-Hadley, and M. Gilbert. Characteristics of cattle movements in Britain – an analysis of records from the Cattle Tracing System. *Animal Science*, 80(3):265?273, 2005. `doi:10.1079/ASC50020265`.

40 Andrew Mitchell, David Bourn, J. Mawdsley, William Wint, Richard Clifton-Hadley, and Marius Gilbert. Characteristics of cattle movements in Britain – an analysis of records from the Cattle Tracing System. *Animal Science*, 80(3):265–273, 2005.

41 Maria Noremark and Stefan Widgren. EpiContactTrace: an R-package for contact tracing during livestock disease outbreaks and for risk-based surveillance. *BMC Veterinary Research*, 10(1), 2014. doi:10.1186/1746-6148-10-71.

42 Piotr Sapiezynski, Arkadiusz Stopczynski, Radu Gatej, and Sune Lehmann. Tracking human mobility using wifi signals. *PloS One*, 10(7):e0130824, 2015.

43 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag Berlin Heidelberg, 2003.

44 John Kit Tang, Mirco Musolesi, Cecilia Mascolo, and Vito Latora. Characterising temporal distance and reachability in mobile and online social networks. *ACM Computer Communication Review*, 40(1):118–124, 2010.

45 Dimitrios M. Thilikos, Maria Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *Journal of Algorithms*, 56(1):1–24, 2005. doi:10.1016/j.jalgor.2004.12.001.

46 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

47 Eugenio Valdano, Luca Ferreri, Chiara Poletto, and Vittoria Colizza. Analytical Computation of the Epidemic Threshold on Temporal Networks. *Phys. Rev. X*, 5:021005, April 2015. doi:10.1103/PhysRevX.5.021005.

48 Eugenio Valdano, Chiara Poletto, Armando Giovannini, Diana Palma, Lara Savini, and Vittoria Colizza. Predicting Epidemic Risk from Past Temporal Contact Data. *PLoS Computational Biology*, 11(3):e1004152, March 2015. doi:10.1371/journal.pcbi.1004152.

49 Jordan Viard, Matthieu Latapy, and Clémence Magnien. Revealing contact patterns among high-school students using maximal cliques in link streams. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1517–1522, 2015.

50 Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, 2016.

51 Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. On efficiently finding small separators in temporal graphs. Technical Report available at https://arxiv.org/abs/1803.00882.

Constant Delay Enumeration with FPT-Preprocessing for Conjunctive Queries of Bounded Submodular Width

Christoph Berkholz
Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany
berkholz@informatik.hu-berlin.de

Nicole Schweikardt
Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany
schweikn@informatik.hu-berlin.de

Abstract

Marx (STOC 2010, J. ACM 2013) introduced the notion of submodular width of a conjunctive query (CQ) and showed that for any class Φ of Boolean CQs of bounded submodular width, the model-checking problem for Φ on the class of all finite structures is fixed-parameter tractable (FPT). Note that for non-Boolean queries, the size of the query result may be far too large to be computed entirely within FPT time. We investigate the free-connex variant of submodular width and generalise Marx's result to non-Boolean queries as follows: For every class Φ of CQs of bounded free-connex submodular width, within FPT-preprocessing time we can build a data structure that allows to enumerate, without repetition and with constant delay, all tuples of the query result. Our proof builds upon Marx's splitting routine to decompose the query result into a union of results; but we have to tackle the additional technical difficulty to ensure that these can be enumerated efficiently.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Finite Model Theory; Theory of computation → Fixed parameter tractability; Theory of computation → Database theory; Theory of computation → Database query processing and optimization (theory)

Keywords and phrases conjunctive queries, constant delay enumeration, hypertree decompositions, submodular width, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.58

Funding Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SCHW 837/5-1.

1 Introduction

In the past decade, starting with Durand and Grandjean [19], the fields of logic in computer science and database theory have seen a large number of contributions that deal with the efficient enumeration of query results. In this scenario, the objective is as follows: given a finite relational structure (i.e., a database) and a logical formula (i.e., a query), after a short preprocessing phase, the query results shall be generated one by one, without repetition, with guarantees on the maximum delay time between the output of two tuples. In this vein, the best that one can hope for is *constant* delay (i.e., the delay may depend on the size of the query but not on that of the input structure) and *linear* preprocessing time (i.e., time $f(\varphi)\cdot O(N)$ where N is the size of a reasonable representation of the input structure, φ is the query, and $f(\varphi)$ is a number only depending on the query but not on the input structure). Constant delay enumeration has also been adopted as a central concept in *factorised databases* that gained recent attention [37, 36].

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 58; pp. 58:1–58:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Quite a number of query evaluation problems are known to admit constant delay algorithms preceded by linear or pseudo-linear time preprocessing. This is the case for all first-order queries, provided that they are evaluated over classes of structures of bounded degree [19, 27, 13, 30], low degree [20], bounded expansion [28], locally bounded expansion [41], and on classes that are nowhere dense [39]. Also different data models have been investigated, including tree-like data and document spanners [8, 29, 7]. Recently, also the *dynamic* setting, where a fixed query has to be evaluated repeatedly against a database that is constantly updated, has received quite some attention [31, 13, 12, 25, 14, 5, 35, 34, 6].

This paper deals with the classical, *static* setting without database updates. We focus on evaluating conjunctive queries (CQs, i.e., primitive-positive formulas) on arbitrary relational structures.[1] In the following, *FPT-preprocessing* (resp., *FPL-preprocessing*) means preprocessing that takes time $f(\varphi)\cdot N^{O(1)}$ (resp., $f(\varphi)\cdot O(N)$), and *constant delay* means delay $f(\varphi)$, where f is a computable function, φ is the query, and N is the size of the input structure.

Bagan et al. [10] showed that every free-connex acyclic CQ allows constant delay enumeration after FPL-preprocessing. More refined results in this vein are due to Bagan [9] and Brault-Baron [16]; see [40] for a survey. Bagan et al. [10] complemented their result by a conditional lower bound: assuming that Boolean matrix multiplication cannot be accomplished in time $O(n^2)$, self-join-free acyclic CQs that are *not* free-connex *cannot* be enumerated with constant delay and FPL-preprocessing. This demonstrates that even if the evaluation of *Boolean* queries is easy (as known for all acyclic CQs [42]), the enumeration of the results of non-Boolean queries might be hard (here, for acyclic CQs that are not free-connex).

Bagan et al. [10] also introduced the notion of *free-connex* (fc) treewidth (tw) of a CQ and showed that for every class Φ of CQs of bounded fc-tw, within FPT-preprocessing time, one can build a data structure that allows constant delay enumeration of the query results. This can be viewed as a generalisation, to the non-Boolean case, of the well-known result stating that the model-checking problem for classes of Boolean CQs of bounded treewidth is FPT. Note that for non-Boolean queries – even if they come from a class of bounded fc-tw – the size of the query result may be $N^{\Omega(\|\varphi\|)}$, i.e., far too large to be computed entirely within FPT-preprocessing time; and generalising the known tractability result for Boolean CQs to the non-Boolean case is far from trivial.

In a series of papers, the FPT-result for Boolean CQs has been strengthened to more and more general width-measures, namely to classes of queries of bounded generalised hypertree width (ghw) [23], bounded fractional hypertree width (fhw) [24], and bounded submodular width (subw) [33]. The result on bounded fhw has been generalised to the non-Boolean case in the context of factorised databases [37], which implies constant delay enumeration after FPT-preprocessing for CQs of bounded free-connex fractional hypertree width (fc-fhw). Related data structures that allow constant delay enumeration after FPT-preprocessing for (quantifier-free) CQs of bounded (fc-)fhw have also been provided in [18, 26].

An analogous generalisation of the result on bounded submodular width, however, is still missing. The present paper's main result closes this gap: we show that on classes of CQs of bounded fc-subw, within FPT-preprocessing time one can build a data structure that allows constant delay enumeration of the query results. And within the same FPT-preprocessing time, one can also construct a data structure that enables to test in constant time whether an input tuple belongs to the query result. Our proof uses Marx's splitting routine [33] to decompose the query result of φ on $\mathcal{A}$ into the union of results of several queries φ_i on several

[1] In this paper, structures will always be finite and relational.

structures $\mathcal{A}_i$ but we have to tackle the additional technical difficulty to ensure that the results of all the φ_i on $\mathcal{A}_i$ can be enumerated efficiently. Once having achieved this, we can conclude by using an elegant trick provided by Durand and Strozecki [21] for enumerating, without repetition, the union of query results.

As an immediate consequence of the lower bound provided by Marx [33] in the context of Boolean CQs of unbounded submodular width, one obtains that our main result is tight for certain classes of CQs, namely, recursively enumerable classes Φ of quantifier-free and self-join-free CQs: assuming the exponential time hypothesis (ETH), such a class Φ allows constant delay enumeration after FPT-preprocessing if, and only if, Φ has bounded fc-subw.

Let us mention a related recent result which, however, is incomparable to ours. Abo Khamis et al. [3] designed an algorithm for evaluating a quantifier-free CQ φ of submodular width w within time $O(N^w)\cdot(\log N)^{f(\varphi)} + O(r\cdot \log N)$; and an analogous result is also achieved for non-quantifier-free CQs of fc-subw w [2]. Here, N is the size of the input structure, r is the number of tuples in the query result, and $f(\varphi)$ is at least exponential in number of variables of φ. In particular, the algorithm does not distinguish between a preprocessing phase and an enumeration phase and does not provide a guarantee on the delay. Also, due to the factor $(\log N)^{f(\varphi)}$, where $f(\varphi)$ is not bounded in terms of w, it seems unlikely that the approach can easily be adapted to yield a method for constant delay enumeration after FPT-preprocessing for classes of CQs of bounded fc-subw.

Outline. The rest of the paper is structured as follows. Section 2 provides basic notations concerning structures, queries, and constant delay enumeration. Section 3 recalls concepts of (free-connex) decompositions of queries, provides a precise statement of our main result, and collects the necessary tools for obtaining this result. Section 4 is devoted to the detailed proof of our main result. We conclude in Section 5. Due to space restrictions, some proof details had to be deferred to the full version of the paper.

2 Preliminaries

In this section we fix notation and summarise basic definitions.

Basic notation. We write $\mathbb{N}$ and $\mathbb{R}_{\geqslant 0}$ for the set of non-negative integers and reals, respectively, and we let $\mathbb{N}_{\geqslant 1} := \mathbb{N} \setminus \{0\}$ and $[n] := \{1, \dots, n\}$ for all $n \in \mathbb{N}_{\geqslant 1}$. By 2^S we denote the power set of a set S. Whenever G denotes a graph, we write $V(G)$ and $E(G)$ for the set of nodes and the set of edges, respectively, of G. Whenever writing a to denote a k-tuple (for some arity $k \in \mathbb{N}$), we write a_i to denote the tuple's i-th component; i.e., $a = (a_1, \dots, a_k)$. For a k-tuple a and indices $i_1, \dots, i_\ell \in [k]$ we let $\pi_{i_1,\dots,i_\ell}(a) := (a_{i_1}, \dots, a_{i_\ell})$. For a set S of k-tuples we let $\pi_{i_1,\dots,i_\ell}(S) := \{\pi_{i_1,\dots,i_\ell}(a) \;:\; a \in S\}$.

If h and g are mappings with domains X and Y, respectively, we say that h and g are *joinable* if $h(z) = g(z)$ holds for all $z \in X \cap Y$. In case that h and g are joinable, we write $h \bowtie g$ to denote the mapping f with domain $X \cup Y$ where $f(x) = h(x)$ for all $x \in X$ and $f(y) = g(y)$ for all $y \in Y$. If A and B are sets of mappings with domains X and Y, respectively, then $A \bowtie B := \{h \bowtie g \;:\; h \in A,\ g \in B,$ and h and g are joinable$\}$.

We use the following further notation where A is a set of mappings with domain X and $h \in A$. For a set $I \subseteq X$, the projection $\pi_I(h)$ is the restriction $h_{|I}$ of h to I; and $\pi_I(A) := \{\pi_I(h) \;:\; h \in A\}$. For objects z, c where $z \notin X$, we write $h \cup \{(z, c)\}$ for the extension h' of h to domain $X \cup \{z\}$ with $h'(z) = c$ and $h'(x) = h(x)$ for all $x \in X$.

Signatures and structures. A *signature* is a finite set σ of relation symbols, where each $R \in \sigma$ is equipped with a fixed *arity* $\operatorname{ar}(R) \in \mathbb{N}_{\geqslant 1}$. A *$\sigma$-structure* $\mathcal{A}$ consists of a finite set A (called the *universe* or *domain* of $\mathcal{A}$) and an $\operatorname{ar}(R)$-ary relation $R^{\mathcal{A}} \subseteq A^{\operatorname{ar}(R)}$ for each $R \in \sigma$. The *size* $\|\sigma\|$ of a signature σ is $|\sigma| + \sum_{R\in\sigma} \operatorname{ar}(R)$. We write $n^{\mathcal{A}}$ to denote the cardinality $|A|$ of $\mathcal{A}$'s universe, we write $m^{\mathcal{A}}$ to denote the number of tuples in $\mathcal{A}$'s largest relation, and we write $N^{\mathcal{A}}$ or $\|\mathcal{A}\|$ to denote the size of a reasonable encoding of $\mathcal{A}$. To be specific, let $N^{\mathcal{A}} = \|\mathcal{A}\| = \|\sigma\| + n^{\mathcal{A}} + \sum_{R\in\sigma} \|R^{\mathcal{A}}\|$, where $\|R^{\mathcal{A}}\| = \operatorname{ar}(R)\cdot|R^{\mathcal{A}}|$. Whenever $\mathcal{A}$ is clear from the context, we will omit the superscript $\cdot^{\mathcal{A}}$ and write n, m, N instead of $n^{\mathcal{A}}, m^{\mathcal{A}}, N^{\mathcal{A}}$. Consider signatures σ and τ with $\sigma \subseteq \tau$. The *σ-reduct* of a τ-structure $\mathcal{B}$ is the σ-structure $\mathcal{A}$ with $A = B$ and $R^{\mathcal{A}} = R^{\mathcal{B}}$ for all $R \in \sigma$. A *τ-expansion* of a σ-structure $\mathcal{A}$ is a τ-structure $\mathcal{B}$ whose σ-reduct is $\mathcal{A}$.

Conjunctive Queries. We fix a countably infinite set $\mathbf{var}$ of *variables.* We allow queries to use arbitrary relation symbols of arbitrary arities. An *atom* α is of the form $R(v_1,\ldots,v_r)$ with $r = \operatorname{ar}(R)$ and $v_1,\ldots,v_r \in \mathbf{var}$. We write $\operatorname{vars}(\alpha)$ to denote the set of variables occurring in α. A *conjunctive query* (CQ, for short) is of the form $\exists z_1 \cdots \exists z_\ell \big(\alpha_1 \wedge \cdots \wedge \alpha_d\big)$, where $\ell \in \mathbb{N}$, $d \in \mathbb{N}_{\geqslant 1}$, α_j is an atom for every $j \in [d]$, and $z_1,\ldots,z_\ell$ are pairwise distinct elements in $\operatorname{vars}(\alpha_1) \cup \cdots \cup \operatorname{vars}(\alpha_d)$. For such a CQ φ we let $\operatorname{atoms}(\varphi) = \{\alpha_1,\ldots,\alpha_d\}$. We write $\operatorname{vars}(\varphi)$ and $\sigma(\varphi)$ for the set of variables and the set of relation symbols occurring in φ, respectively. The set of *quantified* variables of φ is $\operatorname{quant}(\varphi) := \{z_1,\ldots,z_\ell\}$, and the set of *free* variables is $\operatorname{free}(\varphi) := \operatorname{vars}(\varphi) \setminus \operatorname{quant}(\varphi)$. We sometimes write $\varphi(x_1,\ldots,x_k)$ to indicate that $x_1,\ldots,x_k$ are the free variables of φ. The *arity* of φ is the number $k := |\operatorname{free}(\varphi)|$. The query φ is called *quantifier-free* if $\operatorname{quant}(\varphi) = \emptyset$, it is called *Boolean* if its arity is 0, and it is called *self-join-free* if no relation symbol occurs more than once in φ.

The semantics are defined as usual: A *valuation* for φ on a $\sigma(\varphi)$-structure $\mathcal{A}$ is a mapping $\beta : \operatorname{vars}(\varphi) \to A$. A valuation β is a *homomorphism* from φ to a $\mathcal{A}$ if for every atom $R(v_1,\ldots,v_r) \in \operatorname{atoms}(\varphi)$ we have $\big(\beta(v_1),\ldots,\beta(v_r)\big) \in R^{\mathcal{A}}$. The *query result* $[\![\varphi]\!]^{\mathcal{A}}$ of a CQ φ on the $\sigma(\varphi)$-structure $\mathcal{A}$ is defined as the set $\{\,\pi_{\operatorname{free}(\varphi)}(\beta) : \beta$ is a homomorphism from φ to $\mathcal{A}\}$. Often, we will identify the mappings $g \in [\![\varphi]\!]^{\mathcal{A}}$ with tuples $(g(x_1),\ldots,g(x_k))$, where $x_1,\ldots,x_k$ is a fixed listing of the free variables of φ.

The size $\|\varphi\|$ of a query φ is the length of φ when viewed as a word over the alphabet $\sigma(\varphi) \cup \operatorname{vars}(\varphi) \cup \{\exists, \wedge, (\,,)\,\} \cup \{\,,\}$.

Model of computation. For the complexity analysis we assume the RAM-model with a uniform cost measure. In particular, storing and accessing elements from a structure's universe requires $O(1)$ space and time. For an r-ary relation $R^{\mathcal{A}}$ we can construct in time $O(\|R^{\mathcal{A}}\|)$ an index that allows to enumerate $R^{\mathcal{A}}$ with $O(1)$ delay and to test for a given r-tuple a whether $a \in R^{\mathcal{A}}$ in time $O(r)$. Moreover, for every $\{i_1,\ldots,i_\ell\} \subseteq [r]$ we can build a data structure where we can enumerate for every ℓ-tuple b the selection $\{a \in R^{\mathcal{A}} \ : \ \pi_{i_1,\ldots,i_\ell}(a) = b\}$ with $O(1)$ delay. Such a data structure can be constructed in time $O(\|R^{\mathcal{A}}\|)$, for instance by a linear scan over $R^{\mathcal{A}}$ where we add every tuple $a \in R^{\mathcal{A}}$ to a list $\mathcal{L}_{\pi_{i_1,\ldots,i_\ell}(a)}$. Using a constant access data structure of linear size, the list $\mathcal{L}_b$ can be accessed in time $O(\ell)$ when receiving an ℓ-tuple b.

Constant delay enumeration and testing. An *enumeration algorithm* for query evaluation consists of two phases: the preprocessing phase and the enumeration phase. In the preprocessing phase the algorithm is allowed to do arbitrary preprocessing on the query φ and the input structure $\mathcal{A}$. We denote the time required for this phase by t_p. In the subsequent enumeration phase the algorithm enumerates, without repetition, all tuples (or, mappings) in the query result $[\![\varphi]\!]^{\mathcal{A}}$, followed by the end-of-enumeration message `EOE`. The *delay* t_d is the maximum time that passes between the start of the enumeration phase and the output of the first tuple, between the output of two consecutive tuples, and between the last tuple and `EOE`.

A *testing algorithm* for query evaluation also starts with a preprocessing phase of time t_p in which a data structure is computed that allows to test for a given tuple (or, mapping) b whether it is contained in the query result $[\![\varphi]\!]^{\mathcal{A}}$. The *testing time* t_t of the algorithm is an upper bound on the time that passes between receiving b and providing the answer.

One speaks of *constant* delay (testing time) if the delay (testing time) depends on the query φ, but not on the input structure $\mathcal{A}$.

We make use of the following result from Durand and Strozecki, which allows to efficiently enumerate the union of query results, provided that each query result in the union can be enumerated and tested efficiently. Note that this is not immediate, because the union might contain many duplicates that need to be avoided during enumeration.

▶ **Theorem 2.1** ([21]). *Suppose that there is an enumeration algorithm $\mathbb{A}$ that receives a query φ and a structure $\mathcal{A}$ and enumerates $[\![\varphi]\!]^{\mathcal{A}}$ with delay $t_d(\varphi)$ after $t_p(\varphi,\mathcal{A})$ preprocessing time. Further suppose that there is a testing algorithm $\mathbb{B}$ that receives a query φ and a structure $\mathcal{A}$ and has $t_p(\varphi,\mathcal{A})$ preprocessing time and $t_t(\varphi)$ testing time. Then there is an algorithm $\mathbb{C}$ that receives ℓ queries φ_i and structures $\mathcal{A}_i$ and allows to enumerate $\bigcup_{i\in[\ell]}[\![\varphi_i]\!]^{\mathcal{A}_i}$ with $O(\sum_{i\in[\ell]} t_d(\varphi_i)+\sum_{i\in[\ell]} t_t(\varphi_i))$ delay after $O(\sum_{i\in[\ell]} t_p(\varphi_i,\mathcal{A}_i))$ preprocessing time.*

Proof (sketch). The induction start $\ell=1$ is trivial. For the induction step $\ell\to\ell+1$ start an enumeration of $\bigcup_{i\in[\ell]}[\![\varphi_i]\!]^{\mathcal{A}_i}$ and test for every tuple whether it is contained in $[\![\varphi_{\ell+1}]\!]^{\mathcal{A}_{\ell+1}}$. If the answer is no, then output the tuple. Otherwise discard the tuple and instead output the next tuple in an enumeration of $[\![\varphi_{\ell+1}]\!]^{\mathcal{A}_{\ell+1}}$. Subsequently enumerate the remaining tuples from $[\![\varphi_{\ell+1}]\!]^{\mathcal{A}_{\ell+1}}$. ◀

3 Main Result

At the end of this section, we provide a precise statement of our main result. Before we can do so, we have to recall the concept of free-connex decompositions of queries and the notion of submodular width. It will be convenient for us to use the following notation.

▶ **Definition 3.1.** *Let $\varphi=\exists z_1\cdots\exists z_\ell\,\big(\alpha_1\wedge\cdots\wedge\alpha_d\big)$ be a CQ and $S\subseteq\text{vars}(\varphi)$. We write $\varphi\langle S\rangle$ for the CQ that is equivalent to the expression*

$$\big(\,\exists y_1\cdots\exists y_r\ \alpha_1\,\big)\wedge\cdots\wedge\ \big(\,\exists y_1\cdots\exists y_r\ \alpha_d\,\big),\tag{1}$$

where $\{y_1,\ldots,y_r\}=\text{vars}(\varphi)\setminus S$.

Note that $\varphi\langle S\rangle$ is obtained from φ by discarding existential quantification and projecting every atom to S, hence $\text{free}(\varphi\langle S\rangle)=S$. However, $[\![\varphi\langle S\rangle]\!]^{\mathcal{A}}$ shall not be confused with the projection of $[\![\varphi]\!]^{\mathcal{A}}$ to S. In fact, it might be that $[\![\varphi]\!]^{\mathcal{A}}$ is empty, but $[\![\varphi\langle S\rangle]\!]^{\mathcal{A}}$ is not, as the following example illustrates:

$$\begin{aligned}\varphi &= E(x,y)\wedge E(y,z)\wedge E(x,z)\quad\text{and} &(2)\\ \varphi\langle\{x,z\}\rangle &\equiv \exists yE(x,y)\wedge\exists yE(y,z)\wedge\exists yE(x,z) &(3)\\ &\equiv E(x,z)\ . &(4)\end{aligned}$$

3.1 Constant delay enumeration using tree decompositions

We use the same notation as [22] for decompositions of queries: A *tree decomposition* (TD, for short) of a CQ φ is a tuple $TD = (T, \chi)$, for which the following two conditions are satisfied:

1. $T = (V(T), E(T))$ is a finite undirected tree.
2. χ is a mapping that associates with every node $t \in V(T)$ a set $\chi(t) \subseteq \text{vars}(\varphi)$ such that
   a. for each atom $\alpha \in \text{atoms}(\varphi)$ there exists $t \in V(T)$ such that $\text{vars}(\alpha) \subseteq \chi(t)$, and
   b. for each variable $v \in \text{vars}(\varphi)$ the set $\chi^{-1}(v) := \{t \in V(T) \ : \ v \in \chi(t)\}$ induces a connected subtree of T (this condition is called *path condition*).

To use a tree decomposition $TD = (T, \chi)$ of φ for query evaluation one considers, for each $t \in V(T)$ the query $\varphi\langle S\rangle$ for $S := \chi(t)$, evaluates this query on the input structure $\mathcal{A}$, and then combines these results for all $t \in V(T)$ along a bottom-up traversal of T. If the query is Boolean, this yields the result of φ on $\mathcal{A}$; if it is non-Boolean, $[\![\varphi]\!]^{\mathcal{A}}$ can be computed by performing additional traversals of T. This approach is efficient if the result sets $[\![\varphi\langle\chi(t)\rangle]\!]^{\mathcal{A}}$ are small and can be computed efficiently (later on, we will sometimes refer to the sets $[\![\varphi\langle\chi(t)\rangle]\!]^{\mathcal{A}}$ as *projections on bags*).

The simplest queries where this is the case are acyclic queries [11, 15]. A number of equivalent characterisations of the acyclic CQs have been provided in the literature (cf. [1, 23, 25, 17]); among them a characterisation by Gottlob et al. [23] stating that a CQ is acyclic if and only if it has a tree-decomposition where every bag is covered by an atom, i.e., for every bag $\chi(t)$ there is some atom α in φ with $\chi(t) \subseteq \text{vars}(\alpha)$. The approach described above leads to a linear time algorithm for evaluating an acyclic CQ φ that is Boolean, and if φ is non-Boolean, $[\![\varphi]\!]^{\mathcal{A}}$ is computed in time linear in $\|\mathcal{A}\| + |\,[\![\varphi]\!]^{\mathcal{A}}|$. This method is known as *Yannakakis' algorithm.* But this algorithm does not distinguish between a preprocessing phase and an enumeration phase and does not guarantee constant delay enumeration. In fact, Bagan et al. identified the following additional property that is needed to ensure constant delay enumeration.

▶ **Definition 3.2** ([10]). *A tree decomposition $TD = (T, \chi)$ of a CQ φ is* free-connex *if there is a subset $U \subseteq V(T)$ that induces a connected subtree of T and that satisfies the condition* $\text{free}(\varphi) = \bigcup_{t \in U} \chi(t)$.

Bagan et al. [10] identified the *free-connex* acyclic CQs, i.e., the CQs φ that have a free-connex tree decomposition where every bag is covered by an atom, as the fragment of the acyclic CQs whose results can be enumerated with constant delay after FPL-preprocessing:

▶ **Theorem 3.3** (Bagan et al. [10]). *There is a computable function f and an algorithm which receives a free-connex acyclic CQ φ and a $\sigma(\varphi)$-structure $\mathcal{A}$ and computes within $t_p = f(\varphi)O(\|\mathcal{A}\|)$ preprocessing time and space a data structure that allows to*
(i) *enumerate $[\![\varphi]\!]^{\mathcal{A}}$ with $f(\varphi)$ delay and*
(ii) *test for a given tuple (or, mapping) b if $b \in [\![\varphi]\!]^{\mathcal{A}}$ within $f(\varphi)$ testing time.*

The approach of using free-connex tree decompositions for constant delay enumeration can be extended from acyclic CQs to arbitrary CQs. To do this, we have to compute for every bag $\chi(t)$ in the tree decomposition the projection $[\![\varphi\langle\chi(t)\rangle]\!]^{\mathcal{A}}$. This reduces the task to the acyclic case, where the free-connex acyclic query contains one atom α with $\text{vars}(\alpha) = \chi(t)$ for every bag $\chi(t)$ and the corresponding relation is defined by $[\![\varphi\langle\chi(t)\rangle]\!]^{\mathcal{A}}$. Because the runtime in this approach is dominated by computing $[\![\varphi\langle\chi(t)\rangle]\!]^{\mathcal{A}}$, it is only feasible if the projections are efficiently computable for every bag. If the decomposition has bounded treewidth or bounded fractional hypertree width, then it is possible to compute $[\![\varphi\langle\chi(t)\rangle]\!]^{\mathcal{A}}$ for every bag in time $f(\varphi)\cdot\|\mathcal{A}\|^{O(1)}$ [24], which in turn implies that the result can be enumerated after FPT-preprocessing time for CQs of bounded fc-tw [10] and for CQs of bounded fc-fhw [37].

3.2 Submodular width and statement of the main result

Before providing the precise definition of the submodular width of a query, let us first consider an example. The central idea behind algorithms that rely on submodular width [33, 3, 38] is to split the input structure into several parts and use for every part a different tree decomposition of φ. This will give a significant improvement over the fractional hypertree width, which uses only one tree decomposition of φ. A typical example to illustrate this idea is the following 4-cycle query (see also [3, 38]): $\varphi_4 := E_{12}(x_1,x_2) \wedge E_{23}(x_2,x_3) \wedge E_{34}(x_3,x_4) \wedge E_{41}(x_4,x_1)$.

There are essentially two non-trivial tree decompositions $\mathit{TD}' = (T,\chi')$, $\mathit{TD}'' = (T,\chi'')$ of φ_4, which are both defined over the two-vertex tree $T = (\{t_1,t_2\},\{(t_1,t_2)\})$ by $\chi'(t_1) = \{x_1,x_2,x_3\}$, $\chi'(t_2) = \{x_1,x_3,x_4\}$ and $\chi''(t_1) = \{x_2,x_3,x_4\}$, $\chi''(t_2) = \{x_1,x_2,x_4\}$. Both tree decompositions lead to an optimal fractional hypertree decomposition of width $\mathit{fhw}(\varphi_4) = 2$. Indeed, for the worst-case instance $\mathcal{A}$ with

$$E_{12}^{\mathcal{A}} = E_{34}^{\mathcal{A}} := [\ell]\times\{a\} \ \cup\ \{b\}\times[\ell] \qquad\qquad E_{23}^{\mathcal{A}} = E_{41}^{\mathcal{A}} := [\ell]\times\{b\} \ \cup\ \{a\}\times[\ell]$$

we have $\|\mathcal{A}\| = O(\ell)$ while the projections on the bags have size $\Omega(\ell^2)$ in both decompositions:[2]

$$\begin{aligned}
[\![\varphi_4\langle\chi'(t_1)\rangle]\!]^{\mathcal{A}} &= [\![\varphi_4\langle\chi'(t_2)\rangle]\!]^{\mathcal{A}} = [\ell]\times\{a\}\times[\ell] \ \cup\ \{b\}\times[\ell]\times\{b\},\\
[\![\varphi_4\langle\chi''(t_1)\rangle]\!]^{\mathcal{A}} &= [\![\varphi_4\langle\chi''(t_2)\rangle]\!]^{\mathcal{A}} = [\ell]\times\{b\}\times[\ell] \ \cup\ \{a\}\times[\ell]\times\{a\}.
\end{aligned}$$

However, we can split $\mathcal{A}$ into $\mathcal{A}'$ and $\mathcal{A}''$ such that $[\![\varphi_4]\!]^{\mathcal{A}}$ is the disjoint union of $[\![\varphi_4]\!]^{\mathcal{A}'}$ and $[\![\varphi_4]\!]^{\mathcal{A}''}$ and the bag-sizes in the respective decompositions are small:

$$\begin{aligned}
E_{12}^{\mathcal{A}'} &= E_{34}^{\mathcal{A}'} := \{b\}\times[\ell] & E_{23}^{\mathcal{A}'} &= E_{41}^{\mathcal{A}'} := [\ell]\times\{b\}\\
E_{12}^{\mathcal{A}''} &= E_{34}^{\mathcal{A}''} := [\ell]\times\{a\} & E_{23}^{\mathcal{A}''} &= E_{41}^{\mathcal{A}''} := \{a\}\times[\ell]\\
[\![\varphi_4\langle\chi'(t_1)\rangle]\!]^{\mathcal{A}'} &= [\![\varphi_4\langle\chi'(t_2)\rangle]\!]^{\mathcal{A}'} = \{b\}\times[\ell]\times\{b\},\\
[\![\varphi_4\langle\chi''(t_1)\rangle]\!]^{\mathcal{A}''} &= [\![\varphi_4\langle\chi''(t_2)\rangle]\!]^{\mathcal{A}''} = \{a\}\times[\ell]\times\{a\}.
\end{aligned}$$

Thus, we can efficiently evaluate φ_4 on $\mathcal{A}'$ using TD' and φ_4 on $\mathcal{A}''$ using TD'' (in time $O(\ell)$ in this example) and then combine both results to obtain $\varphi_4(\mathcal{A})$. Using the strategy of Alon et al. [4], it is possible to split *every* database $\mathcal{A}$ for this particular 4-cycle query φ_4 into two instances $\mathcal{A}'$ and $\mathcal{A}''$ such that the bag sizes in TD' on $\mathcal{A}'$ as well as in TD'' on $\mathcal{A}''$ are bounded by $\|\mathcal{A}\|^{3/2}$ and can be computed in time $O(\|\mathcal{A}\|^{3/2})$ (see [3, 38] for a detailed account on this strategy). As both decompositions are free-connex, this also leads to a constant delay enumeration algorithm for φ_4 with $O(\|\mathcal{A}\|^{3/2})$ time preprocessing, which improves the $O(\|\mathcal{A}\|^2)$ preprocessing time that follows from using one decomposition.

In general, whether such a data-dependent decomposition is possible is determined by the submodular width $\mathit{subw}(\varphi)$ of the query. The notion of submodular width was introduced in [33]. To present its definition, we need the following terminology. A function $g\colon 2^{\mathrm{vars}(\varphi)} \to \mathbb{R}_{\geqslant 0}$ is

- *monotone* if $g(U) \leqslant g(V)$ for all $U \subseteq V \subseteq \mathrm{vars}(\varphi)$.

[2] recall from Section 2 our convention to identify mappings in query results with tuples; the free variables are listed canonically here, by increasing indices

- *edge-dominated* if $g(\mathrm{vars}(\alpha)) \leqslant 1$ for every atom $\alpha \in \mathrm{atoms}(\varphi)$.
- *submodular*, if $g(U) + g(V) \geqslant g(U \cap V) + g(U \cup V)$ for every $U, V \subseteq \mathrm{vars}(\varphi)$.

We denote by $\mathsf{S}(\varphi)$ the set of all monotone, edge-dominated, submodular functions $g\colon 2^{\mathrm{vars}(\varphi)} \to \mathbb{R}_{\geqslant 0}$ that satisfy $g(\emptyset) = 0$, and by $\mathsf{T}(\varphi)$ the set of all tree decompositions of φ. The *submodular width* of a conjunctive query φ is

$$subw(\varphi) \quad := \quad \sup_{g \in \mathsf{S}(\varphi)} \quad \min_{(T,\chi) \in \mathsf{T}(\varphi)} \quad \max_{t \in V(T)} \; g(\chi(t)). \tag{5}$$

In particular, if the submodular width of φ is bounded by w, then for every submodular function g there is a tree decomposition in which every bag B satisfies $g(B) \leqslant w$.

It is known that $subw(\varphi) \leqslant fhw(\varphi)$ for all queries φ [33, Proposition 3.7]. Moreover, there is a constant c and a family of queries φ such that $subw(\varphi) \leqslant c$ is bounded and $fhw(\varphi) = \Omega(\sqrt{\log \|\varphi\|})$ is unbounded [32, 33]. The main result in [33] is that the submodular width characterises the tractability of Boolean CQs in the following sense.

▶ **Theorem 3.4** ([33]).

(1) *There is a computable function f and an algorithm that receives a Boolean CQ φ, $subw(\varphi)$, and a $\sigma(\varphi)$-structure $\mathcal{A}$ and evaluates φ on $\mathcal{A}$ in time $f(\varphi)\|\mathcal{A}\|^{O(subw(\varphi))}$.*

(2) *Let Φ be a recursively enumerable class of Boolean, self-join-free CQs of unbounded submodular width. Assuming the exponential time hypothesis (ETH) there is no algorithm which, upon input of a query $\varphi \in \Phi$ and a structure $\mathcal{A}$, evaluates φ on $\mathcal{A}$ in time $\|\mathcal{A}\|^{o(subw(\varphi)^{1/4})}$.*

The *free-connex submodular width* of a conjunctive query φ is defined in a similar way as submodular width, but this time ranges over the set $\mathsf{fcT}(\varphi)$ of all free-connex tree decompositions of φ (it is easy to see that we can assume that $\mathsf{fcT}(\varphi)$ is finite).

$$fc\text{-}subw(\varphi) := \sup_{g \in \mathsf{S}(\varphi)} \quad \min_{(T,\chi) \in \mathsf{fcT}(\varphi)} \quad \max_{t \in V(T)} \; g(\chi(t)). \tag{6}$$

Note that if φ is either quantifier-free or Boolean, we have $fc\text{-}subw(\varphi) = subw(\varphi)$. In general, this is not always the case. Consider for example the following quantified version $\varphi_4' := \exists x_1 \exists x_3 \, \varphi_4$ of the quantifier-free 4-cycle query φ_4. Here we have $subw(\varphi_4') = \frac{3}{2}$, but $fc\text{-}subw(\varphi_4') = 2$: one can verify $fc\text{-}subw(\varphi_4') \geqslant 2$ by noting that every free-connex tree decomposition contains a bag $\{x_1, x_2, x_3, x_4\}$ and taking the submodular function $g(U) := \frac{1}{2}|U|$. Now we are ready to state the main theorem of this paper.

▶ **Theorem 3.5.** *For every $\delta > 0$ and $w \geqslant 1$ there is a computable function f and an algorithm which receives a CQ φ with $fc\text{-}subw(\varphi) \leqslant w$ and a $\sigma(\varphi)$-structure $\mathcal{A}$ and computes within $t_p = f(\varphi)\|\mathcal{A}\|^{(2+\delta)w}$ preprocessing time and space $f(\varphi)\|\mathcal{A}\|^{(1+\delta)w}$ a data structure that allows to*

(i) *enumerate $[\![\varphi]\!]^{\mathcal{A}}$ with $f(\varphi)$ delay and*

(ii) *test for a given tuple (or, mapping) b if $b \in [\![\varphi]\!]^{\mathcal{A}}$ within $f(\varphi)$ testing time.*

The following corollary is an immediate consequence of Theorem 3.5 and Theorem 3.4. A class Φ of CQs is said to be of *bounded free-connex submodular width* if there exists a number w such that $fc\text{-}subw(\varphi) \leqslant w$ for all $\varphi \in \Phi$. And by an *algorithm for Φ that enumerates with constant delay after FPT-preprocessing* we mean an algorithm that receives a query $\varphi \in \Phi$ and a $\sigma(\varphi)$-structure $\mathcal{A}$ and spends $f(\varphi)\|\mathcal{A}\|^{O(1)}$ preprocessing time and then enumerates $[\![\varphi]\!]^{\mathcal{A}}$ with delay $f(\varphi)$, for a computable function f.

▶ **Corollary 3.6.**

(1) *For every class Φ of CQs of bounded free-connex submodular width, there is an algorithm for Φ that enumerates with constant delay after FPT-preprocessing.*

(2) *Let Φ be a recursively enumerable class of quantifier-free self-join-free CQs and assume that the exponential time hypothesis (ETH) holds.*
Then there is an algorithm for Φ that enumerates with constant delay after FPT-preprocessing if, and only if, Φ has bounded free-connex submodular width.

4 Proof of the Main Result

To prove Theorem 3.5, we make use of Marx's splitting routine for queries of bounded submodular width. In the following, we will adapt the main definitions and concepts from [33] to our notions. While doing this, we provide the following additional technical contributions: First, we give a detailed time and space analysis of the algorithm and improve the runtime of the consistency algorithm [33, Lemma 4.5] from quadratic to linear (see Lemma 4.2). Second, we fix an oversight in [33, Lemma 4.12] by establishing strong M-consistency (unfortunately, this fix incurs a blow-up in running time). Afterwards we prove our main theorem, where the non-Boolean setting requires us to relax Marx's partition into *refinements* (Lemma 4.5) so that the subinstances are no longer disjoint.

Let φ be a quantifier-free CQ with $\text{vars}(\varphi) = \{x_1, \dots, x_k\}$, and let $\sigma := \sigma(\varphi)$. For every $S = \{x_{i_1}, \dots, x_{i_\ell}\} \subseteq \text{vars}(\varphi)$ where $i_1 < \dots < i_\ell$ we set $x_S := (x_{i_1}, \dots, x_{i_\ell})$ and let $R_S \notin \sigma$ be a fresh ℓ-ary relation symbol. For every collection $\mathfrak{s} \subseteq 2^{\text{vars}(\varphi)}$ we let

$$\sigma_{\mathfrak{s}} := \sigma \cup \{R_S \ : \ S \in \mathfrak{s}\} \quad \text{and} \tag{7}$$

$$\varphi_{\mathfrak{s}} := \varphi \wedge \textstyle\bigwedge_{S \in \mathfrak{s}} R_S(x_S). \tag{8}$$

A *refinement* of φ and a σ-structure $\mathcal{A}$ is a pair $(\mathfrak{s}, \mathcal{B})$, where $\mathfrak{s} \subseteq 2^{\text{vars}(\varphi)}$ is closed under taking subsets and $\mathcal{B}$ is a $\sigma_{\mathfrak{s}}$-expansion of $\mathcal{A}$. Note that if $(\mathfrak{s}, \mathcal{B})$ is a refinement of φ and $\mathcal{A}$, then $[\![\varphi_{\mathfrak{s}}]\!]^{\mathcal{B}} \subseteq [\![\varphi]\!]^{\mathcal{A}}$. In the following we will construct refinements that do not change the result relation, i. e., $[\![\varphi_{\mathfrak{s}}]\!]^{\mathcal{B}} = [\![\varphi]\!]^{\mathcal{A}}$. Subsequently, we will split refinements in order to partition the query result.

The following definition collects useful properties of refinements. Recall from Section 2 that for a CQ ψ and a structure $\mathcal{B}$, the query result $[\![\psi]\!]^{\mathcal{B}}$ actually is a set of mappings from $\text{free}(\psi)$ to B. For notational convenience we define $\mathbf{R}_S^{\mathcal{B}} := [\![R_S(x_S)]\!]^{\mathcal{B}}$ and use the set $\mathbf{R}_S^{\mathcal{B}}$ of mappings instead of the relation $R_S^{\mathcal{B}}$. In particular, by addressing/inserting/deleting a mapping $h \colon S \to B$ from $\mathbf{R}_S^{\mathcal{B}}$ we mean addressing/inserting/deleting the tuple $(h(x_{i_1}), \dots, h(x_{i_\ell}))$ from $R_S^{\mathcal{B}}$, where $(x_{i_1}, \dots, x_{i_\ell}) = x_S$.

▶ **Definition 4.1.** *Let φ be a quantifier-free σ-CQ, $\mathcal{A}$ a σ-structure, $(\mathfrak{s}, \mathcal{B})$ a refinement of φ and $\mathcal{A}$, and M an integer.*

1. *The refinement $(\mathfrak{s}, \mathcal{B})$ is* consistent *if*

$$\mathbf{R}_S^{\mathcal{B}} = [\![\varphi_{\mathfrak{s}}\langle S\rangle]\!]^{\mathcal{B}} \text{ for all } S \in \mathfrak{s} \text{ and} \tag{9}$$

$$\mathbf{R}_S^{\mathcal{B}} = \pi_S\big(\mathbf{R}_T^{\mathcal{B}}\big) \text{ for all } S, T \in \mathfrak{s} \text{ with } S \subset T. \tag{10}$$

2. *The refinement $(\mathfrak{s}, \mathcal{B})$ is M-*consistent *if it is consistent and*

$$S \in \mathfrak{s} \quad \Longleftrightarrow \quad \text{for all } T \subseteq S\text{: } |\,[\![\varphi_{\mathfrak{s}}\langle T\rangle]\!]^{\mathcal{B}}| \leqslant M. \tag{11}$$

3. *The refinement $(\mathfrak{s}, \mathcal{B})$ is* strongly M-consistent *if it is M-consistent and*

$$S \in \mathfrak{s},\ T \in \mathfrak{s},\ (S \cup T) \notin \mathfrak{s} \quad \Longrightarrow \quad |\,[\![\varphi_{\mathfrak{s}}\langle S \cup T\rangle]\!]^{\mathcal{B}}| > M. \tag{12}$$

The proof of the following lemma is deferred to the full version of the paper.

▶ **Lemma 4.2.** *There is an algorithm that receives a refinement $\mathcal{R} = (\mathfrak{s}, \mathcal{B})$ of φ and $\mathcal{A}$ and computes in time $O(|\mathfrak{s}| \cdot \|\mathcal{B}\|)$ a consistent refinement $(\mathfrak{s}, \mathcal{B}')$ with $R_S^{\mathcal{B}'} \subseteq R_S^{\mathcal{B}}$ for all $S \in \mathfrak{s}$ and $[\![\varphi_{\mathfrak{s}}]\!]^{\mathcal{B}'} = [\![\varphi_{\mathfrak{s}}]\!]^{\mathcal{B}}$.*

▶ **Lemma 4.3.** *Let φ be a quantifier-free CQ, let $\mathcal{A}$ be a $\sigma(\varphi)$-structure where the largest relation contains m tuples, and let $M \geqslant m$. There is an algorithm that computes in time $O(2^{|\mathrm{vars}(\varphi)|} \cdot M^2)$ and space $O(2^{|\mathrm{vars}(\varphi)|} \cdot M)$ a strongly M-consistent refinement $(\mathfrak{s}, \mathcal{B})$ that satisfies $[\![\varphi]\!]^{\mathcal{A}} = [\![\varphi_{\mathfrak{s}}]\!]^{\mathcal{B}}$.*

Algorithm 1 Computing a strongly M-consistent refinement.

```
INPUT: quantifier-free CQ $\varphi(x_1, \dots, x_k)$, $\sigma(\varphi)$-structure $\mathcal{A}$
$\mathfrak{s} \leftarrow \emptyset$ ; $\mathcal{B} \leftarrow \mathcal{A}$
repeat
    for $\ell = 1, \cdots, k$ do                                  ▷ Step 1: Ensure consition (11).
        for $S = \{x_{i_1}, \dots, x_{i_\ell}\} \subseteq \mathrm{vars}(\varphi)$ do
            if $S \notin \mathfrak{s}$ and $S \setminus \{x\} \in \mathfrak{s}$ for all $x \in S$ then
                $\mathbf{R}_S^{\mathcal{B}} \leftarrow \emptyset$
                Choose $x \in S$ arbitrary
                for $h \in \mathbf{R}_{S\setminus\{x\}}^{\mathcal{B}}$ and $c \in A$ do
                    if $h \cup \{(x,c)\} \in [\![\varphi_{\mathfrak{s}}\langle S \rangle]\!]^{\mathcal{B}}$ then $\mathbf{R}_S^{\mathcal{B}} \leftarrow \mathbf{R}_S^{\mathcal{B}} \cup \{h \cup \{(x,c)\}\}$
                if $|\mathbf{R}_S^{\mathcal{B}}| \leqslant M$ then $\mathfrak{s} \leftarrow \mathfrak{s} \cup \{S\}$

    for $S, T \in \mathfrak{s}$ such that $S \cup T \notin \mathfrak{s}$ do              ▷ Step 2: Ensure condition (12).
        for $g \in \mathbf{R}_S^{\mathcal{B}}$ and $h \in \mathbf{R}_T^{\mathcal{B}}$ do
            if $g \bowtie h \in [\![\varphi_{\mathfrak{s}}\langle S \cup T \rangle]\!]^{\mathcal{B}}$ then $\mathbf{R}_{S\cup T}^{\mathcal{B}} \leftarrow \mathbf{R}_{S\cup T}^{\mathcal{B}} \cup \{g \bowtie h\}$
            if $|\mathbf{R}_{S\cup T}^{\mathcal{B}}| > M$ then break
        if $|\mathbf{R}_{S\cup T}^{\mathcal{B}}| \leqslant M$ then $\mathfrak{s} \leftarrow \mathfrak{s} \cup \{S \cup T\}$

    $(\mathfrak{s}, \mathcal{B}) \leftarrow$ CONSISTENT$(\mathfrak{s}, \mathcal{B})$          ▷ Step 3: Apply Lemma 4.2 to ensure (9), (10).
until $\mathfrak{s}$ remains unchanged
return $(\mathfrak{s}, \mathcal{B})$
```

Proof. The pseudocode of the algorithm is shown in Algorithm 1. For computing the strongly M-consistent refinement we first compute all sets S where for all $T \subseteq S$ we have $|\,[\![\varphi_{\mathfrak{s}}\langle T \rangle]\!]^{\mathcal{B}}| \leqslant M$; as in [33], we say that such sets S are *M-small.* First note that the empty set is M-small. For nonempty sets S we know that S is only M-small if for every $x \in S$ the set $S \setminus \{x\}$ is M-small and hence already included in $\mathfrak{s}$. If this is the case, then $[\![\varphi_{\mathfrak{s}}\langle S \rangle]\!]^{\mathcal{B}}$ can be computed in time $O(M \cdot n)$ by testing for every $h \in \mathbf{R}_{S\setminus\{x\}}^{\mathcal{B}}$ (for an arbitrary $x \in S$) and every element c in the structure's universe, whether $h \cup \{(x,c)\} \in [\![\varphi_{\mathfrak{s}}\langle S \rangle]\!]^{\mathcal{B}}$. If $|[\![\varphi_{\mathfrak{s}}\langle S \rangle]\!]^{\mathcal{B}}| \leqslant M$, then we include S and $\mathbf{R}_S^{\mathcal{B}} := [\![\varphi_{\mathfrak{s}}\langle S \rangle]\!]^{\mathcal{B}}$ into our current refinement. Afterwards, we want to satisfy the condition on strong M-consistency (12) by trying all pairs of M-small sets S and T. This is the bottleneck of our algorithm and requires time $O(|\mathbf{R}_S^{\mathcal{B}}| \cdot |\mathbf{R}_T^{\mathcal{B}}|) \leqslant O(M^2)$. In the third step we apply Lemma 4.2 to enforce consistency of the current refinement. In particular, every set $S \cup T$ that was found in step 2 becomes M-small. Note that after deleting tuples to ensure consistency, new sets may become M-small. Therefore, we have to repeat steps 1–3 until no more sets became M-small. Overall, we repeat the outer loop at most 2^k times, step 1 takes time $2^{O(k)} \cdot M \cdot n$, step 2 takes time $2^{O(k)} \cdot M^2$ and step 3 takes time $2^{O(k)} \cdot M$. Since $n \leqslant M$ this leads to the required running time. ◀

The key step in the proof of Theorem 3.5 is to compute $f(\varphi)$ strongly M-consistent refinements $(\mathfrak{s}_i, \mathcal{B}_i)$ of φ and $\mathcal{A}$ such that $[\![\varphi]\!]^{\mathcal{A}} = \bigcup_i [\![\varphi_{\mathfrak{s}_i}]\!]^{\mathcal{B}_i}$. In addition to being strongly M-consistent, we want the structures $\mathcal{B}_i$ to be uniform in the sense that the degree of tuples (i. e. the number of extensions) is roughly the average degree. We make this precise in a moment, but for illustration it might be helpful to consult the example from Section 3.2 again. In every relation in $\mathcal{A}$ there is one vertex (a or b) of out-degree ℓ and there are ℓ vertices of out-degree 1. Hence the average out-degree is $2\ell/(\ell+1)$ and the vertex degrees are highly imbalanced. However, after splitting the instance in $\mathcal{A}'$ and $\mathcal{A}''$, in every relation, all vertices have either out-degree ℓ or 1 and the out-degree of every vertex matches the average out-degree of the corresponding relation. The next definition generalises this to tuples of variables. We call a refinement $(\mathfrak{s}, \mathcal{B})$ *non-trivial*, if every additional relation in the expansion $\mathcal{B}$ is non-empty. For a non-trivial consistent refinement $(\mathfrak{s}, \mathcal{B})$ and $S, T \in \mathfrak{s}$, $S \subseteq T$, we let

$$\operatorname{avgdeg}(S,T) := |\mathbf{R}_T^{\mathcal{B}}|/|\mathbf{R}_S^{\mathcal{B}}| \qquad \text{and} \tag{13}$$

$$\operatorname{maxdeg}(S,T) := \max_{g \in \mathbf{R}_S^{\mathcal{B}}} \{h \in \mathbf{R}_T^{\mathcal{B}} \ : \ \pi_S(h) = g\}. \tag{14}$$

Note that consistency ensures that these numbers are well-defined and non-zero. Furthermore, we can compute them from $(\mathfrak{s}, \mathcal{B})$ in time $O(|\mathfrak{s}|^2 \cdot \|\mathcal{B}\|)$. By definition we have $\operatorname{maxdeg}(S,T) \geqslant \operatorname{avgdeg}(S,T)$. The next definition states that maximum degree does not deviate too much from the average degree.

▶ **Definition 4.4.** *Let $(\mathfrak{s}, \mathcal{B})$ be a non-trivial consistent refinement of φ and $\mathcal{A}$, and let m be the number of tuples of largest relation of $\mathcal{A}$. The refinement $(\mathfrak{s}, \mathcal{B})$ is* ε-uniform *if for all $S, T \in \mathfrak{s}$ with $S \subseteq T$ we have* $\operatorname{maxdeg}(S,T) \leqslant m^{\varepsilon} \cdot \operatorname{avgdeg}(S,T)$.

The next lemma uses Marx's splitting routine to obtain a partition into strongly M-consistent ε-uniform refinements, for $M := m^c$.

▶ **Lemma 4.5.** *Let φ be a quantifier-free CQ, let $\mathcal{A}$ be a $\sigma(\varphi)$-structure where the largest relation contains m tuples, and let $c \geqslant 1$ and $\epsilon > 0$ be real numbers. There is a computable function f and an algorithm that computes in time $O(f(\varphi, c, \epsilon) \cdot m^{2c})$ and space $O(f(\varphi, c, \epsilon) \cdot m^c)$ a sequence of $\ell \leqslant f(\varphi, c, \epsilon)$ strongly m^c-consistent ε-uniform refinements $(\mathfrak{s}_i, \mathcal{B}_i)$ such that $[\![\varphi]\!]^{\mathcal{A}}$ is the disjoint union of the sets $[\![\varphi_{\mathfrak{s}_i}]\!]^{\mathcal{B}_i}$*

Proof (sketch). We follow the same splitting strategy as in [33], but use the improved algorithm from Lemma 4.3 to ensure strong m^c-consistency. Starting with the trivial refinement $(\emptyset, \mathcal{A})$, in each step we first apply Lemma 4.3 to ensure strong m^c-consistency. Afterwards, we check whether the current refinement $(\mathfrak{s}, \mathcal{B})$ contains sets $S, T \in \mathfrak{s}$ that contradict ϵ-uniformity, i. e., $S \subseteq T$ and $\operatorname{maxdeg}(S,T) > m^{\varepsilon} \cdot \operatorname{avgdeg}(S,T)$. If this is the case, we split the refinement $(\mathfrak{s}, \mathcal{B})$ into $(\mathfrak{s}, \mathcal{B}')$ and $(\mathfrak{s}, \mathcal{B}'')$ such that $\mathbf{R}_S^{\mathcal{B}}$ is partitioned into tuples of small degree and tuples of large degree:

$$\mathbf{R}_U^{\mathcal{B}'} = \mathbf{R}_U^{\mathcal{B}''} := \mathbf{R}_U^{\mathcal{B}} \quad \text{for all } U \in \mathfrak{s} \setminus \{S\}, \tag{15}$$

$$\mathbf{R}_S^{\mathcal{B}'} := \big\{g \in \mathbf{R}_S^{\mathcal{B}} \ : \ \big|\{h \in \mathbf{R}_T^{\mathcal{B}} \ : \ \pi_S(h) = g\}\big| \leqslant m^{\varepsilon/2} \cdot \operatorname{avgdeg}(S,T)\big\} \tag{16}$$

$$\mathbf{R}_S^{\mathcal{B}''} := \big\{g \in \mathbf{R}_S^{\mathcal{B}} \ : \ \big|\{h \in \mathbf{R}_T^{\mathcal{B}} \ : \ \pi_S(h) = g\}\big| > m^{\varepsilon/2} \cdot \operatorname{avgdeg}(S,T)\big\} \tag{17}$$

It is clear that $[\![\varphi]\!]^{\mathcal{B}}$ is the disjoint union of $[\![\varphi]\!]^{\mathcal{B}'}$ and $[\![\varphi]\!]^{\mathcal{B}''}$ and that the recursion terminates at some point with a sequence of strongly m^c-consistent ϵ-uniform refinements that partition $[\![\varphi]\!]^{\mathcal{A}}$. It is also not hard to show that the height of the recursion tree is bounded by $2^{O(|\mathrm{vars}(\varphi)|)} \cdot \frac{c}{\epsilon}$ (see [33, Lemma 4.11]). Hence, by Lemma 4.3 the procedure can be implemented in time $O(f(\varphi, c, \epsilon) \cdot m^{2c})$ and space $O(f(\varphi, c, \epsilon) \cdot m^c)$. ◀

The nice thing about ϵ-uniform and strongly m^c-consistent refinements is that they define, for small enough ϵ, a submodular function $g \in \mathsf{S}(\varphi)$, which in turn guarantees the existence of a tree decomposition with small projections on the bags. The following lemma from [33, Lemma 4.12] provides these functions. However, there is an oversight in Marx's proof and in order to fix this, we have to ensure *strong* m^c-consistency instead of only m^c-consistency as stated in [33, Lemma 4.12]. As suggested by Marx (personal communication), an alternative way to achieve strong m^c-consistency would be to enforce m^{2c}-consistency, which leads to the same runtime guarantees, but requires more space.

▶ **Lemma 4.6.** *Let $(\mathfrak{s}, \mathcal{B})$ be an ϵ-uniform strongly m^c-consistent refinement of φ and $\mathcal{A}$, and let $c \geqslant 1$ and $|\mathrm{vars}(\varphi)|^{-3} \geqslant \epsilon > 0$ be real numbers. Then $g_{\mathfrak{s},\mathcal{B}} \colon 2^{\mathrm{vars}(\varphi)} \to \mathbb{R}_{\geqslant 0}$ is a monotone, edge-dominated, submodular function that satisfies $g_{\mathfrak{s},\mathcal{B}}(\emptyset) = 0$:*

$$g_{\mathfrak{s},\mathcal{B}}(U) := \begin{cases} (1-\varepsilon^{1/3}) \cdot \log_m \left(|\mathbf{R}^{\mathcal{B}}_U|\right) + h(U) & \text{if } U \in \mathfrak{s} \\ (1-\varepsilon^{1/3}) \cdot c + h(U) & \text{if } U \notin \mathfrak{s}, \end{cases} \tag{18}$$

where $h(U) := 2\epsilon^{2/3}|U| - \epsilon|U|^2 \geqslant 0$ for all $U \subseteq \mathrm{vars}(\varphi)$.

The proof can be copied verbatim from Marx's proof of [33, Lemma 4.12] by using the notion of strong consistency instead of plain consistency and is provided in the full version of the paper. Now we are ready to prove our main theorem.

Proof of Theorem 3.5. We fix $c = (1+\delta)w$ and let ε be the minimum of $\big(1 - 1/(1+\delta)\big)^4$ and $|\mathrm{vars}(\varphi)|^{-4}$. Suppose that φ is of the form $\exists x_1 \cdots \exists x_k \, \widetilde{\varphi}$ where $\widetilde{\varphi}$ is quantifier-free. We apply Lemma 4.5 to $\widetilde{\varphi}$, $\mathcal{A}$, c, ε to obtain in time $O(f(\varphi)m^{2c})$ a sequence of $\ell \leqslant f(\varphi)$ strongly m^c-consistent ε-uniform refinements $(\mathfrak{s}_i, \mathcal{B}_i)$ such that $[\![\widetilde{\varphi}]\!]^{\mathcal{A}}$ is the disjoint union of $[\![\widetilde{\varphi}_{\mathfrak{s}_1}]\!]^{\mathcal{B}_1}$, ..., $[\![\widetilde{\varphi}_{\mathfrak{s}_\ell}]\!]^{\mathcal{B}_\ell}$. By Lemma 4.6 we have $g_{\mathfrak{s}_i,\mathcal{B}_i} \in \mathsf{S}(\widetilde{\varphi}) = \mathsf{S}(\varphi)$ for every $i \in [\ell]$. Hence, by the definition of free-connex submodular width (5), we know that there is a free-connex tree decomposition (T_i, χ_i) of φ such that $g_{\mathfrak{s}_i,\mathcal{B}_i}(\chi_i(t)) \leqslant w$ for every $t \in V(T_i)$. Note that by the choice of c , ϵ and the non-negativity of h (see Lemma 4.6) we have

$$w = c/(1+\delta) \leqslant (1-\varepsilon^{1/4}) \cdot c < (1-\varepsilon^{1/3}) \cdot c + h(U). \tag{19}$$

Hence, $g_{\mathfrak{s}_i,\mathcal{B}_i}(U) \leqslant w$ implies $U \in \mathfrak{s}$ and therefore $|\mathbf{R}^{\mathcal{B}_i}_U| = |\,[\![\varphi_{\mathfrak{s}_i}\langle U\rangle]\!]^{\mathcal{B}_i}| \leqslant m^c$ by (9) and (11). Thus, every bag of the free-connex tree-decomposition (T_i, χ_i) is small in the ith refinement. However, (T_i, χ_i) is a tree-decomposition of φ, but not necessarily of $\varphi_{\mathfrak{s}_i}$! In fact, $\varphi_{\mathfrak{s}_i}$ can be very dense, e. g., if $\mathfrak{s}_i = 2^{\mathrm{vars}(\varphi)}$. To take care of this, we thin out the refinement and only keep those atoms and relations that correspond to bags of the decomposition. In particular, for every $i \in [\ell]$ we define $\widetilde{\psi}_i := \bigwedge_{t \in V(T_i)} R_{\chi_i(t)}(x_{\chi_i(t)})$ and let $\psi_i := \exists x_1 \cdots \exists x_k \, \widetilde{\psi}_i$ be the quantified version. Note that ψ_i is a free-connex acyclic CQ. Additionally, we let $\mathcal{C}_i$ be the $\sigma(\psi_i)$-reduct of $\mathcal{B}_i$. We argue that $[\![\widetilde{\varphi}_{\mathfrak{s}_i}]\!]^{\mathcal{B}_i} \subseteq [\![\widetilde{\psi}_i]\!]^{\mathcal{C}_i} \subseteq [\![\widetilde{\varphi}]\!]^{\mathcal{A}}$. The first inclusion holds because $\widetilde{\varphi}_{\mathfrak{s}_i}$ and $\mathcal{B}_i$ refine $\widetilde{\psi}_i$ and $\mathcal{C}_i$. The second inclusion holds because every atom from $\widetilde{\varphi}$ is contained in a bag of the decomposition and is hence covered by an atom in $\widetilde{\psi}_i$ because of consistency. It therefore also follows that $\pi_F\big([\![\widetilde{\varphi}_{\mathfrak{s}_i}]\!]^{\mathcal{B}_i}\big) \subseteq \pi_F\big([\![\widetilde{\psi}_i]\!]^{\mathcal{C}_i}\big) \subseteq \pi_F\big([\![\widetilde{\varphi}]\!]^{\mathcal{A}}\big)$ for $F := \mathrm{free}(\varphi)$, and hence $[\![\varphi_{\mathfrak{s}_i}]\!]^{\mathcal{B}_i} \subseteq [\![\psi_i]\!]^{\mathcal{C}_i} \subseteq [\![\varphi]\!]^{\mathcal{A}}$. Overall, we have that $[\![\varphi]\!]^{\mathcal{A}} = \bigcup_{i \in [\ell]} [\![\psi_i]\!]^{\mathcal{C}_i}$, where the union is not necessarily disjoint, each ψ_i is free-connex acyclic, and $\|\mathcal{C}_i\| = O(|\mathrm{vars}(\varphi)|^2 m^{(1+\delta)w})$. By combining Theorem 3.3 with Theorem 2.1, the theorem follows. ◀

5 Final Remarks

In this paper, we have investigated the enumeration complexity of conjunctive queries and have shown that every class of conjunctive queries of bounded free-connex submodular width admits constant delay enumeration with FPT-preprocessing. These are by now the largest classes of CQs that allow efficient enumeration in this sense.

For quantifier-free self-join-free CQs this upper bound is matched by Marx's lower bound [33]. I. e., recursively enumerable classes of quantifier-free self-join-free CQs of unbounded free-connex submodular width do not admit constant delay enumeration after FPT-preprocessing (assuming the exponential time hypothesis ETH).

A major future task is to obtain a complete dichotomy, or at least one for all self-join-free CQs. The gray-zone for the latter are classes of CQs that have bounded submodular width, but unbounded free-connex submodular width. An intriguing example in this gray-zone is the k-star query with a quantified center, i. e., the query ψ_k of the form $\exists z \bigwedge_{i=1}^{k} R_i(z, x_i)$. Here we have $\mathit{subw}(\psi_k) = 1$ and $\mathit{fc\text{-}subw}(\psi_k) = k$. It is open whether the class $\Psi = \{\psi_k : k \in \mathbb{N}_{\geqslant 1}\}$ admits constant delay enumeration with FPT-preprocessing.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

2 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do Shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? *CoRR*, abs/1612.02503, 2016. arXiv:1612.02503.

3 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What Do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another? In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems PODS 2017*, pages 429–444, 2017. doi:10.1145/3034786.3056105.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and Counting Given Length Cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

5 Antoine Amarilli, Pierre Bourhis, and Stefan Mengel. Enumeration on Trees under Relabelings. In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, pages 5:1–5:18, 2018. doi:10.4230/LIPIcs.ICDT.2018.5.

6 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on Trees with Tractable Combined Complexity and Efficient Updates. *CoRR*, abs/1812.09519, 2018. arXiv:1812.09519.

7 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-Delay Enumeration for Nondeterministic Document Spanners. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, pages 22:1–22:19, 2019. doi:10.4230/LIPIcs.ICDT.2019.22.

8 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, pages 167–181, 2006. doi:10.1007/11874683_11.

9 Guillaume Bagan. *Algorithmes et complexité des problèmes d'énumération pour l'évaluation de requêtes logiques. (Algorithms and complexity of enumeration problems for the evaluation of logical queries)*. PhD thesis, University of Caen Normandy, France, 2009. URL: https://tel.archives-ouvertes.fr/tel-00424232.

10 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *Proceedings of the 16th Annual Conference of the EACSL, CSL'07, Lausanne, Switzerland, September 11–15, 2007*, pages 208–222, 2007. doi:10.1007/978-3-540-74915-8_18.

11 Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the Desirability of Acyclic Database Schemes. *J. ACM*, 30(3):479–513, 1983. doi:10.1145/2402.322389.

12 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering Conjunctive Queries under Updates. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS'17, Chicago, IL, USA, May 14–19, 2017*, pages 303–318, 2017. Full version available at `http://arxiv.org/abs/1702.06370`. `doi:10.1145/3034786.3034789`.

13 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD Queries Under Updates on Bounded Degree Databases. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory, ICDT 2017, March 21–24, 2017, Venice, Italy*, volume 68 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.ICDT.2017.8`.

14 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs under Updates and in the Presence of Integrity Constraints. In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, pages 8:1–8:19, 2018. `doi:10.4230/LIPIcs.ICDT.2018.8`.

15 Philip A. Bernstein and Nathan Goodman. Power of Natural Semijoins. *SIAM J. Comput.*, 10(4):751–771, 1981. `doi:10.1137/0210059`.

16 Johann Brault-Baron. *De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre. (The relevance of the list: propositional logic and complexity of the first order)*. PhD thesis, University of Caen Normandy, France, 2013. URL: `https://tel.archives-ouvertes.fr/tel-01081392`.

17 Johann Brault-Baron. Hypergraph Acyclicity Revisited. *ACM Comput. Surv.*, 49(3):54:1–54:26, 2016. `doi:10.1145/2983573`.

18 Shaleen Deep and Paraschos Koutris. Compressed Representations of Conjunctive Query Results. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 307–322, 2018. `doi:10.1145/3196959.3196979`.

19 Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4), 2007. `doi:10.1145/1276920.1276923`.

20 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22–27, 2014*, pages 121–131, 2014. `doi:10.1145/2594538.2594539`.

21 Arnaud Durand and Yann Strozecki. Enumeration Complexity of Logical Query Problems with Second-order Variables. In *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway, Proceedings*, pages 189–202, 2011. `doi:10.4230/LIPIcs.CSL.2011.189`.

22 Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. Hypertree Decompositions: Questions and Answers. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 57–74, 2016. `doi:10.1145/2902251.2902309`.

23 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree Decompositions and Tractable Queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002. `doi:10.1006/jcss.2001.1809`.

24 Martin Grohe and Dániel Marx. Constraint Solving via Fractional Edge Covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014. `doi:10.1145/2636918`.

25 Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. The Dynamic Yannakakis Algorithm: Compact and Efficient Query Processing Under Updates. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1259–1274, 2017. `doi:10.1145/3035918.3064027`.

26 Ahmet Kara and Dan Olteanu. Covers of Query Results. In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, pages 16:1–16:22, 2018. `doi:10.4230/LIPIcs.ICDT.2018.16`.

27 Wojciech Kazana and Luc Segoufin. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science*, 7(2), 2011. doi:10.2168/LMCS-7(2:20)2011.

28 Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 297–308, 2013. doi:10.1145/2463664.2463667.

29 Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. Comput. Log.*, 14(4):25:1–25:12, 2013. doi:10.1145/2528928.

30 Dietrich Kuske and Nicole Schweikardt. First-order logic with counting. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005133.

31 Katja Losemann and Wim Martens. MSO queries on trees: enumerating answers under updates. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 67:1–67:10, 2014. doi:10.1145/2603088.2603137.

32 Dániel Marx. Tractable Structures for Constraint Satisfaction with Truth Tables. *Theory Comput. Syst.*, 48(3):444–464, 2011. doi:10.1007/s00224-009-9248-9.

33 Dániel Marx. Tractable Hypergraph Properties for Constraint Satisfaction and Conjunctive Queries. *Journal of the ACM (JACM), Volume 60, Issue 6, Article No. 42*, November 2013. doi:10.1145/2535926.

34 Matthias Niewerth. MSO queries on trees: Enumerating answers under updates using forest algebras. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 769–778, 2018. doi:10.1145/3209108.3209144.

35 Matthias Niewerth and Luc Segoufin. Enumeration of MSO Queries on Strings with Constant Delay and Logarithmic Updates. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 179–191, 2018. doi:10.1145/3196959.3196961.

36 Dan Olteanu and Maximilian Schleich. Factorized Databases. *SIGMOD Record*, 45(2):5–16, 2016. doi:10.1145/3003665.3003667.

37 Dan Olteanu and Jakub Závodný. Size Bounds for Factorised Representations of Query Results. *ACM Trans. Database Syst.*, 40(1):2:1–2:44, 2015. doi:10.1145/2656335.

38 Francesco Scarcello. From Hypertree Width to Submodular Width and Data-dependent Structural Decompositions. In *Proceedings of the 26th Italian Symposium on Advanced Database Systems, Castellaneta Marina (Taranto), Italy, June 24-27, 2018.*, 2018. URL: http://ceur-ws.org/Vol-2161/paper24.pdf.

39 Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO Queries over Nowhere Dense Graphs. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 151–163, 2018. doi:10.1145/3196959.3196971.

40 Luc Segoufin. Constant Delay Enumeration for Conjunctive Queries. *SIGMOD Record*, 44(1):10–17, 2015. doi:10.1145/2783888.2783894.

41 Luc Segoufin and Alexandre Vigny. Constant Delay Enumeration for FO Queries over Databases with Local Bounded Expansion. In *20th International Conference on Database Theory, ICDT 2017, March 21–24, 2017, Venice, Italy*, pages 20:1–20:16, 2017. doi:10.4230/LIPIcs.ICDT.2017.20.

42 Mihalis Yannakakis. Algorithms for Acyclic Database Schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 82–94, 1981.

Counting Homomorphisms Modulo a Prime Number

Amirhossein Kazeminia
Simon Fraser University, Canada
amirhossein.kazeminia@sfu.ca

Andrei A. Bulatov
Simon Fraser University, Canada
abulatov@sfu.ca

Abstract

Counting problems in general and counting graph homomorphisms in particular have numerous applications in combinatorics, computer science, statistical physics, and elsewhere. One of the most well studied problems in this area is $\#\mathsf{GraphHom}(H)$ – the problem of finding the number of homomorphisms from a given graph G to the graph H. Not only the complexity of this basic problem is known, but also of its many variants for digraphs, more general relational structures, graphs with weights, and others. In this paper we consider a modification of $\#\mathsf{GraphHom}(H)$, the $\#_p\mathsf{GraphHom}(H)$ problem, p a prime number: Given a graph G, find the number of homomorphisms from G to H modulo p. In a series of papers Faben and Jerrum, and Göbel et al. determined the complexity of $\#_2\mathsf{GraphHom}(H)$ in the case H (or, in fact, a certain graph derived from H) is square-free, that is, does not contain a 4-cycle. Also, Göbel et al. found the complexity of $\#_p\mathsf{GraphHom}(H)$ when H is a tree for an arbitrary prime p. Here we extend the above result to show that the $\#_p\mathsf{GraphHom}(H)$ problem is $\#_p$P-hard whenever the derived graph associated with H is square-free and is not a star, which completely classifies the complexity of $\#_p\mathsf{GraphHom}(H)$ for square-free graphs H.

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

Keywords and phrases graph homomorphism, modular counting, computational hardness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.59

Acknowledgements This work was supported by an NSERC Discovery grant

1 Introduction

A homomorphism from a graph G to a graph H is an edge-preserving mapping from the vertex set of G to that of H. Graph homomorphisms provide a powerful framework to model a wide range of combinatorial problems in computer science, as well as a number of phenomena in combinatorics and graph theory, such as graph parameters [13, 14]. Two of the most natural problems related to graph homomorphisms is $\mathsf{GraphHom}(H)$: Given a graph G, decide whether there is a homomorphism from G to a fixed graph H, and its counting version $\#\mathsf{GraphHom}(H)$ of finding the number of such homomorphisms. Special cases of these problems include the k-Colouring and $\#k$-Colouring problems (H is a k-clique), Bipartiteness (H is an edge), counting independent sets (H is an edge with a loop at one vertex) and many others.

In general the $\mathsf{GraphHom}(H)$ and $\#\mathsf{GraphHom}(H)$ problems are NP-complete and #P-complete, respectively. However, for certain graphs H these problems are significantly easier. Hell and Nesetril [12] were the first to address this phenomenon in a systematic way. They proved that the $\mathsf{GraphHom}(H)$ problem is polynomial time solvable if and only if H has a loop or is bipartite, and $\mathsf{GraphHom}(H)$ is NP-complete otherwise. In the counting case a similar result was obtained by Dyer and Greenhill [3], in this case the $\#\mathsf{GraphHom}(H)$ problem

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 59; pp. 59:1–59:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is solvable in polynomial time if and only if H a complete graph with all loops present or a complete bipartite graph, otherwise the problem is #P-complete. This result was later generalized to computing partition functions for weighted graphs with nonnegative weights by Bulatov and Grohe [1], graphs with real weights by Goldberg et al. [10], and finally for complex weights by Cai et al. [2]. There have also been major attempts to find approximation algorithms for the number of homomorphisms and other related graph parameters, see, e.g. [5, 6, 11].

The modification of the $\#\mathsf{GraphHom}(H)$ problem we consider in this paper concerns finding the number of homomorphisms modulo a natural number k. The corresponding problem will be denoted by $\#_k\mathsf{GraphHom}(H)$. Although modular counting has been considered by Valiant [17] in the context of holographic algorithms, Faben and Jerrum [4] were the first who systematically considered the problem $\#_2\mathsf{GraphHom}(H)$. In particular, they posed a conjecture stating that this problem is polynomial time solvable if and only if a certain graph H^{*2} derived from H (to be defined later in this section) contains at most one vertex, and is complete in the class $\oplus\mathrm{P} = \#_2\mathrm{P}$ otherwise. Note that hardness results in this area usually show completeness in a complexity class $\#_k\mathrm{P}$ of counting the number of accepting paths in polynomial time nondeterministic Turing machines modulo k. The standard notion of reduction in this case is Turing reduction. Faben and Jerrum proved their conjecture in the case when H is a tree. This result has been extended by Göbel et al. first to the class of cactus graphs [7] and then to square-free graphs [8] (a graph is a square-free if it does not contain a 4-cycle).

In this paper we follow the lead of Göbel et al. [9] and consider the problem $\#_p\mathsf{GraphHom}(H)$ for a prime number p. We only consider loopless graphs without parallel edges. There are similarities with the $\pmod 2$ case. In particular, the derived graph constructed in [4] can also be constructed following the same principles, it is denoted H^{*p}, and it suffices to study $\#_p\mathsf{GraphHom}(H)$ for this only. On the other hand, the problem is richer, as, for example, the polynomial time solvable cases include complete bipartite graphs. Göbel et al. [9] considered the case when H is a tree. Recall that a *star* is a complete bipartite graph of the form $K_{1,n}$. Stars are the only complete bipartite graphs that are trees, and also the only complete bipartite graphs that are square-free. The main result of [9] establishes that $\#_p\mathsf{GraphHom}(H)$, H is a tree, is polynomial time solvable if and only if H^{*p} is a star. We generalize this result to arbitrary square-free graphs.

▶ **Theorem 1.** *Let H be a square-free graph and p a prime number. Then the $\#_p\mathsf{GraphHom}(H)$ problem is solvable in polynomial time if and only if the graph H^{*p} is a star, and is $\#_p$P-complete otherwise.*

We now explain the main ideas behind our result, as well as, the majority of results in this area. As it was observed by Faben and Jerrum [4], the automorphism group $\mathsf{Aut}(H)$ of graph H plays a very important role in solving the $\#_p\mathsf{GraphHom}(H)$ problem. Let φ be a homomorphism from a graph G to H. Then composing φ with an element from $\mathsf{Aut}(H)$ we again obtain a homomorphism from G to H. The set of all such homomorphisms forms the orbit of φ under the action of $\mathsf{Aut}(H)$. If $\mathsf{Aut}(H)$ contains an automorphism π of order p (that is, p is the smallest number such that π^p is the identity permutation), the cardinality of the orbit of φ is divisible by p, unless $\pi \circ \varphi = \varphi$, that is, the range of φ is the set of fixed points $\mathsf{Fix}(\pi)$ of π ($a \in V(H)$ is a fixed point of π if $\pi(a) = a$). Let H^π denote the subgraph of H induced by $\mathsf{Fix}(\pi)$. We write $H \Rightarrow_p H'$ if there is $\pi \in \mathsf{Aut}(H)$ such that H' is isomorphic to H^π. We also write $H \Rightarrow_p^* H'$ if there are graphs $H_1, \ldots, H_k$ such that H is isomorphic to H_1, H' is isomorphic to H_k, and $H_1 \Rightarrow_p H_2 \Rightarrow_p \cdots \Rightarrow_p H_k$.

▶ **Lemma 2** ([4]). *Let H be a graph and p a prime. Up to an isomorphism there is a unique smallest (in terms of the number of vertices) graph H^{*p} such that $H \Rightarrow_p^* H^{*p}$, and for any graph G it holds*

$$|\mathsf{Hom}(G,H)| \equiv |\mathsf{Hom}(G,H^{*p})| \pmod{p}.$$

Moreover, H^{*p} does not have automorphisms of order p.

The easiness part of Theorem 1 follows from the classification of the complexity of $\#\mathsf{GraphHom}(H)$ by Dyer and Greenhill [3]. Since whenever $\#\mathsf{GraphHom}(H)$ is polynomial time solvable, so is $\#_p\mathsf{GraphHom}(H)$ for any p, Lemma 2 implies that if H^{*p} is a complete graph with all loops present or a complete bipartite graph the problem $\#_p\mathsf{GraphHom}(H)$ is also solvable in polynomial time. We restrict ourselves to loopless square-free graphs, therefore, as H^{*p} is isomorphic to an induced subgraph of H, [3] only guarantees polynomial time solvability when H^{*p} is a star.

Another ingredient in our result is the $\#_p$P-hard problem we reduce to $\#_p\mathsf{GraphHom}(H)$. In most of the cited works the hard problem used to prove the hardness of $\#_2\mathsf{GraphHom}(H)$ is the problem $\#_2 IS$ of finding the parity of the number of independent sets. This problem was shown to be $\#_2$P-complete by Valiant [17]. We use a slightly different problem. For two positive real numbers λ_1, λ_2, let $\#_p BIS_{\lambda_1,\lambda_2}$ denote the following problem of counting weighted independent sets in bipartite graphs

Name: $\#_p BIS_{\lambda_1,\lambda_2}$
Input: a bipartite graph G
Output: $Z_{\lambda_1,\lambda_2}(G) = \sum_{I \in \mathcal{IS}(G)} \lambda_1^{|V_L \cap I|} \lambda_2^{|V_R \cap I|} \pmod{p}$.

It was shown by Göbel et al. in [9] that $\#_p BIS_{\lambda_1,\lambda_2}$ is $\#_p$P-complete for any λ_1, λ_2, unless one of them is equal to $0 \pmod{p}$. The main technical statement we prove here is the following

▶ **Theorem 3.** *Let H be a square-free graph such that H^{*p} is not a star. Then there are $\lambda_1, \lambda_2 \not\equiv 0 \pmod{p}$ such that $\#_p BIS_{\lambda_1,\lambda_2}$ is polynomial time reducible to the $\#_p\mathsf{GraphHom}(H)$ problem.*

We note that the requirement of being square-free is present in all results on modular counting of graph homomorphisms, explicitly or implicitely (when the graph class in question consists of square-free graphs). Clearly, this is an artifact of the techniques used in all these works, and so overcoming this requirement would be a substantial achievement.

2 Preliminaries

We use $[n]$ to denote the set $\{1,2,3,...,n\}$. Also, we usually abbreviate $A \setminus \{x\}$ to $A - x$. Let k be a positive integer, then for a function f its k-fold composition is denoted by $f^{(k)} = f \circ f \circ \cdots \circ f$.

Graphs. In this paper, *graphs* are undirected, and have no parallel edges or loops. For a graph G, the set of vertices of G is denoted by $V(G)$, and the set of edges is denoted by $E(G)$. We use uv to denote an edge of G. The set of *neighbours* of a vertex $v \in V(G)$ is denoted by $N_G(v) = \{u \in V(G) : uv \in E(G)\}$, and the degree of v is denoted by $\mathsf{deg}(v)$.

A set $I \subseteq V(G)$ is an *independent set* of G if and only if uv is an edge of G for no $u, v \in I$. The set of all independent sets of G is denoted by $\mathcal{IS}(G)$. If G is a bipartite graph, the parts of a bipartition of $V(G)$ will be denoted $V_R(G)$ and $V_L(G)$ in no particular order.

Homomorphisms. A *homomorphism* from a graph G to a graph H is a mapping φ from $V(G)$ to $V(H)$ which preserves edges, i.e. for any $uv \in E(G)$ the pair $\varphi(u)\varphi(v)$ is an edge of H. The set of all homomorphisms from G to H, is denoted by $\mathsf{Hom}(G, H)$. For a graph H the problem of counting homomorphism from a graph G to H is denoted by $\#\mathsf{GraphHom}(H)$. The problem of finding the number of homomorphisms from a given graph G to H modulo k is denoted by $\#_k\mathsf{GraphHom}(H)$:

Name: $\#_k\mathsf{GraphHom}(H)$
Input: a graph G
Output: $|\mathsf{Hom}(G, H)| \pmod{k}$.

It will be convenient to denote the vertices of the graph H by lowercase Greek letters.

A homomorphism φ from G to H is an *isomorphism* if it is bijective and for all $u, v \in V(G)$, $uv \in E(G)$ if and only if $\varphi(u)\varphi(v) \in E(H)$. An *automorphism* of G is an isomorphism from graph G to itself. The automorphism group of G is denoted by $\mathsf{Aut}(G)$. An automorphism π is an *automorphism of order k* if k is the smallest positive integer such that $\pi^{(k)}$ is the identity transformation. A *fixed point* of an automorphism π of G is a vertex $v \in V(G)$ such that $v = \pi(v)$.

Partially labelled graphs. A *partial function* from X to Y is a function $f : X' \to Y$ for a subset $X' \subseteq X$. For a graph H, a *partial H-labelled graph* $\mathcal{G}$ is a graph G (called the *underlying graph* of $\mathcal{G}$) equipped with a *pinning function* τ, which is a partial function from $V(G)$ to $V(H)$. A homomorphism from a partial H-labelled graph $\mathcal{G} = (G, \tau)$ to a graph H is a homomorphism $\sigma : G \to H$ that extends the pinning function τ, that is, for all $v \in \mathrm{dom}(\tau)$, $\sigma(v) = \tau(v)$. The set of all such homomorphisms is denoted by $\mathsf{Hom}(\mathcal{G}, H)$.

In certain situations it will be convenient to use a slightly different view on collections of homomorphisms of H-labelled graphs. A set of homomorphisms φ from a graph G to H that maps vertices $x_1, x_2, ..., x_r \in V(G)$ to the vertices $y_1, y_2, ..., y_r \in V(H)$ such that $\varphi(x_i) = y_i$ for $i \in [r]$ is denoted by $\mathsf{Hom}((G, x_1, x_2, ..., x_r), (H, y_1, y_2, ..., y_r))$.

Counting complexity classes. The class #P is defined to be the class of problems of counting the accepting paths in a polynomial time nondeterministic Turing machine. This means every problem in NP has an associated counting problem in #P, so for $A \in \mathrm{NP}$, an associated counting problem will be denoted by $\#A$. (Strictly speaking for every such problem the corresponding counting one is not uniquely defined, but in our case there will always be the "natural" one.) Classes $\#_k\mathrm{P}$, where k is a natural number are defined in a similar way, as counting the accepting paths in a polynomial time nondeterministic Turing machine modulo k. For $A \in \mathrm{NP}$ the corresponding problem in $\#_k\mathrm{P}$ is denoted by $\#_kA$.

Several kinds of reductions between counting problems have appeared in the literature. The first one, parsimonious, was introduced in the foundational papers [15, 16] by Valiant. A counting problem A is *parsimoniously reducible* to a counting problem B, denoted $A \leq B$, if there is a polynomial time algorithm that, given an instance I of A, produces an instance J of B such that the answers to I and J are the same. The other type of reduction frequently used for counting problems is Turing reduction. Counting problem A is *Turing reducible* to problem B, denoted $A \leq_T B$, if there exists a polynomial time algorithm solving A and using B as an oracle.

These two types of reductions can be applied to modular counting as well. Turing reduction does not require any modifications. For parsimonious reduction we say that a problem A from $\#_k\mathrm{P}$ is *parsimoniously reducible* to a problem B from $\#_k\mathrm{P}$ if there is a polynomial time algorithm that, given an instance I of A, produces an instance J of B such that the answers to I and J are congruent modulo k. In this paper we mostly claim Turing reducibility, although our main technical result constructs a parsimonious reduction. However, the proof of Theorem 1 involves other reductions that are not always parsimonious. Problem $\#_k A$ is said to be $\#_k$P-*complete* if it belongs to $\#_k$P and every problem from $\#_k$P is Turing reducible to $\#_k A$.

3 Outline of the proof

In this section we outline our proof strategy and formally introduce all the necessary intermediate problems and existing results. Fix a prime number p.

As it was observed in the introduction, Lemma 2 proved by Faben and Jerrum [4] combined with the classification by Dyer and Greenhill [3] proves the easiness part of Theorem 1. We therefore focus on proving the hardness part. Again, by Lemma 2 we may assume that H does not have automorphisms of order p.

For the hardness part, we use two auxiliary problems. The first one is the problem $\#_p BIS_{\lambda_1,\lambda_2}$ mentioned in the introduction. Let $\lambda_1, \lambda_2 \in \{0, \dots p-1\}$, and let $G = (V_L \cup V_R, E)$ be a bipartite graph. Define the following weighted sum over independent sets of G:

$$Z_{\lambda_1,\lambda_2}(G) = \sum_{I \in \mathcal{IS}(G)} \lambda_1^{|V_L \cap I|} \lambda_2^{|V_R \cap I|}.$$

The problem of computing function $Z_{\lambda_1,\lambda_2}(G)$ for a given bipartite graph G, prime number p and $\lambda_1, \lambda_2 \in \{0, \dots p-1\}$, is defined as follows:

Name: $\#_p BIS_{\lambda_1,\lambda_2}$
Input: a bipartite graph G
Output: $Z_{\lambda_1,\lambda_2}(G) \pmod p$.

The complexity of $\#_p BIS_{\lambda_1,\lambda_2}$ was determined by Göbel, Lagodzinski and Seidel [9].

▶ **Theorem 4.** *[9] If $\lambda_1 \equiv 0 \pmod p$ or $\lambda_2 \equiv 0 \pmod p$ then the problem $\#_p BIS_{\lambda_1,\lambda_2}$ is solvable in polynomial time, otherwise it is $\#_p$P-complete.*

The second auxiliary problem has been used in all works on $\#_p\mathsf{GraphHom}(H)$ starting from the initial paper by Faben and Jerrum [4]. It is the problem of counting homomorphisms from a given partially H-labelled graph $\mathcal{G}$ to a fixed graph H modulo prime p.

Name: $\#_p\mathsf{PartHom}(H)$
Input: a partial H-labelled graph $\mathcal{G} = (G, \tau)$
Output: $|\mathsf{Hom}(\mathcal{G}, H)| \pmod p$.

The chain of reductions we use to prove the hardness part of Theorem 1 is the following:

$$\#_p BIS_{\lambda_1,\lambda_2} \le_T \#_p\mathsf{PartHom}(H^{*p}) \le_T \#_p\mathsf{GraphHom}(H^{*p}) \le_T \#_p\mathsf{GraphHom}(H). \qquad (1)$$

The last reduction is by Lemma 2. Acually, the two last problems in the chain are polynomial time interreducible through (modular) parsimonious reduction. The second step, the reduction from $\#_p\mathsf{PartHom}(H)$ to $\#_p\mathsf{GraphHom}(H)$ was proved by Göbel, Lagodzinski and Seidel [9].

▶ **Theorem 5.** *[9] Let p be a prime number and let H be a graph that does not have any automorphism of order p. Then $\#_p\mathsf{PartHom}(H)$ can be reduced to $\#_p\mathsf{GraphHom}(H)$ through a polynomial time Turing reduction.*

Finally, the first reduction in the chain is our main contribution. We show it in three steps. Recall that we are reducing the problem of finding of the number of (weighted) independent sets in a bipartite graph to the problem of finding the number of extensions of a partial homomorphism from a given graph to H. First, in Section 4 starting from a bipartite graph G we replace its vertices and edges with gadgets, whose exact structure we do not specify at that point. We call those gadget the *vertex* and *edge* gadgets. Then we show that if the vertex and edge gadgets satisfy certain conditions, in terms of the number of homomorphisms of certain kind from the gadgets to H (Theorem 7), then $Z_{\lambda_1,\lambda_2}(G) \pmod p$ can be found in polynomial time from $|\mathsf{Hom}(\mathcal{G}, H)| \pmod p$, where $\mathcal{G}$ is the partially H-labelled graph constructed in the reduction. In the second step, Section 5, we introduce several variants of vertex and edge gadgets and show some of their properties. Finally, in Section 6 we consider several cases depending on the degree sequence of the graph H. In every case we construct vertex and edge gadgets that satisfy the conditions of Theorem 7, thus completing the reduction.

4 Hardness gadgets

Our goal in this section is to describe a general scheme of a reduction from $\#_pBIS_{\lambda_1,\lambda_2}$ to $\#_p\mathsf{PartHom}(H)$, where p is prime and H is a square-free graph.

The general idea is, given a bipartite graph $G = (V_L \cup V_R, E)$, where V_L, V_R is the bipartition of G, to construct a new partially H-labelled graph $\mathcal{G}'$, which is obtained from G by adding a copy of a *vertex* gadget $\mathcal{J}$ to every vertex of G, and replacing every edge from E with a copy of an *edge* gadget $\mathcal{K}$. The gadgets are partially H-labelled graphs and their pinning functions will define the pinning function of $\mathcal{G}'$. Since G is a bipartite graph, the vertex gadget comes in two versions, left, $\mathcal{J}_L$, and right, $\mathcal{J}_R$. Also, both vertex gadgets have a distinguished vertex, s for $\mathcal{J}_L$ and t for $\mathcal{J}_R$. The edge gadget $\mathcal{K}$ has two distinguished vertices, s and t. These distinguished vertices will be identified with the vertices of the original graph G, as shown in Fig. 1.

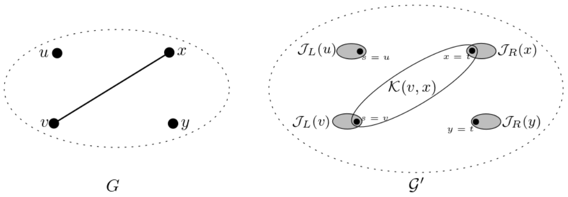


Figure 1 The structure of graph $\mathcal{G}'$. The original graph G is on the left. The resulting graph $\mathcal{G}'$ is on the right: vertex gadgets $\mathcal{J}_L, \mathcal{J}_R$ are added to every vertex, and the only edge vx of G is replaced with a copy of gadget $\mathcal{K}$.

The gadgets $\mathcal{J}_L, \mathcal{J}_R$ are associated with sets $\Delta_1, \Delta_2 \subseteq V(H)$ and vertices $\delta_1 \in \Delta_1, \delta_2 \in \Delta_2$, respectively. The pinning functions of $\mathcal{J}_L, \mathcal{J}_R$ will be defined in such a way that for any homomorphism φ of $\mathcal{J}_L$ ($\mathcal{J}_R$) to H, vertex s (respectively, t) is forced to be mapped to Δ_1 (respectively, Δ_2). For $x \in V_L$ let $\mathcal{J}_L(x)$ denote the copy of $\mathcal{J}_L$ connected to x, that is, s in $\mathcal{J}_L(x)$ is identified with x. For $y \in V_R$ the copy $\mathcal{J}_R(y)$ is defined in the same way. The vertices δ_1, δ_2 will help to encode independent sets of G. Specifically, with every independent set I of G we will associate a set of homomorphisms $\varphi : \mathcal{G}' \to H$ such that for every vertex $x \in V_L$, $x \in I$ if and only if $\varphi(x) \neq \delta_1$ (recall that x is also a vertex of $\mathcal{G}'$ identified with s in $\mathcal{J}_L(x)$); and similarly, for every $y \in V_R$, $y \in I$ if and only if $\varphi(y) \neq \delta_2$. Finally, the edge gadgets $\mathcal{K}(x, y)$ replacing every edge $xy \in E$ make sure that every homomorphism from $\mathcal{G}'$ to H is associated with an independent set.

Note that just an association of independent sets with collections of homomorphisms is not enough, the number of homomorphisms in those collections have to allow one to compute the function $Z_{\lambda_1,\lambda_2}(G)$.

Next we introduce conditions such that if for the graph H there are vertex and edge gadgets satisfying these conditions, then $\#_pBIS_{\lambda_1,\lambda_2}$ for some nonzero (modulo p) λ_1, λ_2 is reducible to $\#_p\mathsf{PartHom}(H)$.

▶ **Definition 6** (Hardness gadget). *A graph H has* hardness gadgets *if there are $\Delta_1, \Delta_2 \subseteq V(H)$, vertices $\delta_1 \in \Delta_1$ and $\delta_2 \in \Delta_2$, and three partially H-labelled graphs $\mathcal{J}_L$, $\mathcal{J}_R$, and $\mathcal{K}$ that satisfy the following properties:*

(i) $|\Delta_1| - 1 \not\equiv 0 \pmod p$, $|\Delta_2| - 1 \not\equiv 0 \pmod p$;

(ii) *for any homomorphism $\sigma : \mathcal{J}_L \to H$ ($\sigma : \mathcal{J}_R \to H$) it holds that $\sigma(s) \in \Delta_1$ (respectively, $\sigma(t) \in \Delta_2$); for any homomorphism $\sigma : \mathcal{K} \to H$ it holds that $\sigma(s) \in \Delta_1, \sigma(t) \in \Delta_2$;*

(iii) *for any $\gamma_1 \in \Delta_1, \gamma_2 \in \Delta_2$, it holds $|\mathsf{Hom}((\mathcal{J}_L, s), (H, \gamma_1))| \equiv |\mathsf{Hom}((\mathcal{J}_R, t), (H, \gamma_2))| \equiv 1 \pmod p$, and*
for any $\gamma_1 \notin \Delta_1, \gamma_2 \notin \Delta_2$, it holds $\mathsf{Hom}((\mathcal{J}_L, s), (H, \gamma_1)) = \mathsf{Hom}((\mathcal{J}_L, s), (H, \gamma_1)) = \emptyset$;

(iv) *for any $\alpha_1 \in \Delta_1 - \delta_1, \alpha_2 \in \Delta_2 - \delta_2$, it holds $\mathsf{Hom}((\mathcal{K}, s, t), (H, \alpha_1, \alpha_2)) = \emptyset$;*

(v) *for any $\alpha_1 \in \Delta_1 - \delta_1$, it holds $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \alpha_1, \delta_2))| \equiv 1 \pmod p$;*

(vi) *for any $\alpha_2 \in \Delta_2 - \delta_2$, it holds $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \delta_1, \alpha_2))| \equiv 1 \pmod p$;*

(vii) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \delta_1, \delta_2))| \equiv 1 \pmod p$.

Now we are ready to state the main result of this section.

▶ **Theorem 7.** *If H has hardness gadgets, then for some $\lambda_1, \lambda_2 \not\equiv 0 \pmod p$ the problem $\#_pBIS_{\lambda_1,\lambda_2}$ is polynomial time reducible to $\#_p\mathsf{PartHom}(H)$. In particular, $\#_p\mathsf{PartHom}(H)$ is $\#_pP$-complete.*

5 Hardness gadgets and nc-walks

In this section we make the next iteration in constructing hardness gadgets and give a generic structure of such gadgets that will later be adapted to specific types of the graph H.

These gadgets make use of the square-freeness of graph H that we will apply in the following form.

▶ **Observation 8.** *Let H be a square-free graph. Then for any $\alpha, \beta \in H$, $|N_H(\alpha) \cap N_H(\beta)| \leq 1$.*

Proof. If there are two different elements γ, δ in $N_H(\alpha) \cap N_H(\beta)$, then $\alpha, \gamma, \beta, \delta$ form a 4-cycle. ◀

We call a walk in H a *non-consecutive-walk* or *nc-walk*, if it does not traverse an edge forth and then immediately back. More formally, an nc-walk is a walk $v_0, v_1, \dots, v_k$ such that for no $i \in [k-1]$ we have $v_{i-1} = v_{i+1}$.

5.1 Edge gadget

Let $W = \gamma_0\gamma_1 \cdots \gamma_k$ be an nc-walk in H of length at least one. Then the edge gadget $\mathcal{K}$ is a path $sv_1v_2 \cdots v_{k-1}t$, where each v_i is connected to another vertex u_i which is pinned to γ_i. More formally, the gadget $\mathcal{K} = (K, \tau)$ is defined as follows

$$V(K) = \{s, t\} \cup \{v_i, u_i : i \in [k-1]\},$$
$$E(K) = \{v_iv_{i+1} : i \in [k-2]\} \cup \{v_iu_i : i \in [k-1]\} \cup \{sv_1, v_{k-1}t\}.$$

The pinning function is $\tau(u_i) = \gamma_i$ for all $i \in [k-1]$.

The next two lemmas give some of the properties listed in Definition 6.

▶ **Lemma 9 (Shifting).** *Let H, $W = \gamma_0\gamma_1 \cdots \gamma_k$, and $\mathcal{K}$ be as above. Then*

(1) *For every $\theta \in N_H(\gamma_0) - \gamma_1$ and $\sigma \in \mathsf{Hom}((\mathcal{K}, s), (H, \theta))$, we have $\sigma(v_i) = \gamma_{i-1}$ for all $i \in [k-1]$.*

(2) *For every $\theta \in N_H(\gamma_k) - \gamma_{k-1}$ and $\sigma \in \mathsf{Hom}((\mathcal{K}, t), (H, \theta))$, we have $\sigma(v_i) = \gamma_{i+1}$ for all $i \in [k-1]$.*

Proof. If $k = 1$, then both cases are trivial. We prove item (1) by induction on $j \in [k-1]$, item (2) can be proved using $\gamma_k\gamma_{k-1} \cdots \gamma_0$ instead of $\gamma_0\gamma_1 \cdots \gamma_k$.

For $j = 1$, the vertex v_1 must be mapped to a common neighbour of θ and γ_1 because $\tau(u_1) = \gamma_1$. It means $\sigma(v_1) \in N_H(\theta) \cap N_H(\gamma_1) = \{\gamma_0\}$, because $\gamma_0 \in N_H(\theta) \cap N_H(\gamma_1)$ and H is a square-free graph.

Now assume that $\sigma(v_{j-1}) = \gamma_{j-2}$. Similar to the base case, $\sigma(v_j) \in N_H(\gamma_{j-2}) \cap N_H(\gamma_j)$. By the same argument, the only member of this intersection is γ_{j-1}. Thus, $\sigma(v_j) = \gamma_{j-1}$. ◀

▶ **Lemma 10 (Counting).** *Let H be a square-free graph and let $W = \gamma_0\gamma_1 \cdots \gamma_k$, $k \geq 1$ be an nc-walk in H. For any $\alpha_s \in N_H(\gamma_0) - \gamma_1$ and $\alpha_t \in N_H(\gamma_k) - \gamma_{k-1}$ the following equalities hold*

(1) $\mathsf{Hom}((\mathcal{K}, s, t), (H, \alpha_s, \alpha_t)) = \emptyset$,

(2) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \gamma_1, \alpha_t))| = 1$,

(3) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \alpha_s, \gamma_{k-1}))| = 1$,

(4) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \gamma_1, \gamma_{k-1}))| = 1 + \sum_{i=1}^{k-1}(\mathsf{deg}(\gamma_i) - 1)$.

5.2 Vertex gadgets

In this section we construct a vertex gadget. The main role of these gadgets is to restrict the possible images of the designated vertices s and t as required in Definition 6(ii), and then do it in such a way that property (iii) in Definition 6 is also satisfied. We present vertex gadgets of two types.

For the graph H and vertices $\alpha, \beta \in V(H)$, we define gadgets $\mathcal{J}_L = (J_L, \tau_L)$ and $\mathcal{J}_R = (J_R, \tau_R)$ as follows: Graphs J_L, J_R are just edges sx and ty, respectively. The pinning functions are given by $\tau_L(x) = \alpha$, $\tau_R(y) = \beta$.

The next lemma follows straightforwardly from the definitions and guarantees that these gadgets satisfy items (iii) and (ii) of Definition 6 (note that (1) is a direct implication of (3)).

▶ **Lemma 11.** *For graph H, vertices $\alpha, \beta \in V(H)$, and $\Delta_1 = N_H(\alpha), \Delta_2 = N_H(\beta)$ the following hold*

(1) *if $\sigma \in \mathsf{Hom}(\mathcal{J}_L, H)$ then $\sigma(s) \in \Delta_1$, and if $\sigma \in \mathsf{Hom}(\mathcal{J}_R, H)$ then $\sigma(t) \in \Delta_2$,*

(2) *for any $\gamma_1 \in \Delta_1$ and $\gamma_2 \in \Delta_2$, it holds that $|\mathsf{Hom}((\mathcal{J}_L, s), (H, \gamma_1))| = |\mathsf{Hom}((\mathcal{J}_R, t), (H, \gamma_2))| = 1$,*

(3) *for any $\gamma_1' \notin \Delta_1$ and $\gamma_2' \notin \Delta_2$, it holds that $\mathsf{Hom}((\mathcal{J}_L, s), (H, \gamma_1')) = \mathsf{Hom}((\mathcal{J}_R, t), (H, \gamma_2')) = \emptyset$.*

The other type of a vertex gadget uses a cycle in H.

Let $C = \theta\gamma_1\gamma_2 \cdots \gamma_k\theta$ be a cycle in H of length at least three. Gadgets $\mathcal{J}_{CL} = (J_{CL}, \tau_{CL})$ and $\mathcal{J}_{CR} = (J_{CR}, \tau_{CR})$ are defined as follows

$$V(J_{CL}) = \{s\} \cup \{v_i, u_i : i \in [k]\} \cup \{x\},$$

$$E(J_{CL}) = \{v_iv_{i+1} : i \in [k-1]\} \cup \{v_iu_i : i \in [k]\} \cup \{sv_1, v_ks, sx\}.$$

The pinning function is given by $\tau(u_i) = \gamma_i$ for all $i \in [k]$ and $\tau(x) = \theta$.

The gadget $\mathcal{J}_{CR}$ is defined in the same way, except s is replaced with t.

▶ **Lemma 12.** *For a square-free graph H, a cycle $C = \theta\gamma_1\gamma_2\cdots\gamma_k\theta$ in H of length at least three, and $\Delta = \{\gamma_1, \gamma_k\}$ the following hold*

(1) *if $\sigma \in \mathsf{Hom}(\mathcal{J}_{CL}, H)$ or $\sigma \in \mathsf{Hom}(\mathcal{J}_{CR}, H)$, then $\sigma(s) \in \{\gamma_1, \gamma_k\}$ and $\sigma(t) \in \{\gamma_1, \gamma_k\}$, respectively;*

(2) *for any $\gamma \in \Delta$ it holds $|\mathsf{Hom}((J_{CL}, s), (H, \gamma))| = |\mathsf{Hom}((J_{CR}, t), (H, \gamma))| = 1$,*

(3) *for any $\gamma' \notin \Delta$ it holds $\mathsf{Hom}((\mathcal{J}_{CL}, s), (H, \gamma')) = \mathsf{Hom}((\mathcal{J}_{CR}, t), (H, \gamma')) = \emptyset$,*

6 The hardness of $\#_p$PartHom(H)

In this section we prove the hardness part of Theorem 1. More specifically we will apply Theorem 7 and the constructions from Section 5 to show that $\#_p BIS_{\lambda_1,\lambda_2}$ is Turing reducible to $\#_p\mathsf{PartHom}(H)$.

We consider three cases depending on the existence of vertices of certain degree in H. In each of the three cases we use slightly different variations of vertex and edge gadgets.

Case 1. The graph H has at least two vertices α and β such that $\mathsf{deg}(\alpha), \mathsf{deg}(\beta) \not\equiv 1 \pmod p$.

Let $S = \{\gamma \in V(H) : \mathsf{deg}(\gamma) \not\equiv 1 \mod p\}$; we know that S contains at least two elements. Pick $\alpha, \beta \in S$ such that the distance between them is minimal. Let $W = \alpha\gamma_1\cdots\gamma_{k-1}\gamma_k\beta$ be a shortest path between α, β. By the choice of W, $\mathsf{deg}(\gamma_i) \equiv 1 \pmod p$ for all $i \in [k]$.

We make an edge gadget $\mathcal{K} = (K, \tau)$ for this case based on this path as defined in Section 5.1. More precisely,

$$\begin{aligned} V(K) &= \{s, t\} \cup \{v_i, u_i : i \in [k]\}, \\ E(K) &= \{\{v_i, v_{i+1}\} : i \in [k-1]\} \cup \{\{v_i, u_i\} : i \in [k]\} \cup \{\{s, v_1\}, \{v_k, t\}\}. \end{aligned}$$

The labelling function is given by $\tau(u_i) = \gamma_i$ for all $i \in [k]$.

Any path is a nc-walk, so we can apply Lemma 10 to W. For the gadgets we use $\Delta_1 = N_H(\alpha), \Delta_2 = N_H(\beta)$ and $\delta_1 = \gamma_1, \delta_2 = \gamma_k$. This satisfies property (i) of hardness gadgets, because $\mathsf{deg}(\alpha), \mathsf{deg}(\beta) \not\equiv 1 \pmod p$. Then for any $\alpha_s \in \Delta_1 - \delta_1$ and $\alpha_t \in \Delta - \delta_2$ we have

(1) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \alpha_s, \alpha_t))| = 0$;

(2) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \gamma_1, \alpha_t))| \equiv 1 \pmod p$;

(3) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \alpha_s, \gamma_k))| \equiv 1 \pmod p$

Also, for any $i \in [k]$ we have $\mathsf{deg}(\gamma_i) \equiv 1 \pmod p$, and so

$$|\mathsf{Hom}((\mathcal{K}, s, t), (H, \gamma_1, \gamma_k))| = 1 + \sum_{i=1}^{k} (\mathsf{deg}(\gamma_i) - 1) = 1 + 0 \equiv 1 \pmod p.$$

Hence,

(4) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \gamma_1, \gamma_k))| \equiv 1 \pmod p$.

Thus $\mathcal{K}$ satisfies properties (iv), (v), (vi), and (vii) of hardness gadgets.

For a vertex gadget we use the first type, that is $\mathcal{J}_L, \mathcal{J}_R$ are just edges sx and ty, respectively, see Fig. 2. By Lemma 11 these gadgets satisfy properties (ii) and (iii) of hardness gadgets.

Thus Theorem 7 yields a required reduction.

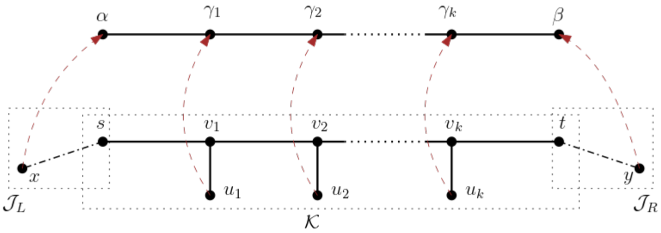


Figure 2 Vertex and edge gadgets based on the path $W = \alpha\gamma_1 \cdots \gamma_k\beta$. The vertex gadgets $\mathcal{J}_L$ and $\mathcal{J}_R$ are shown as dot-dashed boxes. The pinning function is shown by dashed lines.

Case 2. Graph H has exactly one vertex θ such that $\mathsf{deg}(\theta) \not\equiv 1 \pmod p$.

In this case we further split into two subcases. However, before we proceed with that we rule out the case of trees.

▶ **Lemma 13.** *Let H be a tree that has no automorphism of order p and is not a star. Then H has at least two vertices α and β such that $\mathsf{deg}(\alpha), \mathsf{deg}(\beta) \not\equiv 1 \pmod p$.*

Thus, we may assume that H is not a tree.

Case 2.1. The vertex θ, $\mathsf{deg}(\theta) \not\equiv 1 \pmod p$, is on a cycle C.

In this case the edge gadget is based on the cycle C. More precisely, let $C = \theta\gamma_1\gamma_2 \cdots \gamma_k\theta$ be a cycle in H of length at least 3 and such that for all $i \in [k]$ it holds that $\mathsf{deg}(v_i) \equiv 1 \pmod p$ and $\mathsf{deg}(\theta) \not\equiv 1 \pmod p$. We define gadget $\mathcal{K} = (K, \tau)$ as follows:

- $V(K) = \{s, t\} \cup \{v_i, u_i : i \in [k]\}$;
- $E(K) = \{v_iv_{i+1} : i \in [k-1]\} \cup \{v_iu_i : i \in [k]\} \cup \{sv_1, v_kt\}$;
- the labeling function is given by $\tau(u_i) = \gamma_i$ for all $i \in [k]$.

Set $\Delta_1 = \Delta_2 = N_H(\theta)$ and $\delta_1 = \gamma_1, \delta_2 = \gamma_k$. These parameters satisfy property (i) of a hardness gadget, because $\mathsf{deg}(\theta) \not\equiv 1 \pmod p$. A cycle is a nc-walk, so we can apply Lemma 10 to obtain the following

▶ **Lemma 14.** *Let H be a square-free graph and $\mathcal{K}$ an edge gadget based on the cycle $C = \theta\gamma_1\gamma_2 \cdots \gamma_k\theta$ in H. For any $\alpha_s \in \Delta_1 - \delta_1$ and $\alpha_t \in \Delta_2 - \delta_2$,*

(1) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \alpha_s, \alpha_t))| = 0$*;*
(2) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \delta_1, \alpha_t))| \equiv 1 \pmod p$*;*
(3) $|Hom((\mathcal{K}, s, t), (H, \alpha_s, \delta_2))| \equiv 1 \pmod p$*;*
(4) $|Hom((\mathcal{K}, s, t), (H, \delta_1, \delta_2))| \equiv 1 \pmod p$*.*

Proof. The cycle C is a nc-walk. Therefore by Lemma 10 items (1), (2), and (3) hold. For item (4) note that $\mathsf{deg}(\gamma_i) \equiv 1 \pmod p$ for all $i \in [k]$, therefore

$$|\mathsf{Hom}((\mathcal{K}, s, t), (H, \gamma_1, \gamma_k))| = 1 + \sum_{i=1}^{k}(\mathsf{deg}(\gamma_i) - 1) = 1 + 0 \equiv 1 \pmod p. \qquad ◀$$

By Lemma 14 gadget $\mathcal{K}$ satisfies properties (iv), (v), (vi), and (vii) of hardness gadgets.

For vertex gadgets we take $\mathcal{J}_L, \mathcal{J}_R$ (which are just edges) defined in Section 5.2, with $\alpha = \beta = \theta$, see Fig 3. By Lemma 11, these gadgets satisfy properties (ii) and (iii) of hardness gadgets.

Finally, by Theorem 7 $\#_pBIS_{\lambda_1,\lambda_2}$ is Turing reducible to $\#_p\mathsf{PartHom}(H)$.

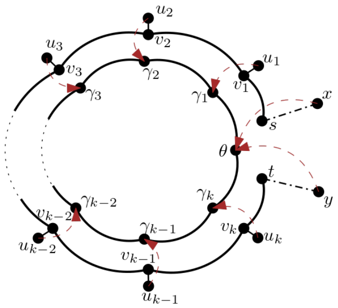


Figure 3 Hardness gadgets corresponding to a cycle $C = \theta\gamma_1 \cdots \gamma_k\theta$. The vertex gadgets $\mathcal{J}_L$ and $\mathcal{J}_R$ are shown by dot-dashed lines. The pinning function is shown by dashed lines.

Case 2.2. The vertex θ is not on any cycle.

Since H is not a tree, it contains at least one cycle; let C be such a cycle. Let $P = \gamma_0\gamma_{k+1}\gamma_{k+2}\cdots\gamma_{k+k'}\theta$ be a shortest path from a vertex γ on cycle $C = \gamma_0\gamma_1\gamma_2\cdots\gamma_k\gamma_0$ to θ. Note that $\mathsf{deg}(\gamma_i) \equiv 1 \pmod p$ for all γ_i, $i \in \{0, \dots, k+k'\}$. Edge gadget $\mathcal{K}$ in this case is based on the walk $W = \theta\gamma_{k+k'}\cdots\gamma_{k+2}\gamma_{k+1}\gamma_0\gamma_1\gamma_2\cdots\gamma_k\gamma_{k+1}\gamma_{k+2}\cdots\gamma_{k+k'}\theta$. Note that, W is an nc-walk. More precisely, the gadget $\mathcal{K} = (K, \tau)$ is defined as follows:

- $V(K) = \{s, t\} \cup \{v_i, u_i : i \in [k+2k'+2]\}$;
- $E(K) = \{v_iv_{i+1} : i \in [k+2k'+1]\} \cup \{v_iu_i : i \in [k+2k'+2]\} \cup \{sv_1, v_{k+2k'+1}t\}$;
- the pinning function is given by

$$\tau(u_i) = \begin{cases} \gamma_{k+k'+1-i} & 1 \le i \le k', \\ \gamma_{i-k'-1} & k'+1 \le i \le k+k'+1, \\ \gamma_0 & i = k+k'+2, \\ \gamma_{i-k'-2} & k+k'+3 \le i \le k+2k'+2. \end{cases}$$

Set $\delta_1 = \delta_2 = \gamma_{k+k'}$ and $\Delta_1 = \Delta_2 = N_H(\theta)$. These parameters satisfy property (i) of hardness gadgets, because $\mathsf{deg}(\theta) \not\equiv 1 \pmod p$. As W is an nc-walk, by Lemma 10 for any $\alpha \in N_H(\theta) - \gamma_{k+k'}$, we have

(1) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \alpha, \alpha))| = 0$;

(2) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \gamma_{k+k'}, \alpha))| \equiv 1 \pmod p$;

(3) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \alpha, \gamma_{k+k'}))| \equiv 1 \pmod p$.

Also, $\mathsf{deg}(\gamma_i) \equiv 1 \pmod p$ for all $i \in [k+k'] \cup \{0\}$. Therefore

$$|\mathsf{Hom}((\mathcal{K}, s, t), (H, \gamma_{k+k'}, \gamma_{k+k'}))| = 1 + \sum_{i=1}^{k+k'} (\mathsf{deg}(\gamma_i) - 1) = 1 + 0 \equiv 1 \pmod p.$$

Hence,

(4) $|\mathsf{Hom}((\mathcal{K}, s, t), (H, \gamma_{k+k'}, \gamma_{k+k'}))| \equiv 1 \pmod p$.

Thus the gadget $\mathcal{K}$ satisfies properties (iv), (v), (vi), and (vii) of hardness gadgets.

Finally, for vertex gadgets we again use gadgets $\mathcal{J}_L, \mathcal{J}_R$ introduced in Section 5.2, with $\alpha = \beta = \theta$, see Fig 4. By Lemma 11, these gadgets satisfy properties (ii) and (iii) of hardness gadgets. Thus, by Theorem 7 $\#_pBIS_{\lambda_1,\lambda_2}$, $\lambda_1 = \lambda_2 = |N_H(\theta)| - 1$ is Turing reducible to $\#_p\mathsf{PartHom}(H)$.

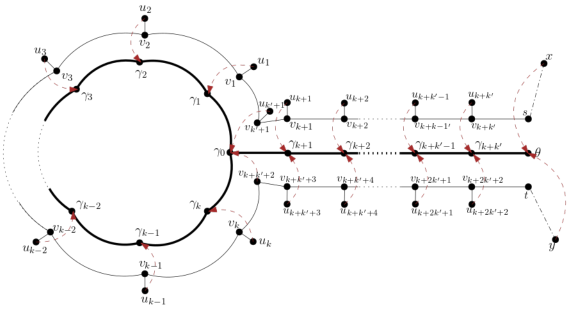


Figure 4 Gadget $\mathcal{K}$ based on the nc-walk $W = \theta\gamma_{k+k'}\cdots\gamma_{k+2}\gamma_{k+1}\gamma_0\gamma_1\gamma_2\cdots\gamma_k \gamma v_{k+1}\gamma_{k+2}\cdots\gamma_{k+k'}\theta$. The vertex gadgets $\mathcal{J}_L$ and $\mathcal{J}_R$ corresponding to θ are shown by dot-dashed lines. The pinning function is shown by dashed lines.

Case 3. For every vertex $\gamma \in V(H)$ it holds $\mathsf{deg}(\gamma) \equiv 1 \pmod{p}$.

By Lemma 13, H is not a tree, therefore it contains a cycle $C = \theta\gamma_1\gamma_2\cdots\gamma_k\theta$ such that $k \geq 3$. Set $\delta_1 = \gamma_1, \delta_2 = \gamma_k$ and $\Delta_1 = \Delta_2 = \{\gamma_1, \gamma_k\}$. These parameters satisfy property (i) of hardness gadget, because $|\Delta_1| = |\Delta_2| \not\equiv 1 \pmod{p}$. An edge gadget $\mathcal{K}$ is based on this cycle C. Since $\mathsf{deg}(\gamma) \equiv 1 \pmod{p}$ for every $\gamma \in V(H)$, as in Lemma 14 by Lemma 10 $\mathcal{K}$ satisfies properties (iv), (v), (vi), and (vii) of hardness gadgets.

For vertex gadgets we choose $\mathcal{J}_{CL}, \mathcal{J}_{CR}$ defined in Section 5.2, see Fig 5. By Lemma 12, these gadgets satisfy properties (ii) and (iii) of hardness gadgets. Therefore, by Theorem 7, $\#_pBIS_{\lambda_1,\lambda_2}$, $\lambda_1 = \lambda_2 = |\{\gamma_1, \gamma_k\}| - 1 = 1$ is Turing reducible to $\#_p\mathsf{PartHom}(H)$.

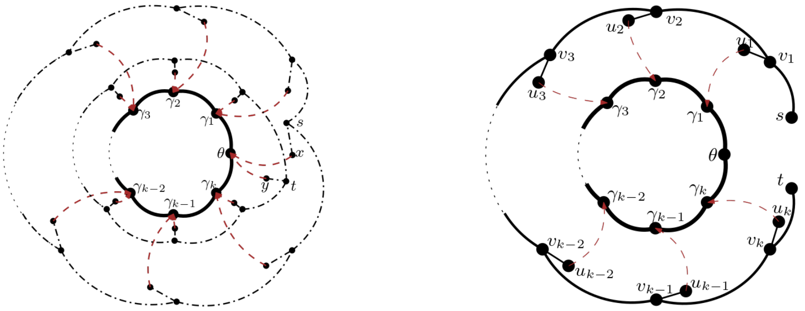


Figure 5 Hardness gadgets based on cycle $C = \theta\gamma_1\cdots\gamma_k\theta$.
On the left are the vertex gadgets $\mathcal{J}_{CL}$ and $\mathcal{J}_{CR}$ shown by dot-dashed lines. $\mathcal{J}_{CL}$ is the cycle containing vertex s, and $\mathcal{J}_{CR}$ is the cycle containing vertex t. The remaining vertices of the gadgets are not labelled. The pinning function is shown by dashed lines.
On the right, the edge gadget $\mathcal{K}$ is highlighted. Again, the pinning function is represented by dashed lines.

Note that the reduction in Case 3 can also be used to prove the result in Case 2.2, as it only depends on the existence of a cycle all of whose vertices have degrees $\equiv 1 \pmod{p}$. We, however, believe that the construction in Case 2.2 is simple, more transparent and deserves being considered.

References

1 Andrei A. Bulatov and Martin Grohe. The complexity of partition functions. *Theor. Comput. Sci.*, 348(2-3):148–186, 2005.

2 Jin-Yi Cai, Xi Chen, and Pinyan Lu. Graph Homomorphisms with Complex Values: A Dichotomy Theorem. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, pages 275–286, 2010.

3 Martin Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures and Algorithms*, 17(3–4):260–289, 2000.

4 John Faben and Mark Jerrum. The Complexity of Parity Graph Homomorphism: An Initial Investigation. *Theory of Computing*, 11:35–57, 2015.

5 Andreas Galanis, Leslie Ann Goldberg, and Mark Jerrum. Approximately Counting H-Colorings is #BIS-Hard. *SIAM J. Comput.*, 45(3):680–711, 2016.

6 Andreas Galanis, Leslie Ann Goldberg, and Mark Jerrum. A Complexity Trichotomy for Approximately Counting List H-Colorings. *TOCT*, 9(2):9:1–9:22, 2017.

7 Andreas Göbel, Leslie Ann Goldberg, and David Richerby. The Complexity of Counting Homomorphisms to Cactus Graphs Modulo 2. *ACM Trans. Comput. Theory*, 6(4):17:1–17:29, August 2014.

8 Andreas Göbel, Leslie Ann Goldberg, and David Richerby. Counting Homomorphisms to Square-Free Graphs, Modulo 2. *ACM Trans. Comput. Theory*, 8(3):12:1–12:29, May 2016.

9 Andreas Göbel, J. A. Gregor Lagodzinski, and Karen Seidel. Counting Homomorphisms to Trees Modulo a Prime. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, pages 49:1–49:13, 2018.

10 Leslie Ann Goldberg, Martin Grohe, Mark Jerrum, and Marc Thurley. A Complexity Dichotomy for Partition Functions with Mixed Signs. *SIAM J. Comput.*, 39(7):3336–3402, 2010.

11 Leslie Ann Goldberg and Heng Guo. The Complexity of Approximating complex-valued Ising and Tutte partition functions. *Computational Complexity*, 26(4):765–833, 2017.

12 Pavol Hell and Jaroslav Nešetřil. On the Complexity of H-coloring. *Journal of Combinatorial Theory, Ser.B*, 48:92–110, 1990.

13 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2004.

14 László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012.

15 Leslie G. Valiant. The Complexity of Computing the Permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

16 Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

17 Leslie G. Valiant. Accidental Algorthims. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, pages 509–517, 2006.

Approximate Counting CSP Seen from the Other Side

Andrei A. Bulatov
School of Computing Science, Simon Fraser University, Canada
abulatov@sfu.ca

Stanislav Živný
Department of Computer science, University of Oxford, UK
standa.zivny@cs.ox.ac.uk

Abstract

In this paper we study the complexity of counting Constraint Satisfaction Problems (CSPs) of the form $\#\mathrm{CSP}(\mathcal{C},-)$, in which the goal is, given a relational structure $\mathbf{A}$ from a class $\mathcal{C}$ of structures and an arbitrary structure $\mathbf{B}$, to find the number of homomorphisms from $\mathbf{A}$ to $\mathbf{B}$. Flum and Grohe showed that $\#\mathrm{CSP}(\mathcal{C},-)$ is solvable in polynomial time if $\mathcal{C}$ has bounded treewidth [FOCS'02]. Building on the work of Grohe [JACM'07] on decision CSPs, Dalmau and Jonsson then showed that, if $\mathcal{C}$ is a recursively enumerable class of relational structures of bounded arity, then assuming FPT $\neq$ #W[1], there are no other cases of $\#\mathrm{CSP}(\mathcal{C},-)$ solvable exactly in polynomial time (or even fixed-parameter time) [TCS'04].

We show that, assuming FPT $\neq$ W[1] (under randomised parametrised reductions) and for $\mathcal{C}$ satisfying certain general conditions, $\#\mathrm{CSP}(\mathcal{C},-)$ is not solvable even *approximately* for $\mathcal{C}$ of unbounded treewidth; that is, there is no fixed parameter tractable (and thus also not fully polynomial) randomised approximation scheme for $\#\mathrm{CSP}(\mathcal{C},-)$. In particular, our condition generalises the case when $\mathcal{C}$ is closed under taking minors.

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

Keywords and phrases constraint satisfaction, approximate counting, homomorphisms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.60

Funding 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.
Andrei A. Bulatov: Andrei Bulatov was supported by an NSERC Discovery grant.
Stanislav Živný: Stanislav Živný was supported by a Royal Society University Research Fellowship.

1 Introduction

The Constraint Satisfaction Problem (CSP) asks to decide the existence of a homomorphism between two given relational structures (or to find the number of such homomorphisms). It has been used to model a vast variety of combinatorial problems and has attracted much attention. Since the general CSP is NP-complete (#P-complete in the counting case) and because one needs to model specific computational problems, various restricted versions of the CSP have been considered. More precisely, let $\mathcal{C}$ and $\mathcal{D}$ be two classes of relational structures. In this paper we will assume that structures from $\mathcal{C}, \mathcal{D}$ only have predicate symbols of bounded arity. The *constraint satisfaction problem* (CSP) parametrised by $\mathcal{C}$ and $\mathcal{D}$ is the following computational problem, denoted by $\mathrm{CSP}(\mathcal{C},\mathcal{D})$: given $\mathbf{A} \in \mathcal{C}$ and $\mathbf{B} \in \mathcal{D}$, is there a homomorphism from $\mathbf{A}$ to $\mathbf{B}$? CSPs in which both input structures are restricted have not received much attention (with a notable exception of matrix partitions [19, 20] and assorted graph problems on restricted classes of graphs). However, the two most

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 60; pp. 60:1–60:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

natural restrictions have been intensively studied over the last two decades. Let $-$ denote the class of all (bounded-arity) relational structures, or, equivalently, indicate that there are no restrictions on the corresponding input structure.

Problems of the form $\mathrm{CSP}(-, \{\mathbf{B}\})$, where $\mathbf{B}$ is a fixed finite relational structure, are known as *nonuniform* or *language-restricted* CSPs [33]. For instance, if $\mathbf{B} = K_3$ is the complete graph on 3 vertices then $\mathrm{CSP}(-, \{\mathbf{B}\})$ is the standard 3-Colouring problem [27]. The study of nonuniform CSPs has been initiated by Schaefer [38] who considered the case of $\mathrm{CSP}(-, \{\mathbf{B}\})$ for 2-element structures $\mathbf{B}$. The complexity of $\mathrm{CSP}(-, \{\mathbf{H}\})$, for a fixed graph $\mathbf{H}$, was studied under the name of $\mathbf{H}$-colouring by Hell and Nešetřil [32]. General nonuniform CSPs have been studied extensively since the seminal paper of Feder and Vardi [21] who in particular proposed the so-called Dichotomy Conjecture stating that every nonuniform CSP is either solvable in polynomial time or is NP-complete. The complexity of nonuniform CSPs has been resolved only recently in two independent papers by Bulatov [3] and Zhuk [39], which confirmed the dichotomy conjecture of Feder and Vardi and also its algebraic version [4].

CSPs restricted on the other side, that is, of the form $\mathrm{CSP}(\mathcal{C}, -)$, where $\mathcal{C}$ is a fixed (infinite) class of finite relational structures, are known as *structurally-restricted* CSPs. For instance, if $\mathcal{C} = \cup_{k\geq 1}\{K_k\}$ is the class of cliques of all sizes then $\mathrm{CSP}(\mathcal{C}, -)$ is the standard Clique problem [27]. In this case the complexity of CSPs is related to various "width" parameters of the associated class of graphs. For a relational structure $\mathbf{A}$ let $G(\mathbf{A})$ denote the Gaifman graph of $\mathbf{A}$, that is, the graph whose vertices are the elements of $\mathbf{A}$, and vertices v, w are connected with an edge whenever v and w occur in the same tuple of some relation of $\mathbf{A}$. Then $G(\mathcal{C})$ denotes the class of Gaifman graphs of structures from $\mathcal{C}$, and we refer to the treewidth of $G(\mathbf{A})$ as the treewidth of $\mathbf{A}$. Dalmau, Kolaitis, and Vardi showed that $\mathrm{CSP}(\mathcal{C}, -)$ is in PTIME if $\mathcal{C}$ has bounded treewidth modulo homomorphic equivalence [10]. Grohe then showed that, assuming $\mathrm{FPT} \neq \mathrm{W}[1]$, there are no other cases of (bounded arity) $\mathrm{CSP}(\mathcal{C}, -)$ solvable in polynomial time (or even fixed-parameter time, where the parameter is the size of the left-hand side structure) [29]. The case of structures with unbounded arity was extensively studied by Gottlob et al. who introduced the concept of bounded hypertree width in an attempt to characterise structurally restricted CSPs solvable in polynomial time [28]. The search for a right condition is still going on, and the most general structural property that guarantees that $\mathrm{CSP}(\mathcal{C}, -)$ is solvable in polynomial time is fractional hypertree width introduced by Grohe and Marx [30].

An important problem related to the CSP is counting: Given a CSP instance, that is, two relational structures $\mathbf{A}$ and $\mathbf{B}$, find the number of homomorphisms from $\mathbf{A}$ to $\mathbf{B}$. We again consider restricted versions of this problem. More precisely, for two classes $\mathcal{C}$ and $\mathcal{D}$ of relational structures, $\#\mathrm{CSP}(\mathcal{C}, \mathcal{D})$ denotes the following computational problem: given $\mathbf{A} \in \mathcal{C}$ and $\mathbf{B} \in \mathcal{D}$, how many homomorphisms are there from $\mathbf{A}$ to $\mathbf{B}$? This problem is referred to as a *counting* CSP. Similar to decision CSPs, problems of the form $\#\mathrm{CSP}(-, \mathcal{D})$ and $\#\mathrm{CSP}(\mathcal{C}, -)$ are the two most studied ways to restrict the counting CSP, and the research on these problems follows a similar pattern as their decision counterparts.

For a fixed finite relational structure $\mathbf{B}$, the complexity of the nonuniform problem $\#\mathrm{CSP}(-, \{\mathbf{B}\})$ was characterised for graphs by Dyer and Greenhill [17] and for 2-element structures by Creignou and Hermann [8]. The complexity of the general nonuniform counting CSPs was resolved by Bulatov [5] and Dyer and Richerby [18]. As in the case of the decision version the complexity of nonuniform counting CSPs is determined by their algebraic properties, and every such CSP is either solvable in polynomial time or is #P-complete. These dichotomy results were later extended to the case of weighted counting CSP, for which Cai and Chen obtained a complexity classification of counting CSPs with complex weights [6].

The complexity of counting CSPs with restrictions on the left hand side structures also turns out to be related to treewidth. Flum and Grohe showed that $\#\mathrm{CSP}(\mathcal{C}, -)$ is solvable in polynomial time if $\mathcal{C}$ has bounded treewidth [22]. Dalmau and Jonsson then showed that, assuming FPT $\neq$ #W[1], there are no other cases of (bounded arity) $\#\mathrm{CSP}(\mathcal{C}, -)$ solvable exactly in polynomial time (or, again, even fixed-parameter time) [9]. Note that the result of Dalmau and Jonsson states that the class $\mathcal{C}$ itself has to be of bounded treewidth, while in Grohe's characterisation of polynomial-time solvable decision CSPs of the form $\mathrm{CSP}(\mathcal{C}, -)$ it is the class of cores of structures from $\mathcal{C}$ that has to have bounded treewidth. To the best of our knowledge there has been no research on counting problems over structures of unbounded arity except for the work of Brault-Baron et al., who showed that the (unbounded arity) structurally-restricted $\#\mathrm{CSP}(\mathcal{C}, -)$ are solvable in polynomial time for the class $\mathcal{C}$ of β-acyclic hypergraphs [2].[1]

The results we have mentioned so far concern exact counting; however, many applications of counting problems allow for approximation algorithms as well. For nonuniform CSPs the complexity landscape is much more complicated than the dichotomy results for decision CSPs or exact counting. The analogue of "easily solvable" problems in this case are those that admit a Fully Polynomial Randomised Approximation Scheme (FPRAS): a randomised algorithm that, given an instance and an error tolerance $\varepsilon \in (0, 1)$ returns in time polynomial in the size of the instance and ε^{-1} a result which is with high probability a multiplicative $(1 + \varepsilon)$-approximation of the exact solution. The parametrised version of this algorithmic model is known as a Fixed Parameter Tractable Randomised Approximation Scheme (FPTRAS). Beyond counting nonuniform CSPs, however, it was conjectured by Dyer et al. [15] that there is an infinite hierarchy of approximation complexities attainable by such problems. Only a handful of results exist for the approximation complexity of counting nonuniform CSPs. The approximation complexity of $\#\mathrm{CSP}(-, \{\mathbf{B}\})$ for 2-element structures $\mathbf{B}$ was characterised by Dyer at al. [16], where a trichotomy theorem was proved: for every 2-element structure $\mathbf{B}$ the problem $\#\mathrm{CSP}(-, \{\mathbf{B}\})$ either admits an FPRAS, or is interreducible with #SAT or with the problem #BIS of counting independent sets in bipartite graphs. Apart from this only partial results are known. If $\mathbf{B}$ is a connected graph and $\#\mathrm{CSP}(-, \{\mathbf{B}\})$ does not admit an FPRAS, then Galanis, Goldberg and Jerrum [25] showed that $\#\mathrm{CSP}(-, \{\mathbf{B}\})$ is at least as hard as #BIS. Also, if every unary relation is a part of $\mathbf{B}$ a complexity classification of $\#\mathrm{CSP}(-, \{\mathbf{B}\})$ can be extracted from the results of Chen et al. [7],[2] see also [26].

Our Contribution

It should be clear by now that the picture painted by the short survey above misses one piece: the approximation complexity of structurally restricted CSPs. This is the main contribution of this paper.

Let $\mathcal{C}$ be a class of bounded-arity relational structures. If the treewidth of $\mathcal{C}$ modulo homomorphic equivalence is unbounded then, by Grohe's result [29], it is hard to test for the existence of a homomorphism from $\mathbf{A}$ to $\mathbf{B}$, where $\mathbf{A} \in \mathcal{C}$, for any instance $\mathbf{A}, \mathbf{B}$ of $\mathrm{CSP}(\mathcal{C}, -)$. Using standard techniques (see, e.g., the proof of [34, Proposition 3.16]), this implies, assuming that FPT $\neq$ #W[1] (under randomised parametrised reductions [14]), that

[1] Brault-Baron et al. [2] show their tractability results for so-called CSPs with default values, which in particular includes $\#\mathrm{CSP}(\mathcal{C}, -)$ as defined here.

[2] Chen et al. [7] studied the weighted version of $\#\mathrm{CSP}(-, \{\mathbf{B}\})$, and although their result does not provide a complete characterisation of the weighted problem, it allows to determine the complexity of $\#\mathrm{CSP}(-, \{\mathbf{B}\})$ as defined here.

there is not an FPTRAS for $\#\mathrm{CSP}(\mathcal{C}, -)$, let alone an FPRAS. Consequently, the tractability boundary for approximate counting of $\#\mathrm{CSP}(\mathcal{C}, -)$ lies between bounded treewidth and bounded treewidth modulo homomorphic equivalence.

As our main result, we show that for $\mathcal{C}$ such that a certain class of graphs (to be defined later) is a subset of $G(\mathcal{C})$, $\#\mathrm{CSP}(\mathcal{C}, -)$ cannot be solved even *approximately* for $\mathcal{C}$ of unbounded treewidth, assuming $\mathrm{FPT} \neq \mathrm{W}[1]$ (under randomised parametrised reductions). Before we introduce the classes of graphs we use, we review how the hardness of $\mathrm{CSP}(\mathcal{C}, -)$ or $\#\mathrm{CSP}(\mathcal{C}, -)$ is usually proved.

We follow the hardness proof of Grohe for decision CSPs [29], which was lifted to exact counting CSPs by Dalmau and Jonsson [9]. In fact Grohe's result had an important precursor [31]. The key idea is a reduction from the parametrised Clique problem to $\mathrm{CSP}(\mathcal{C}, -)$. Let $G = (V, E)$ and k be an instance of the p-Clique problem, where k is the parameter. Broadly speaking, the reduction works as follows. For a class of unbounded treewidth, the Excluded Grid Theorem of Robertson and Seymour [37] guarantees the existence of the $(k \times \binom{k}{2})$-grid (as a minor of some structure $\mathbf{A} \in \mathcal{C}$), which is used to encode the existence of a k-clique in G as a certain structure $\mathbf{B}$. The encoding usually means that G has a k-clique if and only if there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$ whose image covers a copy of the grid built in $\mathbf{B}$. For decision CSPs, the correctness of the reduction – that there are no homomorphisms from $\mathbf{A}$ to $\mathbf{B}$ not satisfying this condition – is achieved by dealing with coloured grids [31] or by dealing with structures whose cores have unbounded treewidth (with another complication caused by minor maps) [29]. For the complexity of exact counting CSPs, the correctness of the reduction [9] is achieved by employing interpolation or the inclusion-exclusion principle, a common tool in exact counting.

None of these two methods can be applied to approximate solving $\#\mathrm{CSP}(\mathcal{C}, -)$. We cannot assume that the class of cores of $\mathcal{C}$ has unbounded treewidth, because then by [29] even the decision problem cannot be solved in polynomial time, which immediately rules out the existence of an FPRAS. Interpolation techniques such as the inclusion-exclusion principle are also well known to be incompatible with approximate counting. The standard tool in approximate counting to achieve the same goal of prohibiting homomorphisms except ones from a certain restricted type, is to use gadgets to amplify the number of homomorphisms of the required type. We give a reduction from p-#Clique to $\#\mathrm{CSP}(\mathcal{C}, -)$ by using "fan-grids", formally introduced in Section 3.3. Unfortunately, due to the delicate nature of approximation preserving reductions, we cannot use minors and minor maps and have to assume that "fan-grids" themselves are present in $G(\mathcal{C})$. (In Section 5, we will briefly discuss how a weaker assumption can be used to obtain the same result.) By the Excluded Grid Theorem [37], if $\mathcal{C}$ is *closed under taking minors*, then $G(\mathcal{C})$ contains all the fan-grids (details are given in Section 3.3 and in particular in Lemma 4). Thus, the classes $\mathcal{C}$ for which we establish the hardness of $\#\mathrm{CSP}(\mathcal{C}, -)$ includes the classes $\mathcal{C}$ that are closed under taking minors.[3]

[3] We remark that the hardness for $\mathcal{C}$ closed under taking minors follows from Grohe's classification [29] of decision CSPs. Indeed, for $\mathcal{C}$ of unbounded treewidth, the Excluded Grid Theorem [37] gives grids of arbitrary sizes. Since every planar graph is a minor of some grid [11], $\mathcal{C}$ contains all planar graphs. As there exist graphs of arbitrary large treewidth that are also minimal with respect to homomorphic equivalence, Grohe's result gives W[1]-hardness of $\mathrm{CSP}(\mathcal{C}, -)$ and hence $\#\mathrm{CSP}(\mathcal{C}, -)$ cannot have an FPRAS/FPTRAS.

2 Preliminaries

$\mathbb{N}$ denotes the set of positive integers. For every $n \in \mathbb{N}$, we let $[n] = \{1, \dots, n\}$.

2.1 Relational Structures and Homomorphisms

A *relational signature* is a finite set τ of relation symbols R, each with a specified arity $\mathrm{ar}(R)$. A *relational structure* $\mathbf{A}$ over a relational signature τ (or a τ-structure, for short) is a finite universe A together with one relation $R^{\mathbf{A}} \subseteq A^{\mathrm{ar}(R)}$ for each symbol $R \in \tau$. The size $\|\mathbf{A}\|$ of a relational structure $\mathbf{A}$ is defined as

$$\|\mathbf{A}\| = |\tau| + |A| + \sum_{R\in\tau} |R^{\mathbf{A}}| \cdot \mathrm{ar}(R).$$

Let R be a binary relational symbol. We will sometimes view graphs as $\{R\}$-structures.

A *homomorphism* from a relational τ-structure $\mathbf{A}$ (with universe A) to a relational τ-structure $\mathbf{B}$ (with universe B) is a mapping $\varphi : A \to B$ such that for all $R \in \tau$ and all tuples $\mathbf{x} \in R^{\mathbf{A}}$ we have $\varphi(\mathbf{x}) \in R^{\mathbf{B}}$.

Two structures $\mathbf{A}$ and $\mathbf{B}$ are *homomorphically equivalent* if there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$ and a homomorphism from $\mathbf{B}$ to $\mathbf{A}$.

Let $\mathcal{C}$ be a class of relational structures. We say that $\mathcal{C}$ has *bounded arity* if there is a constant $r \geq 1$ such that for every τ-structure $\mathbf{A} \in \mathcal{C}$ and $R \in \tau$, we have that $\mathrm{ar}(R) \leq r$.

2.2 Treewidth and Minors

The notion of treewidth, introduced by Robertson and Seymour [36], is a well-known measure of the tree-likeness of a graph [11]. Let $G = (V(G), E(G))$ be a graph. A *tree decomposition* of G is a pair (T, β) where $T = (V(T), E(T))$ is a tree and β is a function that maps each node $t \in V(T)$ to a subset of $V(G)$ such that

1. $V(G) = \bigcup_{t\in V(T)} \beta(t)$,
2. for every $u \in V(G)$, the set $\{t \in V(T) \mid u \in \beta(t)\}$ induces a connected subgraph of T, and
3. for every edge $\{u, v\} \in E(G)$, there is a node $t \in V(T)$ with $\{u, v\} \subseteq \beta(t)$.

The *width* of the decomposition (T, β) is $\max\{|\beta(t)| \mid t \in V(T)\} - 1$. The *treewidth* $tw(G)$ of a graph G is the minimum width over all its tree decompositions.

Let $\mathbf{A}$ be a relational structure over relational signature τ. The *Gaifman graph* (also known as *primal graph*) of $\mathbf{A}$, denoted by $G(\mathbf{A})$, is the graph whose vertex set is the universe of $\mathbf{A}$ and whose edges are the pairs (u, v) for which there is a tuple $\mathbf{x}$ and a relation symbol $R \in \tau$ such that u, v appear in $\mathbf{x}$ and $\mathbf{x} \in R^{\mathbf{A}}$.

Let $\mathcal{C}$ be a class of relational structures. We say that $\mathcal{C}$ has *bounded treewidth* if there exists $w \geq 1$ such that $tw(\mathbf{A}) = tw(G(\mathbf{A})) \leq w$ for every $\mathbf{A} \in \mathcal{C}$. We say that $\mathcal{C}$ has *bounded treewidth modulo homomorphic equivalence* if there exists $w \geq 1$ such that every $\mathbf{A} \in \mathcal{C}$ is homomorphically equivalent to $\mathbf{A}'$ with $tw(\mathbf{A}') \leq w$.

A graph H is a *minor* of a graph G if H is isomorphic to a graph that can be obtained from a subgraph of G by contracting edges (for more details, see, e.g., [11]).

For $k, \ell \geq 1$, the $(k \times \ell)$-grid is the graph with the vertex set $[k] \times [\ell]$ and an edge between (i, j) and (i', j') iff $|i - i'| + |j - j'| = 1$. Treewidth and minors are intimately connected via the celebrated Excluded Grid Theorem of Robertson and Seymour.

▶ **Theorem 1** ([37]). *For every k there exists a $w(k)$ such that the $(k \times k)$-grid is a minor of every graph of treewidth at least $w(k)$.*

Let $\mathcal{C}$ be a class of relational structures. We say that $\mathcal{C}$ if *closed under taking minors* if for every $\mathbf{A} \in \mathcal{C}$ and for every minor H of $G(\mathbf{A})$, there is a structure $\mathbf{A}' \in C$ such that $G(\mathbf{A}')$ is isomorphic to H.

3 Counting CSP

3.1 Exact Counting CSP

Let $\mathcal{C}$ be a class of relational structures. We will be interested in the computational complexity of the following problem.

Name: $\#\mathrm{CSP}(\mathcal{C}, -)$
Input: Two relational structures $\mathbf{A}$ and $\mathbf{B}$ over the same signature with $\mathbf{A} \in \mathcal{C}$.
Output: The number of homomorphisms from $\mathbf{A}$ to $\mathbf{B}$.

We say that $\#\mathrm{CSP}(\mathcal{C}, -)$ is in FP, the class of function problems solvable in *polynomial time*, if there is a deterministic algorithm that solves any instance $\mathbf{A}, \mathbf{B}$ of $\#\mathrm{CSP}(\mathcal{C}, -)$ in time $(\|\mathbf{A}\| + \|\mathbf{B}\|)^{O(1)}$.

We will also consider the parametrised version of $\#\mathrm{CSP}(\mathcal{C}, -)$.

Name: p-$\#\mathrm{CSP}(\mathcal{C}, -)$
Input: Two relational structures $\mathbf{A}$ and $\mathbf{B}$ over the same signature with $\mathbf{A} \in \mathcal{C}$.
Parameter: $\|\mathbf{A}\|$.
Output: The number of homomorphisms from $\mathbf{A}$ to $\mathbf{B}$.

We say that p-$\#\mathrm{CSP}(\mathcal{C}, -)$ is in FPT, the class of problems that are *fixed-parameter tractable*, if there is a deterministic algorithm that solves any instance $\mathbf{A}, \mathbf{B}$ of p-$\#\mathrm{CSP}(\mathcal{C}, -)$ in time $f(\|\mathbf{A}\|) \cdot \|\mathbf{B}\|^{O(1)}$, where $f : \mathbb{N} \to \mathbb{N}$ is an arbitrary computable function.

The class W[1], introduced in [12], can be seen as an analogue of NP in parameterised complexity theory. Proving W[1]-hardness of a problem (under a parametrised reduction which may be randomised), is a strong indication that the problem is not solvable in fixed-parameter time as it is believed that FPT $\neq$ W[1]. For counting problems, #W[1] is the parametrised analogue of #P. Similarly to the belief that FP $\neq$ #P, it is believed that FPT $\neq$ #W[1]. We refer the reader to [24] for the definitions of W[1] and #W[1], and for more details on parameterised complexity in general.

Dalmau and Jonsson established the following result.

▶ **Theorem 2** ([9]). *Assume FPT $\neq$ #W[1] under parametrised reductions. Let $\mathcal{C}$ be a recursively enumerable class of relational structures of bounded arity. Then, the following are equivalent:*

1. $\#\mathrm{CSP}(\mathcal{C}, -)$ *is in FP.*
2. p-$\#\mathrm{CSP}(\mathcal{C}, -)$ *is in FPT.*
3. $\mathcal{C}$ *has bounded treewidth.*

The following problem is an example of a #W[1]-hard problem, as established by Flum and Grohe [23].

Name: p-#Clique
Input: A graph G and $k \in \mathbb{N}$.
Parameter: k.
Output: The number of cliques of size k in G.

Note that p-#Clique can be modelled as p-#CSP$(\mathcal{C}, -)$ if we set $\mathcal{C}$ to be the set of cliques of all possible sizes. The decision version of p-#Clique was shown to be W[1]-hard by Downey and Fellows [13].

Name: p-Clique
Input: A graph G and $k \in \mathbb{N}$.
Parameter: k.
Output: Decide if G contains a clique of size k.

3.2 Approximate Counting CSP

In view of our complete understanding of the exact complexity of #CSP$(\mathcal{C}, -)$ for $\mathcal{C}$ of bounded arity (cf. Theorem 2), we will be interested in *approximation* algorithms for #CSP$(\mathcal{C}, -)$. In particular, are there any new classes $\mathcal{C}$ of bounded arity for which the problem #CSP$(\mathcal{C}, -)$ can be solved efficiently (if only approximately)? We will provide a partial answer to this question (cf. Theorem 3): for certain general bounded-arity classes $\mathcal{C}$ (which include classes that are *closed under taking minors*), the answer is no!

The notion of efficiency for approximate counting is that of a fully polynomial randomised approximation scheme [35] and its parametrised analogue, a fixed parameter tractable randomised approximation scheme, originally introduced by Arvind and Raman [1]. We now define both concepts.

A *randomised approximation scheme* (RAS) for a function $f : \Sigma^* \to \mathbb{N}$ is a randomised algorithm that takes as input $(x, \varepsilon) \in \Sigma^* \times (0, 1)$ and produces as output an integer random variable X satisfying the condition $\Pr(|X - f(x)| \leq \varepsilon f(x)) \geq 3/4$. A RAS for a counting problem is called *fully polynomial* (FPRAS) if on input of size n it runs in time $p(n, \varepsilon^{-1})$ for some fixed polynomial p. A RAS for a parametrised counting problem is called *fixed parameter tractable* (FPTRAS) if on input of size n with parameter k it runs in time $f(k) \cdot p(n, \varepsilon^{-1})$, where p is a fixed polynomial and f is an arbitrary computable function.

To compare approximation complexity of (parametrised) counting problems two types of reductions are used. Suppose $f, g : \Sigma^* \to \mathbb{N}$. An *approximation preserving reduction* (AP-reduction) [15] from f to g is a probabilistic oracle Turing machine M that takes as input a pair $(x, \varepsilon) \in \Sigma^* \times (0, 1)$, and satisfies the following three conditions: (i) every oracle call made by M is of the form (w, δ), where $w \in \Sigma^*$ is an instance of g, and $0 < \delta < 1$ is an error bound satisfying $\delta^{-1} \leq \mathsf{poly}(|x|, \varepsilon^{-1})$; (ii) the TM M meets the specification for being a randomised approximation scheme for f whenever the oracle meets the specification for being a randomised approximation scheme for g; and (iii) the running time of M is polynomial in $|x|$ and ε^{-1}.

Similar to [34] we also use the parametrised version of AP-reductions. Again, let $f, g : \Sigma^* \to \mathbb{N}$. A *parametrised approximation preserving reduction* (parametrised AP-reduction) from f to g is a probabilistic oracle Turing machine M that takes as input a triple $(x, k, \varepsilon) \in \Sigma^* \times (0, 1)$, and satisfies the following three conditions: (i) every oracle call made by M is of the form (w, k', δ), where $w \in \Sigma^*$ is an instance of g, $k' \leq h(k)$ for some computable function h, and $0 < \delta < 1$ is an error bound satisfying $\delta^{-1} \leq \mathsf{poly}(|x|, \varepsilon^{-1})$; (ii) the TM M meets the specification for being a randomised approximation scheme for f whenever the oracle meets the specification for being a randomised approximation scheme for g; and (iii) M is fixed-parameter tractable with respect to k and polynomial in $|x|$ and ε^{-1}.

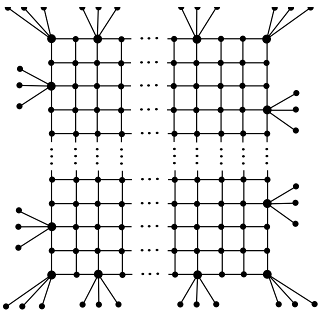


Figure 1 Fan-grid. Fan vertices are shown by larger dots.

3.3 Main Result

The following concept plays a key role in this paper. Let $k, r, \ell_1, \ell_2 \in \mathbb{N}$. The *fan-grid* $L(k, r, \ell_1, \ell_2)$ is a graph with vertex set $L_1 \cup L_2$, where $L_1 = \{(i, p) \mid i \in [k], p \in [r]\}$, $L_2 = M_1 \cup \cdots \cup M_{12}$, where $M_1, \ldots, M_{12}$ are disjoint and $|M_i| = \ell_1$ for $i \in [4]$, and $|M_i| = \ell_2$ for $i \in \{5, \ldots, 12\}$. Vertices from L_1 will be called *grid vertices*. Vertices $u_1 = (1, 1)$, $u_2 = (1, r)$, $u_3 = (k, 1)$, $u_4 = (k, r)$, $u_5 = (1, 3)$, $u_6 = (1, r - 3)$, $u_7 = (k, 3)$, $u_8 = (k, r - 3)$, $u_9 = (3, 1)$, $u_{10} = (4, r)$, $u_{11} = (k - 2, 1)$, $u_{12} = (k - 3, r)$ will be called *fan vertices*, and u_1, u_2, u_3, u_4 will be called *corner vertices*. The edges of the fan grid are as follows: $(i, p)(i', p')$ for $|i - i'| + |p - p'| = 1$, and wu_i for each $w \in M_i$ and $i \in [12]$, see Figure 1.

We call a class $\mathcal{C}$ of relational structures of bounded arity a *fan class* if either $\mathcal{C}$ has bounded treewidth or for any parameters $k, r, \ell_1, \ell_2 \in \mathbb{N}$ we have that $G(\mathcal{C})$ contains the fan-grid $L(k, r, \ell_1, \ell_2)$.

The following is our main result.

▶ **Theorem 3 (Main).** *Assume FPT $\neq$ W[1] under randomised parametrised reductions. Let $\mathcal{C}$ be a recursively enumerable class of relational structures of bounded arity. If $\mathcal{C}$ is a fan class then the following are equivalent:*

1. *$\#\text{CSP}(\mathcal{C}, -)$ is polynomial time solvable.*
2. *$\#\text{CSP}(\mathcal{C}, -)$ admits an FPRAS.*
3. *p-$\#\text{CSP}(\mathcal{C}, -)$ admits an FPTRAS.*
4. *$\mathcal{C}$ has bounded treewidth.*

Let $\mathcal{C}$ be a recursively enumerable class of relational structures of bounded arity and closed under taking minors. We claim that $\mathcal{C}$ is a fan class and thus Theorem 3 applies to such $\mathcal{C}$. For this we need Theorem 1. In particular, for any $k, r, \ell_1, \ell_2 \in \mathbb{N}$, if $\mathcal{C}$ is not of bounded treewidth then, by Theorem 1, $G(\mathcal{C})$ contains an $(s \times s)$-grid, where $s = \max(k + 2\ell_1, r + 2\ell_2)$, and thus also a $(k + 2\ell_1) \times (r + 2\ell_2)$-grid. The following simple lemma then shows that fan-grids are minors of grids (of appropriate size).

▶ **Lemma 4.** *$L(k, r, \ell_1, \ell_2)$ is a minor of $(t \times t')$-grid, where $t = k + 2\ell_1$, $t' = r + 2\ell_2$.*

4 Proof of Theorem 3

Conditions (1) and (4) in Theorem 3 are equivalent by [9]. Implications "(1) ⇒ (2) ⇒ (3)" are obvious; implication "(4) ⇒ (1)" is by the standard treewidth-based dynamic programming for exact counting. Our main contribution is to prove the "(3) ⇒ (4)" implication.

4.1 Construction

Let $G = (V, E)$ be a graph with $n = |V|$ and $m = |E|$. Let $k \in \mathbb{N}$. We construct a graph $H(G, k, W_1, W_2)$ for $W_1, W_2 > 2(n+m)$ as follows. Let $r = \binom{k}{2}$ and let ϱ be a correspondence between $[r]$ and the set of 2-element sets $\{\{i, j\} \mid i, j \in [k], i \neq j\}$. For $i \in [k]$ and $p \in [r]$, we write $i \in p$ rather than $i \in \varrho(p)$. The vertex set of $H(G, k, W_1, W_2)$ is the union of two sets $H_1 \cup H_2$, defined by

$$H_1 = \{(v, e, i, p) \mid v \in V, e \in E, \text{ and } v \in e \iff i \in p\},$$
$$H_2 = K_1 \cup \cdots \cup K_{12},$$

where $K_1, \ldots, K_{12}$ are disjoint and $|K_i| = W_1$ for $i \in [4]$, $|K_i| = W_2$ for $i \in \{5, \ldots, 12\}$.

As in fan-grids, vertices of the form $(v, e, 1, 1)$, $(v, e, 1, r)$, $(v, e, k, 1)$, (v, e, k, r), $(v, e, 1, 3)$, $(v, e, 1, r-3)$, $(v, e, k, 3)$, $(v, e, k, r-3)$, $(v, e, 3, 1)$, $(v, e, 4, r)$, $(v, e, k-2, 1)$, $(v, e, k-3, r)$ will be called *fan vertices*, and vertices of the form $(v, e, 1, 1)$, $(v, e, 1, r)$, $(v, e, k, 1)$, (v, e, k, r) will be called *corner vertices*.

The edge set of $H(G, k, W_1, W_2)$ consists of the following pairs:

- $(v, e, i, p)(v', e, i', p)$ such that $|i - i'| = 1$;
- $(v, e, i, p)(v, e', i, p')$ such that $|p - p'| = 1$;
- $u(v, e, 1, 1)$ for $u \in S_1 \subseteq K_1$ and $(v, e, 1, 1) \in H_1$, where S_1 is an arbitrary subset of K_1 whose cardinality is such that the degree of $(v, e, 1, 1)$ is exactly W_1;
  similarly, $u(v, e, 1, r)$, $u(v, e, k, 1)$, $u(v, e, k, r)$, $u(v, e, 1, 3)$, $u(v, e, 1, r-3)$, $u(v, e, k, 3)$, $u(v, e, k, r-3)$, $u(v, e, 3, 1)$, $u(v, e, 4, r)$, $u(v, e, k-2, 1)$, $u(v, e, k-3, r)$ for $u \in S_j \subseteq K_j$ (for $j = 2, \ldots, 12$ in this order) and $(v, e, 1, r)$, $(v, e, k, 1)$, (v, e, k, r), $(v, e, 1, 3)$, $(v, e, 1, r-3)$, $(v, e, k, 3)$, $(v, e, k, r-3)$, $(v, e, 3, 1)$, $(v, e, 4, r)$, $(v, e, k-2, 1)$, $(v, e, k-3, r) \in H_1$, where $S_2, \ldots, S_{12}$ are arbitrary subsets whose cardinality is such that the degree of $(v, e, 1, r), (v, e, k, 1), (v, e, k, r)$ is exactly W_1 and the degree of the remaining vertices from the list is exactly W_2.

Note that the sizes of sets $K_1, \ldots, K_{12}$ are chosen in such a way that all the corner vertices have degree W_1 and the remaining fan vertices have degree W_2.

We study homomorphisms from $L(k, r, \ell_1, \ell_2)$ to $H(G, k, W_1, W_2)$. A homomorphism $\varphi : L(k, r, \ell_1, \ell_2) \to H(G, k, W_1, W_2)$ is said to be corner-to-corner (or c-c for short) if

$$\varphi(1, 1), \varphi(1, r), \varphi(k, 1), \varphi(k, r) \in \{(v, e, 1, 1), (v, e, 1, r), (v, e, k, 1), (v, e, k, r) \mid v \in V, e \in E\}.$$

Homomorphism φ is called identity (skew identity) if $\varphi(i, p) \in \{(v, e, i, p) \mid v \in V, e \in E\}$ (respectively, $\varphi(i, p) \in \{(v, e, k-i+1, p) \mid v \in V, e \in E\}$) for all $i \in [k]$ and $p \in [r]$.

We define the *weight* of a homomorphism φ from $L(k, r, \ell_1, \ell_2)$ restricted to L_1 (the set of grid vertices) to $H(G, k, W_1, W_2)$ as the number of extensions of φ to a homomorphism from $L(k, r, \ell_1, \ell_2)$.

4.2 Weights of Homomorphisms

We start with a simple lemma.

▶ **Lemma 5.** *The weight of an identity or skew identity homomorphism is* $W_1^{4\ell_1} W_2^{8\ell_2}$.

Proof. The images of grid vertices (the set L_1) under identity and skew identity homomorphisms are fixed, while vertices from L_2 can be mapped to any neighbour of the corresponding fan vertex independently. Since the degree of a corner vertex (v, e, i, p) with $i \in \{1, k\}$ and $p \in \{1, r\}$ is W_1, and the degree of any other fan vertex is W_2, the result follows. ◀

The next lemma, which can proved using Lemma 5, is essentially [9, Lemma 3.1] adapted to our setting, which in turn builds on [29, Lemma 4.4].

▶ **Lemma 6.** *Let N be the number of k-cliques in G. Then there are $2NW_1^{4\ell_1}W_2^{8\ell_2}k!$ identity and skew identity homomorphisms.*

Next we establish an upper bound on the total weight of homomorphisms that are neither identity nor skew identity.

▶ **Lemma 7.** *Let $G = (V, E)$ has $n = |V|$ vertices and $m = |E|$ edges, let $k = 4k'$ for some k', and let $T = \log_{W_2} W_1$. If*

$$\ell_1 > \frac{8T\ell_2}{T-1},$$

then the total weight of homomorphisms that are neither identity nor skew identity is at most

$$W_1^{4\ell_1}W_2^{6\ell_2}(2n+m)^{2\ell_2} \cdot (4W_1 + 8W_2 + nmkr)^{kr}.$$

The key ideas in the proof of Lemma 7 are the following: Firstly, we show that c-c homomorphisms dominate non-c-c homomorphisms. Secondly, using crucially the special structure of fan grids and our choice of k being a multiple of four, we establish an upper bound on any c-c homomorphism that is neither identity nor skew identity. Finally, we give an upper bound on the number of all homomorphisms. These three ingredients together allows us to establish the required bound.

We now have all results required to relate the number of k-cliques in a given graph G and the number of homomorphisms from $L(k, r, \ell_1, \ell_2)$ to $H(G, K, W_1, W_2)$, for appropriately chosen values of ℓ_1, ℓ_2, W_1, W_2.

▶ **Lemma 8.** *Let $N \geq 0$ be the number of k-cliques in G, where $k = 4k'$ for some k', $n = V(G)$, $m = E(G)$, and $2n + m > 6$. Let $M = M(\ell_1, \ell_2, W_1, W_2)$ be the number of homomorphisms from $L(k, r, \ell_1, \ell_2)$, $r = \binom{k}{2}$, to $H(G, k, W_1, W_2)$. If $W_2 = (2n + m)^2$, $W_1 = W_2^2$, $\ell_2 = 8kr$, and $\ell_1 = 17\ell_2$, then we have*

$$N < \frac{M}{2W_1^{4\ell_1}W_2^{8\ell_2}k!} < N + \frac{1}{2}.$$

Finally, as Lemmas 7 and 8 are only proved for $k = 4k'$, we need to show that the problem for other values of the parameter can be reduced to k of such form. The following lemma takes care of that. Let $4p$-#Clique denote the following problem

Name: $4p$-#Clique
Input: A graph G and $k \in \mathbb{N}$.
Parameter: k.
Output: The number of cliques of size $4k$ in G.

▶ **Lemma 9.** *There is a parametrised AP-reduction from p-#Clique to $4p$-#Clique.*

In particular, Lemma 9 establishes #W[1]-hardness of the $4p$-#Clique problem.

4.3 Putting the Pieces Together

Proof of Theorem 3. As we mentioned earlier, conditions (1) and (4) are equivalent, the implications "(1) ⇒ (2) ⇒ (3)" are trivial and the implication "(4) ⇒ (1)" is known: if $\mathcal{C}$ has bounded treewidth then, by [22, Proposition 7], #CSP($\mathcal{C}, -$) belongs to FP.

The rest of the proof establishes "(3) ⇒ (4)". Assume that $\#\mathrm{CSP}(\mathcal{C},-)$ admits an FPTRAS for a fan class $\mathcal{C}$. Our goal is to show that $\mathcal{C}$ has bounded treewidth. For the sake of contradiction, assume that $\mathcal{C}$ has unbounded treewidth. We will exhibit a parametrised reduction from p-#Clique to p-$\#\mathrm{CSP}(\mathcal{C},-)$, which gives an FPTRAS for p-#Clique assuming an FPTRAS for p-$\#\mathrm{CSP}(\mathcal{C},-)$. Under the assumption that $\mathrm{FPT} \neq \mathrm{W}[1]$ (under randomised parametrised reductions [14]), the W[1]-hardness of p-Clique established in [13] implies, by [34, Corollary 3.17], the non-existence of an FPTRAS for the p-#Clique problem, a contradiction.

Let $G = (V, E)$ and k be an instance of the p-#Clique problem. By Lemma 9, we can assume that $k = 4k'$. First, we show that if G has any k-cliques at all, it can be assumed to have many k-cliques. Let $s \in \mathbb{N}$ and G_s is defined as follows. $V(G_s) = \{v_1, \ldots, v_s \mid v \in V\}$ and $v_i w_j \in E(G_s)$, for $v, w \in V$ and $i, j \in [s]$, if and only if $vw \in E$. In other words, every vertex v of G is replaced with s distinct vertices $v_1, \ldots, v_s$, and every edge vw is replaced with a complete bipartite graph $K_{s,s}$.

▷ Claim 1. If N is the number of k-cliques in G, then G_s contains $s^k N$ k-cliques.

Proof of Claim 1. As is easily seen, for any indices $i_1, \ldots, i_k \in [s]$ the vertices $v^1_{i_1}, \ldots, v^k_{i_k}$ induce a clique in G_s if and only if $v^1, \ldots, v^k$ is a clique in G. Moreover, no clique in G_s contains vertices v_i, v_j for $v \in V$ and $i, j \in [s]$. The result follows. ◁

For a given instance $G = (V, E)$, k of p-#Clique and error tolerance $\varepsilon \in (0, 1)$ using Claim 1, we first reduce it to the instance G_s, k of p-#Clique, where

$$s > \left(\frac{1 + \varepsilon/2}{\varepsilon}\right)^{\frac{1}{k}}.$$

Such a choice of s guarantees that if G_s contains any k-clique, it contains at least $\frac{1+\varepsilon/2}{\varepsilon}$ k-cliques. For simplicity we will have this assumption directly for G. We will also assume that if $n = |V|$ and $m = |E|$, then $2n + m > 6$.

Now we construct an instance $\mathbf{A}, \mathbf{B}$ of p-$\#\mathrm{CSP}(\mathcal{C},-)$ such that an $\varepsilon/2$-approximation of the number of homomorphisms from $\mathbf{A}$ to $\mathbf{B}$ yields an ε-approximation of the number of k-cliques in G. Structures $\mathbf{A}, \mathbf{B}$ will be chosen to be (essentially) $\mathbf{A} = L(k, r, \ell_1, \ell_2)$ and $\mathbf{B} = H(G, k, W_1, W_2)$, where the parameters ℓ_1, ℓ_2, W_1, W_2 are set according to Lemma 8. Let $r = \binom{k}{2}$, $\ell_1 = 17\ell_2$, and $\ell_2 = 8kr$.

Since $\mathcal{C}$ is a fan class and we assume that $\mathcal{C}$ is not of bounded treewidth, there is a structure $\mathbf{A}$ in $\mathcal{C}$ such that $L(k, r, \ell_1, \ell_2)$ is the Gaifman graph $G(\mathbf{A})$ of $\mathbf{A}$.

We enumerate the class $\mathcal{C}$ until we find such an $\mathbf{A}$. Since $L(k, r, \ell_1, \ell_2)$ does not contain triangles, we can without loss of generality assume that $\mathbf{A}$ is τ-structure where τ consists of a single binary relation symbol; i.e., $\mathbf{A}$ is a graph and hence $L(k, r, \ell_1, \ell_2)$. Let $\mathbf{B} = H(G, k, W_1, W_2)$, where $W_1 = (2n + m)^4$ and $W_2 = (2n + m)^2$. Since the parameters $n, m, \ell_1, \ell_2, W_1, W_2$ satisfy the conditions of Lemma 8, by that lemma we have

$$N < \frac{M}{2W_1^{4\ell_1} W_2^{8\ell_2} k!} < N + \frac{1}{2}, \tag{1}$$

where N is the number of k-cliques in G, which we want to approximate within ε, and M is the number of homomorphisms from $\mathbf{A}$ to $\mathbf{B}$, for which we have an FPTRAS by assumption. Let $Q = M/(2W_1^{4\ell_1} W_2^{8\ell_2} k!)$. The FPTRAS for p-$\#\mathrm{CSP}(\mathcal{C},-)$ applied with error tolerance $\varepsilon/2$ produces a number M' such that

$$(1 - \varepsilon/2)M < M' < (1 + \varepsilon/2)M. \tag{2}$$

We then return $\lfloor Q' \rfloor$, where

$$Q' = \frac{M'}{2W_1^{4\ell_1} W_2^{8\ell_2} k!}.$$

It remains to show that $(1-\varepsilon)N < Q' < (1+\varepsilon)N$. On one hand, we have

$$Q' > (1-\varepsilon/2)Q \geq (1-\varepsilon/2)N \geq (1-\varepsilon)N,$$

where the first inequality follows from (2) and the definitions of Q and Q', the second inequality follows from (1) and the definitions of Q and N, and the third inequality is trivial.

On the other hand, we have

$$Q' < (1+\varepsilon/2)Q < (1+\varepsilon/2)(N+\frac{1}{2}),$$

where again the first inequality follows from (2) and the second inequality follows from (1).

Assume first that $N = 0$. Then $Q' < \frac{1+\varepsilon/2}{2}$, and by the assumption $\varepsilon < 1$ we have $\lfloor Q' \rfloor = 0$ as required. Otherwise by the assumption on the number of k-cliques in G, $N > \frac{1+\varepsilon/2}{\varepsilon}$; therefore

$$Q' < (1+\varepsilon/2)(N+\frac{1}{2}) = (1+\varepsilon/2)N + \frac{1+\varepsilon/2}{2} < (1+\varepsilon/2)N + (\varepsilon/2)N = (1+\varepsilon)N.$$

Observe that the reduction runs in time $f(k) \cdot \mathsf{poly}(n+m, \varepsilon^{-1})$ and is a parametrised AP-reduction. Thus, the reduction gives an FPTRAS for N. Theorem 3 is proved. ◀

5 Conclusions

We do not know whether Theorem 3 holds for all classes of (bounded-arity) relational structures.

With more technicalities (but the same ideas as presented in this extended abstract), one can weaken the assumption on a fan class to obtain the same result (Theorem 3). In particular, it suffices to require that there are polynomials f_1, f_2, f_3, f_4 such that for any parameters $k, r, \ell_1, \ell_2 \in \mathbb{N}$, $G(\mathcal{C})$ contains the fan-grid $L(k', r', \ell_1', \ell_2')$, where $k' = f_1(k, r, \ell_1, \ell_2) \geq k$, $r' = f_2(k, r, \ell_1, \ell_2) \geq r$, $\ell_1' = f_3(k, r, \ell_1, \ell_2) \geq \ell_1$, $\ell_2' = f_4(k, r, \ell_1, \ell_2) \geq \ell_2$. This can be achieved by making use of Lemma 9 (as it would not be possible to test directly for cliques of all sizes) and by a modification of the construction from Section 4.1 (to accommodate for the fact that some fan-grids may not correspond to cliques due to incompatible numbers).

References

1 Vikraman Arvind and Venkatesh Raman. Approximation Algorithms for Some Parameterized Counting Problems. In *Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC'02)*, volume 2518 of *Lecture Notes in Computer Science*, pages 453–464. Springer, 2002. doi:10.1007/3-540-36136-7_40.

2 Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding Model Counting for beta-acyclic CNF-formulas. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS'15)*, pages 143–156, 2015. doi:10.4230/LIPIcs.STACS.2015.143.

3 Andrei Bulatov. A dichotomy theorem for nonuniform CSP. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*, pages 319–330. IEEE, 2017.

4 Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the Complexity of Constraints using Finite Algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005. doi:10.1137/S0097539700376676.

5 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *Journal of the ACM*, 60(5):34, 2013. doi:10.1145/2528400.

6 Jin-Yi Cai and Xi Chen. Complexity of Counting CSP with Complex Weights. *Journal of the ACM*, 64(3):19:1–19:39, 2017. doi:10.1145/2822891.

7 Xi Chen, Martin E. Dyer, Leslie Ann Goldberg, Mark Jerrum, Pinyan Lu, Colin McQuillan, and David Richerby. The complexity of approximating conservative counting CSPs. *Journal of Computer and System Sciences*, 81(1):311–329, 2015.

8 Nadia Creignou and Miki Hermann. Complexity of Generalized Satisfiability Counting Problems. *Information and Computation*, 125(1):1–12, 1996.

9 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.

10 Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*, volume 2470 of *Lecture Notes in Computer Science*, pages 310–326. Springer, 2002. doi:10.1007/3-540-46135-3_21.

11 Reinhard Diestel. *Graph Theory*. Springer, fourth edition, 2010.

12 Rodney G. Downey and Michael R. Fellows. Fixed-Parameter Tractability and Completeness I: Basic Results. *SIAM Journal on Computing Computing*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.

13 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1–2):109–131, 1995.

14 Rodney G. Downey, Michael R. Fellows, and Kenneth W. Regan. Parameterized Circuit Complexity and the W Hierarchy. *Theoretical Computer Science*, 191(1-2):97–115, 1998. doi:10.1016/S0304-3975(96)00317-9.

15 Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. The Relative Complexity of Approximate Counting Problems. *Algorithmica*, 38(3):471–500, 2004. doi:10.1007/s00453-003-1073-y.

16 Martin E. Dyer, Leslie Ann Goldberg, and Mark Jerrum. An approximation trichotomy for Boolean #CSP. *Journal of Computer and System Sciences*, 76(3-4):267–277, 2010. doi:10.1016/j.jcss.2009.08.003.

17 Martin E. Dyer and Catherine S. Greenhill. The complexity of counting graph homomorphisms. *Random Struct. Algorithms*, 17(3-4):260–289, 2000.

18 Martin E. Dyer and David Richerby. An Effective Dichotomy for the Counting Constraint Satisfaction Problem. *SIAM Journal on Computing*, 42(3):1245–1274, 2013. doi:10.1137/100811258.

19 Tomás Feder, Pavol Hell, Daniel Král', and Jiří Sgall. Two algorithms for general list matrix partitions. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 870–876, 2005.

20 Tomás Feder, Pavol Hell, and Wing Xie. Matrix Partitions with Finitely Many Obstructions. *Electr. J. Comb.*, 14(1), 2007.

21 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.

22 Jörg Flum and Martin Grohe. The Parameterized Complexity of Counting Problems. In *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS'02)*, page 538. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181978.

23 Jörg Flum and Martin Grohe. The Parameterized Complexity of Counting Problems. *SIAM Journal on Computing*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.

24 Jörg Flum and Martin Grohe. *Parametrized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

25 Andreas Galanis, Leslie Ann Goldberg, and Mark Jerrum. Approximately Counting H-Colorings is #BIS-Hard. *SIAM Journal on Computing*, 45(3):680–711, 2016.

26 Andreas Galanis, Leslie Ann Goldberg, and Mark Jerrum. A Complexity Trichotomy for Approximately Counting List H-Colorings. *ACM Transactions on Computation Theory*, 9(2):9:1–9:22, 2017. `doi:10.1145/3037381`.

27 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

28 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decomposition and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002. `doi:10.1006/jcss.2001.1809`.

29 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1–24, 2007. `doi:10.1145/1206035.1206036`.

30 Martin Grohe and Dániel Marx. Constraint Solving via Fractional Edge Covers. *ACM Transactions on Algorithms*, 11(1):4:1–4:20, 2014. `doi:10.1145/2636918`.

31 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *Proceedings 33rd ACM Symposium on Theory of Computing (STOC'01)*, pages 657–666. ACM, 2001. `doi:10.1145/380752.380867`.

32 Pavol Hell and Jaroslav Nešetřil. On the Complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990. `doi:10.1016/0095-8956(90)90132-J`.

33 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`.

34 Kitty Meeks. The challenges of unbounded treewidth in parameterised subgraph counting problems. *Discrete Applied Mathematics*, 198:170–194, 2016. `doi:10.1016/j.dam.2015.06.019`.

35 Michael Mitzenmacher and Eli Upfal. *Probability and Computing.* Cambridge University Press, Cambridge, second edition, 2017. Randomization and Probabilistic techniques in Algorithms and Data Analysis.

36 Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. `doi:10.1016/0095-8956(84)90013-3`.

37 Neil Robertson and Paul D. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986. `doi:10.1016/0095-8956(86)90030-4`.

38 Thomas J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOCS'78)*, pages 216–226, 1978.

39 Dmitriy Zhuk. The Proof of CSP Dichotomy Conjecture. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*, pages 331–342. IEEE, 2017.

Uniformisation Gives the Full Strength of Regular Languages

Nathan Lhote
University of Warsaw, Poland
http://di.ulb.ac.be/verif/lhote/
nlhote@mimuw.edu.pl

Vincent Michielini
University of Warsaw, Poland
michielini@mimuw.edu.pl

Michał Skrzypczak
University of Warsaw, Poland
https://www.mimuw.edu.pl/~mskrzypczak/
mskrzypczak@mimuw.edu.pl

Abstract

Given R a binary relation between words (which we treat as a language over a product alphabet $\mathbb{A} \times \mathbb{B}$), a uniformisation of it is another relation L included in R which chooses a single word over $\mathbb{B}$, for each word over $\mathbb{A}$ whenever there exists one. It is known that **MSO**, the full class of regular languages, is strong enough to define a uniformisation for each of its relations. The quest of this work is to see which other formalisms, weaker than **MSO**, also have this property. In this paper, we solve this problem for pseudo-varieties of semigroups: we show that no nonempty pseudo-variety weaker than **MSO** can provide uniformisations for its relations.

2012 ACM Subject Classification Theory of computation → Regular languages

Keywords and phrases pseudo-variety, finite word, semigroup, uniformisation, regular language

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.61

Funding The first two authors have been supported by the European Research Council (ERC) grant under the European Union's Horizon 2020 research and innovation programme (ERC Consolidator Grant LIPA, grant agreement No. 683080). The first author has also been supported by the DeLTA project (ANR-16-CE40-0007). The last author has been supported by Poland's National Science Centre (NCN) (grant No. 2016/21/D/ST6/00491).

1 Introduction

Regular languages of finite words lie at the core of modern automata theory. The study of their properties has led to multiple fundamental discoveries. Among these discoveries was a formal introduction of the model of non-deterministic automata by Rabin and Scott [16]. Later, the results of Büchi, Elgot, and Trakhtenbrot [2, 7, 25] laid the foundations of the correspondence between automata and Monadic Second-Order (**MSO**) logic. That correspondence is now considered a golden standard, with notable extensions to other structures, like finite and infinite trees. Another breakthrough obtained over finite words was the effective characterisation of the class of star-free languages by McNaughton, Papert and Schützenberger [19, 11]. Again, this result has opened a rich area of extensions, first to infinite words [24], and later to other structures and classes of languages [23, 21, 1]. From the perspective of these results, the theory of regular languages of finite words can be seen as a test ground for novel problems and methods.

The situation is a bit different with the problem of uniformisation. This problem asks, to find an effectively definable graph of a function that is contained in a given relation R.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 61; pp. 61:1–61:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Thus, it can be seen as an instance of the choice axiom: the relation $R(x, y)$ admits multiple witnesses y for each argument x and our task is to choose one of them. The origins of that problem come from descriptive set theory, with the famous theorems like that of Novikov and Kondô [9, Theorem 36.12].

The problem of uniformisation was translated to the context of automata theory by Rabin [15], directly for the most complex structures – infinite trees. In that context, the relation R is given by an **MSO**-definable language over a product alphabet $\mathbb{A} \times \mathbb{B}$, and the question of uniformisation asks to find an **MSO**-definable language that realises a function from structures over $\mathbb{A}$ to structures over $\mathbb{B}$. Therefore, the question of uniformisability can be read as the problem of effective selection of witnesses – a way of making non-determinism somehow controlled. In the course of research over that problem, it was shown that **MSO**-definable uniformisation is always possible over infinite words [20, 10, 17]; but not over infinite trees [8, 4]. In parallel, a study of *sequential uniformisation* was performed in the context of games and problems of synthesis [3].

The fact that **MSO** has[1] the uniformisation property over finite words is easy to prove and is considered folklore: it is enough to choose the lexicographically minimal witness. That is probably the reason why the case of finite words was somehow ignored in the study of uniformisation problems. It was opened by a recent paper [12], where the author asks about the possibility to uniformise for certain logics weaker than **MSO**. The results of that work are not unequivocal. On the one hand, it is shown that for multiple pairs of logics $\mathbf{L}_1 \subseteq \mathbf{L}_2 \subseteq \mathbf{MSO}$, there exists a relation R definable in $\mathbf{L}_1$ with no $\mathbf{L}_2$-definable uniformisation. On the other hand, it is shown that each relation definable in $\mathbf{FO}[\,]$ (First-Order logic with only equality and letter tests) can be effectively uniformised within First-Order logic with the order predicate. This leads to an intriguing graph of logics, one (not) uniformising another. To simplify the situation, the author formulated the above question for $\mathbf{L}_1 = \mathbf{L}_2$, *i.e.* the problem whether a given logic uniformises itself. Based on the provided results, the following conjectured:

▶ **Conjecture 1** (Conjecture 1 in [12])**.** *Let* $\mathbf{L}$ *be a fragment of* **MSO** *such that* $\mathbf{FO}^2[\,] \subseteq \mathbf{L}$ *and* $\mathbf{L}$ *satisfies some closure properties (to be specified). If* $\mathbf{L}$ *has the uniformisation property, then* $\mathbf{L}$ *is* **MSO***.*

The main result of the present paper is a positive answer to the above conjecture. The assumption that the logic $\mathbf{FO}^2[\,]$ (the two-variable fragment of First-Order logic) is contained in a given logic turned out to be unnecessary, and the relevant closure properties boil down to the standard notion of a pseudo-variety.

▶ **Theorem 2.** **MSO** *is the unique nonempty self-uniformisable pseudo-variety of semigroups.*

A class of languages corresponds to a pseudo-variety of semigroups if it is closed under Boolean combinations, left and right quotients, and pre-images under non-erasing homomorphisms. It is known that most of the classically considered logics correspond to pseudo-varieties of semigroups [18]. Since we restrict to semigroups instead of monoids (*i.e.* we require the considered homomorphisms of words to be non-erasing), this definition also captures logics with successor instead of the order. Among the notable examples of logics that do not correspond to pseudo-varieties of semigroups are the logics with modulo predicates, like $\mathbf{FO}[\leq, \mathbf{MOD}_2]$. These logics are not covered by the presented arguments. To deal with them, one would need to consider homomorphisms of words that are *length-preserving* or *length-multiplying*, see *e.g.* [14].

[1] We identify a logic with the class of languages it defines. Therefore **MSO** = **REG**, *i.e.* the class of all regular languages.

When read in terms of logics, the above result says that most of the widely considered formalisms over finite words weaker than **MSO** do not have the uniformisation property. As generally in the case of negative results, the consequences of that are more theoretical than practical: there is no hope in finding a robust formalism, that would be easier to handle than **MSO** and still retain the ability to choose unique witnesses.

The provided proof is independent from the results of [12] – instead of comparing two logics $\mathbf{L}_1 \subseteq \mathbf{L}_2$, we focus on one formalism $\mathbf{L}$ with the assumption that $\mathbf{L}$ can uniformise itself. Based on that assumption, we gradually bootstrap the expressive power of the considered formalism. It is done in a sequence of steps, showing that $\mathbf{L}$ must be able to express more and more complex properties: test the letters that appear in a given word; recognise the order in which the letters appear; *etc.* Each of these steps is based on the assumption that $\mathbf{L}$ has the uniformisation property and therefore must be sensitive to certain modifications of the input words. Thus, the proofs of the lemmas are of a similar structure.

Nevertheless, we believe that it is non-trivial and instructive to see the ways in which uniformisability guarantees the considered expressive abilities. The difficulty of these arguments lies in the fact that the assumptions about $\mathbf{L}$ speak about the algebraic properties of the semigroups recognising languages in $\mathbf{L}$, while the notion of uniformisability is a set-theoretical property of the actual languages in $\mathbf{L}$.

While the proof goes on, we have more and more tools at hand, but at the same time we need to prove stronger and stronger expressibility properties about $\mathbf{L}$. Ultimately, it turns out that $\mathbf{L}$ is able to guess evaluations with respect to arbitrary finite semigroups (*i.e.* in a sense is closed under projection) and therefore must contain all regular languages of finite words. From this perspective, the proof can be seen as a variety of concrete recipes (ranging from the least complex properties to the most complex) explaining what is the interplay between the considered expressibility property and uniformisability.

Although the results are expressed over finite words, we believe that similar arguments can be adapted to the more complex structures, like infinite words or finite trees. Therefore, finite words are used here again as a testing ground, providing new understanding that can later be transferred to richer structures.

The paper is organised as follows. Section 2 is devoted to an introduction of all relevant technical notions. Then, Sections 3.1 up to 3.7 gradually increase the expressive power of the considered class of languages $\mathbf{L}$. Finally, in Section 4 we conclude.

2 Technical background

Words and languages

We identify each natural number n with the set $\{0,\ldots,n{-}1\}$, and we denote the set of all natural numbers by ω. An *alphabet* $\mathbb{A}$ is any finite set. A function from a natural number n to an alphabet $\mathbb{A}$ is called a *(finite) word* over $\mathbb{A}$. The natural number n is the *length* of w, and we denote it by $|w|$. For each $i \in n$ the element $w(i) \in \mathbb{A}$ is called the ith *letter* of w. We write $w = w(0) \cdot w(1) \cdots w(n{-}1)$, but notice that this notation is ambiguous when the alphabet is not clear in the context: if we write $w = a \cdot b$, we do not know *a priori* if we see w as a word over $\{a, b\}$, or over any other alphabet containing $\{a, b\}$. We extend this notation: if w_1 and w_2 are two words over $\mathbb{A}$, then $w_1 \cdot w_2$, the *concatenation* of w_1 and w_2, is the word w over $\mathbb{A}$ of length $|w_1| + |w_2|$ defined by $w(i) = w_1(i)$ for $i \in |w_1|$ and $w(i) = w_2(i - |w_1|)$ for $i \geq |w_1|$.

A word of length 0 is denoted by ϵ and called the *empty word*. The set of all words over $\mathbb{A}$ is denoted by $\mathbb{A}^*$, $\mathbb{A}^n$ is the set of all words of length $n \in \omega$, and $\mathbb{A}^+$ is $\mathbb{A}^* \setminus \{\epsilon\}$, *i.e.* the set of nonempty words over $\mathbb{A}$. Note that $\emptyset^* = \{\epsilon\}$ and $\emptyset^+ = \emptyset$.

In this paper, a *language* of words over an alphabet $\mathbb{A}$ is any subset[2] of $\mathbb{A}^+$. Once again, notice that a language has to be given with its alphabet. To avoid any ambiguity like the one above, we sometimes write $\langle L, \mathbb{A}\rangle$ to emphasise the choice of the alphabet.

In order to ease the reading of the paper, we will denote by $\mathbb{A}, \mathbb{A}_0, \mathbb{A}_1 \ldots$ alphabets of letters $a, b, \ldots, x, y \ldots$ and by $\mathbb{B}, \mathbb{B}_0, \mathbb{B}_1 \ldots$ alphabets of symbols $\Box, \bullet, \triangle, \ldots$ Words over the alphabets of letters will be denoted by $u, v, w \ldots$ while words over the alphabets of symbols will be denoted by $\pi, \sigma, \tau, \ldots$

Semigroups and pseudo-varieties

In this paper, the classes of languages which we focus on correspond to *pseudo-varieties of semigroups.*

A *semigroup* is a set S provided with an associative binary operation, which we will denote by $\cdot$. We identify the semigroup $\langle S, \cdot\rangle$ with its set S. For $s \in S$ and for a natural number $n \geq 1$, s^n denotes the product $s \cdots s$, where s appears n times. An element $e \in S$ is said to be *idempotent* if $e^2 = e$.

It is considered folklore to prove that for every finite semigroup S, there exists a natural number $n \geq 1$ such that s^n is idempotent for each $s \in S$. We denote this natural number by $\sharp(S)$. When the semigroup is known from the context, we just write $\sharp$ instead of $\sharp(S)$. Notice that in the literature the symbol ω is often used instead of $\sharp$.

If S is a semigroup, then $S' \subseteq S$ is said to be a *sub-semigroup* of S if it is stable by the operation of S, that is, if for all s, t in S', $s \cdot t \in S'$. Finally, if S_1 and S_2 are two semigroups, then $S_1 \times S_2$ provided with the operation defined by $\langle s_1, s_2\rangle \cdot \langle t_1, t_2\rangle = \langle s_1 \cdot s_2, t_1 \cdot t_2\rangle$ is also a semigroup. It is called the *product* of S_1 and S_2.

Let S_1 and S_2 be two semigroups. A *homomorphism* from S_1 to S_2 is a function α from S_1 to S_2 such that for all x, y in S_1, we have $\alpha(x \cdot y) = \alpha(x) \cdot \alpha(y)$. Such a homomorphism is *surjective* (resp. *injective*, *bijective*) if so is the respective function α.

For each alphabet $\mathbb{A}$, the set $\mathbb{A}^+$ provided with the concatenation operation, is a semigroup that is known as the *free semigroup on* $\mathbb{A}$. The fact that a homomorphism $\alpha\colon \mathbb{A}^+ \to S$ must preserve the operations of the semigroups, implies that α is uniquely determined by its action on the single letters in $\mathbb{A}$.

The following variant of Ramsey's theorem is often used when working with finite semigroups.

▶ **Theorem 3** (Simon [22], see also Section II.11.1 in [13])**.** *Let $\mathbb{A}$ be an alphabet and α a homomorphism from $\mathbb{A}^+$ to some finite semigroup S. For each natural number $n \geq 2$, there exists a natural number $N(n)$ such that for each word w over $\mathbb{A}$ of length at least $N(n)$, there exists an idempotent e in S and a decomposition $w = u \cdot w_0 \cdots w_{n-1} \cdot v$, where for all $i \in n$, w_i is nonempty and $\alpha(w_i) = e$.*

Let L be a language of words over some alphabet $\mathbb{A}$ and let S be a finite semigroup. We say that S *recognises* L if there exists a homomorphism α from $\mathbb{A}^+$ to S and $T \subseteq S$ such that $L = \alpha^{-1}(T)$. In such a case we also say that the tuple $\langle S, \alpha, T\rangle$ *recognises* L.

A language of words over $\mathbb{A}$ is *regular* if it is recognised by some finite semigroup S. We denote the class of all regular languages by **REG**. As mentioned in the introduction, the class **REG** coincides with the class of languages definable in Monadic Second-Order logic (denoted **MSO**); however as logic is not directly involved in the presentation, we will rather use **REG** to emphasise the automata- and semigroup-based approach.

[2] It is more standard to define languages as subsets of $\mathbb{A}^*$ but then the natural algebraic structures are monoids, and not semigroups.

Let $\mathbf{L}$ be a class of languages and $\mathbf{S}$ a class of finite semigroups, we say that $\mathbf{L}$ *corresponds* to $\mathbf{S}$ if it is exactly the class of languages recognised by the semigroups S of $\mathbf{S}$. Notice that, under this assumption, $\mathbf{L}$ only contains regular languages.

We define now an important notion: the notion of pseudo-varieties.

▶ **Definition 4.** *Let $\mathbf{S}$ be a class of finite semigroups. We say that $\mathbf{S}$ is a* pseudo-variety *of (finite) semigroups if it has the following properties:*

- *if $S_1 \in \mathbf{S}$ and if S_2 is a sub-semigroup of S_1, then $S_2 \in \mathbf{S}$,*
- *if S_1 and S_2 are in $\mathbf{S}$, then the product semigroup $S_1 \times S_2$ is in $\mathbf{S}$,*
- *if $S_1 \in \mathbf{S}$ and if there exists a surjective homomorphism from S_1 to S_2, then $S_2 \in \mathbf{S}$.*

The following theorem is a part of the so-called Eilenberg's variety theory.

▶ **Theorem 5** (Eilenberg [6])**.** *Let $\mathbf{L}$ be a class of regular languages. Then the following propositions are equivalent:*

- *$\mathbf{L}$ corresponds to a pseudo-variety of semigroups;*
- *$\mathbf{L}$ has the following closure properties:*
  - *$\mathbf{L}$ is closed under Boolean operations: for each $\langle L_1, \mathbb{A}\rangle$ and $\langle L_2, \mathbb{A}\rangle$ in $\mathbf{L}$, $\langle L_1 \cup L_2, \mathbb{A}\rangle$ and $\langle L_1^{\mathrm{c}} := \mathbb{A}^+ \setminus L_1, \mathbb{A}\rangle$ are in $\mathbf{L}$ (and therefore also $\langle L_1 \cap L_2, \mathbb{A}\rangle$),*
  - *$\mathbf{L}$ is closed under quotients: for each $\langle L, \mathbb{A}\rangle \in \mathbf{L}$ and each word $w \in \mathbb{A}^+$, $\langle w^{-1} \cdot L, \mathbb{A}\rangle$ and $\langle L \cdot w^{-1}, \mathbb{A}\rangle$ are in $\mathbf{L}$, where $w^{-1} \cdot L = \{u \in \mathbb{A}^+ \mid w \cdot u \in L\}$, and $L \cdot w^{-1}$ is defined symmetrically,*
  - *$\mathbf{L}$ is closed under pre-images of semigroup homomorphisms: for each alphabet $\mathbb{A}$, language $\langle L, \mathbb{B}\rangle \in \mathbf{L}$, and homomorphism φ from $\mathbb{A}^+$ to $\mathbb{B}^+$, the language $\langle \varphi^{-1}(L), \mathbb{A}\rangle$ is in $\mathbf{L}$.*

All the classes of languages discussed in [12] are pseudo-varieties of semigroups. The most common are $\mathbf{MSO}$ and $\mathbf{FO}[<]$ (First-Order logic with the order). Other examples include $\mathbf{FO}^2[<]$, the fragment of $\mathbf{FO}[<]$ where only two distinct variables are allowed; and $\mathbf{FO}[\mathbf{s}]$, where instead of the order one allows the successor function $\mathbf{s}$.

Uniformisation

Let $\mathbb{A}$ and $\mathbb{B}$ be two alphabets. If $a \in \mathbb{A}$ and $\square \in \mathbb{B}$ are two letters then their pair is denoted $\binom{a}{\square} \in \mathbb{A} \times \mathbb{B}$. Let w, π be two words over $\mathbb{A}$ and $\mathbb{B}$ respectively, such that $|w| = |\pi|$. Then the pair $\langle w, \pi\rangle \in \mathbb{A}^* \times \mathbb{B}^*$ can be identified with $\binom{w}{\pi}$, the word over the product alphabet $\mathbb{A} \times \mathbb{B}$ satisfying $\binom{w}{\pi}(i) = \binom{w(i)}{\pi(i)}$ for all $i \in |w|$. This vertical notation is also extended to sets of words of fixed length, for instance, $\binom{\{a,b\}}{\square} = \{\binom{a}{\square}, \binom{b}{\square}\}$ is a set of two letters in $\mathbb{A} \times \mathbb{B}$.

Let $R \subseteq (\mathbb{A} \times \mathbb{B})^+$. Based on the previous identification, R can be seen as a binary relation between words over $\mathbb{A}$ and words over $\mathbb{B}$. The *projection* of R is the set of words $w \in \mathbb{A}^+$ such that there exists a word $\pi \in \mathbb{B}^{|w|}$ with $\binom{w}{\pi} \in R$. We denote this set by $\Pi(R)$.

A *uniformisation* of R is a relation $F \subseteq R$ such that $\Pi(F) = \Pi(R)$, and being functional, *i.e.* if $\binom{w}{\pi_1} \in F$ and $\binom{w}{\pi_2} \in F$ then $\pi_1 = \pi_2$. In that case, for each word $w \in \Pi(R)$, the unique π such that $\binom{w}{\pi} \in F$ is called the *image* of w by F.

A class $\mathbf{L}$ of languages is said to have the *uniformisation property* if each relation $R \in \mathbf{L}$ admits a uniformisation $F \in \mathbf{L}$. We also call such a class *self-uniformisable*. The fact that $\mathbf{REG}$ (*i.e.* the class of all regular languages) has the uniformisation property is considered folklore.

3 Proof of the theorem

We now begin the proof of Theorem 2. For the rest of the section, $\mathbf{L}$ denotes a class of regular languages corresponding to a nonempty pseudo-variety $\mathbf{S}$ of semigroups, and we assume that $\mathbf{L}$ has the uniformisation property. In the following subsections, we show that $\mathbf{L}$ contains certain specific languages and allows to express more and more complex properties of words. Ultimately, we show in Subsection 3.7 that $\mathbf{L}$ can validate evaluations with respect to finite semigroups, and therefore can recognise all regular languages.

3.1 Testing letters

Recall that by Theorem 5 we know that $\mathbf{L}$ is closed under Boolean operations. Also, as $\mathbf{S}$ is nonempty, we know that every *full language* $\mathbb{A}^+$ over some alphabet $\mathbb{A}$ belongs to $\mathbf{L}$: $\mathbb{A}^+ = \alpha^{-1}(S)$, for any semigroup $S \in \mathbf{S}$, and any homomorphism from $\mathbb{A}^+$ to S.

This section is devoted to a first step of the proof: we show that $\mathbf{L}$ must be able to detect which letters appear in the given word. More formally, the main result of this section is the following lemma.

▶ **Lemma 6.** *For all alphabets $\mathbb{A}_1 \subseteq \mathbb{A}_2$, the language $\langle \mathbb{A}_1^+, \mathbb{A}_2 \rangle$ is in $\mathbf{L}$.*

One can equivalently state the above lemma by saying that $\mathbf{FO}^1[] \subseteq \mathbf{L}$, or that $\mathbf{S}$ contains the pseudo-variety $\mathbf{J}_1$ of finite idempotent commutative semigroups, see [5]. However, the above statement seems to better fit the rest of the presentation.

This whole subsection is devoted to a proof of Lemma 6. To prove it, notice first that it is enough to show that the semigroup $2 = \{0, 1\}$ with the operation max belongs to $\mathbf{S}$. Indeed, given two alphabets $\mathbb{A}_1 \subseteq \mathbb{A}_2$ one can consider the homomorphism α from words over $\mathbb{A}_2$ to 2, defined by $\alpha(a) = 0$ for $a \in \mathbb{A}_1$ and $\alpha(a) = 1$ otherwise. Then, $\mathbb{A}_1^+ = \alpha^{-1}(\{0\})$, and therefore belongs to $\mathbf{L}$.

Let R be the full relation between words over $\mathbb{A} = \{x\}$ and words over $\mathbb{B} = \{\square, \triangle\}$: $R = \langle (\mathbb{A} \times \mathbb{B})^+, \mathbb{A} \times \mathbb{B} \rangle$. As discussed above, R is in $\mathbf{L}$. By the assumption, $\mathbf{L}$ must contain a uniformisation F of R that is recognised by some tuple $\langle S, \alpha, T \rangle$, with $S \in \mathbf{S}$.

Let $N = N(2)$ be the number we obtain from Theorem 3 applied for S and α in the particular case $n = 2$. Consider the word $w = x^N$, and take the unique word $\pi \in \{\square, \triangle\}^N$ such that $\binom{w}{\pi} \in F$. For convenience, for all $i \in j \in N{+}1$, we write $w_{i,j}$ (resp. $\pi_{i,j}$) for the word $w(i) \dots w(j{-}1)$ (resp. $\pi(i) \dots \pi(j{-}1)$), and $s_{i,j}$ for $\alpha(\binom{w_{i,j}}{\pi_{i,j}})$.

By the definition of N, we know that there exists an idempotent e of S, and $i \in j \in k \in N{+}1$, such that $s_{i,j} = s_{j,k} = e$. Since $j - i > 0$ and $|\mathbb{B}| \geq 2$, there exists a word $\pi' \in \mathbb{B}^{j-i}$ distinct from $\pi_{i,j}$. We define $s' = \alpha(\binom{w_{i,j}}{\pi'})$.

As e is idempotent, we know that for every $\ell \geq 1$ we have $s_{0,i} \cdot e^\ell \cdot s_{k,N} = s_{0,N} \in T$. Recall that $\sharp = \sharp(S)$ is a number such that for every $s \in S$ the element $s^\sharp$ is idempotent. Consider the particular case of the above equality for $\ell = 3 \times \sharp$, we obtain:

$$\binom{w_{0,i}}{\pi_{0,i}} \cdot \left(\binom{w_{i,j}}{\pi_{i,j}} \cdot \binom{w_{i,j}}{\pi_{i,j}} \cdot \binom{w_{i,j}}{\pi_{i,j}} \right)^\sharp \cdot \binom{w_{k,N}}{\pi_{k,N}} \in F.$$

As $\pi' \neq \pi_{i,j}$ and F is a uniformisation, we know that

$$\binom{w_{0,i}}{\pi_{0,i}} \cdot \left(\binom{w_{i,j}}{\pi_{i,j}} \cdot \binom{w_{i,j}}{\pi'} \cdot \binom{w_{i,j}}{\pi_{i,j}} \right)^\sharp \cdot \binom{w_{k,N}}{\pi_{k,N}} \notin F.$$

This implies that $e' := \left(s_{i,j} \cdot s' \cdot s_{i,j}\right)^\sharp = \left(e \cdot s' \cdot e\right)^\sharp \neq e$. Now, we set $s_0 = e$ and $s_1 = e'$. As both e and e' are idempotents, we know that $s_0 \cdot s_0 = s_0$ and $s_1 \cdot s_1 = s_1$. Moreover, because e is idempotent, it is immediate to see that $s_1 \cdot s_0 = \left(e \cdot s' \cdot e\right)^\sharp \cdot e = \left(e \cdot s' \cdot e\right)^\sharp = s_1$, and that we also have $s_0 \cdot s_1 = s_1$ symmetrically. Therefore, the subset $\{s_0, s_1\}$ of S is a sub-semigroup and it is isomorphic to $\langle 2, \max\rangle$. Because $\mathbf{S}$ is stable by taking sub-semigroups and images by surjective images, $\langle 2, \max\rangle \in \mathbf{S}$. This concludes the proof of Lemma 6.

3.2 Changing alphabets

The aim of this short section is to show that $\mathbf{L}$ is strong enough not to depend on the actual alphabet of a given language. This property is expressed by the following two lemmas.

▶ **Lemma 7.** *Let $\mathbb{A}_1 \subseteq \mathbb{A}_2$ be two alphabets and $L \subseteq \mathbb{A}_1^+$. If $\langle L, \mathbb{A}_2\rangle \in \mathbf{L}$ then $\langle L, \mathbb{A}_1\rangle \in \mathbf{L}$.*

Proof. Let $\langle S, \alpha, T\rangle$ be a tuple recognising L, with $S \in \mathbf{S}$. Let βbe the homomorphism from $\mathbb{A}_1^+$ to S defined by $\beta(a) = \alpha(a)$ for $a \in \mathbb{A}_1$. Then, by the assumption, we have $L = \beta^{-1}(T)$, and therefore $\langle L, \mathbb{A}_1\rangle \in \mathbf{L}$. ◀

▶ **Lemma 8.** *Let $\mathbb{A}_1 \subseteq \mathbb{A}_2$ be two alphabets and $L \subseteq \mathbb{A}_1^+$. If $\langle L, \mathbb{A}_1\rangle \in \mathbf{L}$ then $\langle L, \mathbb{A}_2\rangle \in \mathbf{L}$.*

Proof. Let $\langle L, \mathbb{A}_1\rangle \in \mathbf{L}$ with and $\langle S_1, \alpha_1, T_1\rangle$ a tuple recognising it, with $S_1 \in \mathbf{S}$. Assume that $\mathbb{A}_1 \subseteq \mathbb{A}_2$. We prove that $\langle L, \mathbb{A}_2\rangle$ is in $\mathbf{L}$.

We showed in the proof of Lemma 6 that the semigroup $\langle 2, \max\rangle$ is in $\mathbf{S}$. This implies that $S_1 \times 2$, with the natural product operation, is also in $\mathbf{S}$. Now, let β be the homomorphism from $\mathbb{A}_2^+$ to $S_1 \times S_2$ defined by $\beta(a) = \langle \alpha_1(a), 0\rangle$ for $a \in \mathbb{A}_1$, and $\beta(a) = \langle \alpha_1(a), 1\rangle$ for $a \in \mathbb{A}_2 \setminus \mathbb{A}_1$.

It is easy to verify that $L = \beta^{-1}(T_1 \times \{0\})$, and therefore, $\langle L, \mathbb{A}_2\rangle \in \mathbf{L}$. ◀

Lemmas 7 and 8 imply that if $L \subseteq \mathbb{A}_1^+ \cap \mathbb{A}_2^+$, then $\langle L, \mathbb{A}_1\rangle \in \mathbf{L}$ iff. $\langle L, \mathbb{A}_2\rangle \in \mathbf{L}$. Therefore, we can simply say that $L \in \mathbf{L}$, without being specific about its alphabet. Thus, from that moment on we will speak simply about languages L, instead of $\langle L, \mathbb{A}\rangle$.

Using Lemma 6, we can deduce the following result:

▶ **Corollary 9.** *Let a be any letter if an alphabet $\mathbb{A}$. By $[\exists a]_\mathbb{A}$ we denote the language of words over $\mathbb{A}$ that contain at least one occurrence of a. Then $[\exists a]_\mathbb{A} \in \mathbf{L}$.*

Moreover, by $\mathbb{A}^\oplus$ if we denote the set of all words w over $\mathbb{A}$ such that each letter of $\mathbb{A}$ appears in w, then $\mathbb{A}^\oplus$ too is in $\mathbf{L}$.

Proof. It is enough to observe that $[\exists a]_\mathbb{A} = \left((\mathbb{A} \setminus \{a\})^+\right)^\mathrm{c}$, where $^\mathrm{c}$ denotes the complement over the full language $\mathbb{A}^+$. Lemma 6 tells us that $(\mathbb{A} \setminus \{a\})^+ \in \mathbf{L}$, and, since $\mathbf{L}$ is closed under Boolean combinations, $[\exists a]_\mathbb{A} \in \mathbf{L}$.

Now, $\mathbb{A}^\oplus = \bigcap_{a \in \mathbb{A}} [\exists a]_\mathbb{A}$, and therefore $\mathbb{A}^\oplus \in \mathbf{L}$. ◀

3.3 Counting letters

Our next step towards Theorem 2 is to notice that $\mathbf{L}$ is able to test single occurrences of letters, as expressed by the following lemma:

▶ **Lemma 10.** *Let a be a letter of an alphabet $\mathbb{A}$. Then the language of words over $\mathbb{A}$ having exactly one occurrence of a is in $\mathbf{L}$. We denote this language by $[\exists^{=1} a]_\mathbb{A}$.*

Similarly as in the case of Lemma 6, the above lemma can be equivalently expressed by saying that $\mathbf{FO}^2[\,] \subseteq \mathbf{L}$ (*i.e.* the two-variable fragment of $\mathbf{FO}$ without any predicates except equality and letter tests).

To prove the above lemma, consider three distinct letters, x, y, and z, and four distinct symbols $\otimes$, $\oplus$, $\ominus$, and $\odot$. Let R^x and R^y be the relations defined as:

$$R^x = \left\{\binom{x}{\oplus}, \binom{x}{\ominus}, \binom{y}{\otimes}, \binom{z}{\odot}\right\}^{\oplus},$$
$$R^y = \left\{\binom{x}{\otimes}, \binom{y}{\oplus}, \binom{y}{\ominus}, \binom{z}{\odot}\right\}^{\oplus}.$$

We know that R^x and R^y are in $\mathbf{L}$ because of Corollary 9. Finally, we define $R = R^x \cup R^y$, which is in $\mathbf{L}$ because it is closed under unions.

Since $\mathbf{L}$ has the uniformisation property, there exists $F \in \mathbf{L}$ uniformising R. Let $\langle S, \alpha, T\rangle$, with $S \in \mathbf{S}$, be a triple recognising F.

Now, for $p, q \in \omega$, we define L^x_p and L^y_q as the following two relations:

$$L^x_p = \left\{\binom{u}{\sigma} \in \left\{\binom{x}{\otimes}, \binom{z}{\odot}\right\}^+ \mid \otimes \text{ appears exactly } p \text{ times in } \sigma\right\},$$
$$L^y_q = \left\{\binom{v}{\tau} \in \left\{\binom{y}{\otimes}, \binom{z}{\odot}\Big)\right\}^+ \mid \otimes \text{ appears exactly } q \text{ times in } \tau\right\}.$$

Notice that $\bigcup_{p\in\omega} L^x_p = \left\{\binom{x}{\otimes}, \binom{z}{\odot}\right\}^+$ and similarly $\bigcup_{q\in\omega} L^y_q = \left\{\binom{y}{\otimes}, \binom{z}{\odot}\right\}^+$.

▷ Claim 11. At least one of the two following propositions is true:

- for all $p \geq 2$ we have $\alpha(L^x_1) \cap \alpha(L^x_p) = \emptyset$,
- for all $q \geq 2$ we have $\alpha(L^y_1) \cap \alpha(L^x_q) = \emptyset$.

Proof. Assume the contrary and take:

$$p \geq 2,\ \binom{u_1}{\sigma_1} \in L^x_1, \text{ and } \binom{u_p}{\sigma_p} \in L^x_p \text{ such that } \alpha\Big(\binom{u_1}{\sigma_1}\Big) = \alpha\Big(\binom{u_p}{\sigma_p}\Big)\text{; and} \tag{1}$$
$$q \geq 2,\ \binom{v_1}{\tau_1} \in L^y_1, \text{ and } \binom{v_p}{\tau_q} \in L^y_q \text{ such that } \alpha\Big(\binom{v_1}{\tau_1}\Big) = \alpha\Big(\binom{v_q}{\tau_q}\Big). \tag{2}$$

Let w be the word $u_p \cdot v_q \cdot z$ and assume that π is the unique word over $\{\oplus, \ominus, \otimes, \odot\}$ such that $\binom{w}{\pi} \in F$. Clearly, w is in the projection of both R^x and R^y; suppose that $\binom{w}{\pi} \in R^x$ (the case $\binom{w}{\pi} \in R^y$ is symmetric). As R^x determines the symbols below the letters y and z, we know that π is of the form $\sigma \cdot \tau_q \cdot \odot$, for some word σ over $\{\oplus, \ominus, \odot\}$ of length $|u_p|$.

Consider now the new word w' over $\{x, y, z\}$ defined with $w' = u_1 \cdot v_q \cdot z$. We know that w' belongs to the projection of R^y but not to the projection of R^x, because u_1 has only one occurrence of x. Let π' be the unique word such that $\binom{w'}{\pi'} \in F$. Similarly as before, $\pi' = \sigma_1 \cdot \tau' \cdot \odot$ for some word τ' over $\{\oplus, \ominus, \odot\}$ of length $|v_q|$.

Using (1) we know that $\alpha\Big(\binom{u_1}{\sigma_1}\Big) = \alpha\Big(\binom{u_p}{\sigma_p}\Big)$, and therefore $\binom{u_p}{\sigma_p} \cdot \binom{v_q}{\tau'} \cdot \binom{z}{\odot} \in F$, whose projection onto $\mathbb{A}$ equals w, which contradicts the fact that F is a uniformisation. ◀

By the symmetry, let us assume that the first item of Claim 11 holds, *i.e.* for all $\binom{u_1}{\sigma_1} \in L^x_1$ and $\binom{u_p}{\sigma_p} \in L^x_p$ with $p \geq 2$, we have $\alpha\Big(\binom{u_1}{\sigma_1}\Big) \neq \alpha\Big(\binom{u_p}{\sigma_p}\Big)$.

▷ Claim 12. The language L^x_1 is in $\mathbf{L}$.

Proof. The language $L^x_0 = \left\{\binom{z}{\odot}\right\}^+$ is in $\mathbf{L}$. Therefore, $\bigcup_{p\geq 1} L^x_p = \left\{\binom{x}{\otimes}, \binom{z}{\odot}\right\}^+ \setminus L^x_0$ also belongs to $\mathbf{L}$. Thus, the above assumption about α-values implies the claim, because $L^x_1 = \alpha^{-1}(\alpha(L^x_1)) \cap \bigcup_{p\geq 1} L^x_p$. ◁

Now take $a \in \mathbb{A}$ as in the statement of Lemma 10. Consider a homomorphism β from $\mathbb{A}^+$ to $\{\binom{x}{\otimes}, \binom{z}{\odot}\}^+$ defined by $\beta(a) = \binom{x}{\otimes}$ and $\beta(b) = \binom{z}{\odot}$ for $b \neq a$. We have $[\exists^{=1}a]_{\mathbb{A}} = \beta^{-1}(F_1^x)$, and, because $\mathbf{L}$ is closed under pre-images under homomorphisms, $[\exists^{=1}a]_{\mathbb{A}}$ is in $\mathbf{L}$. This concludes the proof of Lemma 10.

With a similar – yet more technical – proof, one can show that for all $p \in \omega$, $\mathbf{L}$ contains $[\exists^{=p}a]_{\mathbb{A}}$, the language of words over $\mathbb{A}$ having exactly p letters a, but this point will not be involved in the following demonstrations. This results show that $\mathbf{L}$ must contain $\mathbf{FO}[\,]$, First-Order logic with only equalities between positions and letter tests. Recall that by Proposition 2 in [12], $\mathbf{FO}[\,]$ can be uniformised within $\mathbf{FO}[<]$. This explains why our proof of Theorem 2 needs to use the fact that $\mathbf{L}$ uniformises itself more than once.

▶ **Corollary 13.** *By modifying the homomorphism used in the proof of Lemma 10, we obtain that if $\mathbb{A}_1 \subseteq \mathbb{A}_2$ then the language $[\exists^{=1}\mathbb{A}_1]_{\mathbb{A}_2}$ of words over $\mathbb{A}_2$ that contain exactly one occurrence of a letter from $\mathbb{A}_1$ also belongs to $\mathbf{L}$.*

3.4 Order on letters

Our next goal is to introduce the order $<$ on the positions of letters in a given word. This is achieved gradually, with the first instance of the order expressed by the following lemma:

▶ **Lemma 14.** *Let $\mathbb{A}$ be an alphabet and $a_0, \ldots, a_{p-1}$ be $p \geq 1$ pairwise distinct letters, that do not belong to $\mathbb{A}$. Then the language $L = \mathbb{A}^* \cdot a_0 \cdot \mathbb{A}^* \cdots \mathbb{A}^* \cdot a_{p-1} \cdot \mathbb{A}^*$ is in $\mathbf{L}$.*

Proof. The proof is quite similar to the previous one. We consider two distinct letters, x and y, and $p{+}1$ distinct symbols $\square, \triangle_0, \ldots, \triangle_{p-1}$, and we define the relation $R := \mathbb{C}^{\oplus}$, where $\mathbb{C}$ is the alphabet $\{\binom{y}{\square}\} \cup \{\binom{x}{\triangle_i} \mid i \in p\}$.

We know that $R \in \mathbf{L}$ and therefore it admits a uniformisation $F \in \mathbf{L}$. Let $\langle S, \alpha, T\rangle$ be a triple recognising it, with $S \in \mathbf{S}$. Fix $\sharp = \sharp(S)$.

We define now the word $u = y^\sharp \cdot x \cdot y^\sharp \cdots y^\sharp \cdot x \cdot y^\sharp$, where x appears exactly p times. Since u is in the projection of R, it also belongs to the projection of F. Let π be the image of u by F, *i.e.* the unique word π satisfying $\binom{u}{\pi} \in F$. The word π is necessarily of the shape $\square^\sharp \cdot \triangle_{\sigma(0)} \cdot \square^\sharp \cdots \square^\sharp \cdot \triangle_{\sigma(p-1)} \cdot \square^\sharp$, where σ is a permutation of $p = \{0, \ldots, p{-}1\}$.

Let $e = \alpha(\binom{y}{\square}^\sharp) \in S$. By the definition of $\sharp(S)$, e is idempotent. Consider β the homomorphism from words over $\mathbb{A}' := \mathbb{A} \sqcup \{a_i \,|\, i \in p\}$ to S defined by $\beta(a_i) = e \cdot \alpha(\binom{x}{\triangle_{\sigma(i)}}) \cdot e$ for $i \in p$, and $\beta(a) = e$ for $a \in \mathbb{A}$.

Now, consider σ' a second permutation of p, and w the word $w_0 \cdot a_{\sigma'(0)} \cdot w_1 \cdots w_{p-1} \cdot a_{\sigma'(p-1)} \cdot w_p \in \mathbb{A}'^*$, where the w_i's are arbitrary words over $\mathbb{A}$. Because e is idempotent, we know that

$$\beta(w) = \alpha(\binom{y}{\square}^\sharp \cdot \binom{x}{\triangle_{\sigma'(\sigma(0))}} \cdot \binom{y}{\square}^\sharp \cdots \binom{y}{\square}^\sharp \cdot \binom{x}{\triangle_{\sigma'(\sigma(p-1))}} \cdot \binom{y}{\square}^\sharp).$$

Since F is a uniformisation, $\beta(w) \in T$ if and only if σ' is the identity. Therefore,

$$\beta^{-1}(T) \cap \bigcap_{i \in p} [\exists^{=1}a_i]_{\mathbb{A}'} = \mathbb{A}^* \cdot a_0 \cdot \mathbb{A}^* \cdots \mathbb{A}^* \cdot a_{p-1} \cdot \mathbb{A}^* = L.$$

Using Lemma 10, each of the languages $[\exists^{=1}\, a_i]_{\mathbb{A}'}$ is in $\mathbf{L}$. Because $\mathbf{L}$ is closed under intersections, we can conclude that L is in $\mathbf{L}$. ◀

3.5 Subsequences

Now we need to strengthen the above lemma, to be able to compare the positions of not necessarily distinct letters. This ability is expressed by the following lemma:

▶ **Lemma 15.** *Let $\mathbb{A}$ be an alphabet, and let $a_0, \ldots, a_{p-1}$ be letters of $\mathbb{A}$, with $p \geq 1$. Then the language $\mathbb{A}^* \cdot a_0 \cdot \mathbb{A}^* \cdots \mathbb{A}^* \cdot a_{p-1} \cdot \mathbb{A}^*$ is in $\mathbf{L}$. We denote this language by $[\exists a_0 {<} a_1 {<} \ldots {<} a_{p-1}]_{\mathbb{A}}$.*

Again, this lemma is equivalent to saying that $\mathcal{B}\Sigma_1[<] \subseteq \mathbf{L}$ (*i.e.* Boolean combinations of existential First-Order sentences with the order) or equivalently that the pseudo-variety of J-trivial semigroups $\mathbf{J}$ is contained in $\mathbf{S}$.

Let $\mathbb{B} := \{\triangle_0, \ldots, \triangle_{p-1}, \square\}$ be an alphabet containing $p{+}1$ pairwise distinct symbols. First, we consider the following relation:

$$R = \binom{\mathbb{A}}{\square}^* \cdot \binom{a_0}{\triangle_0} \cdot \binom{\mathbb{A}}{\square}^* \cdots \binom{\mathbb{A}}{\square}^* \cdot \binom{a_{p-1}}{\triangle_{p-1}} \cdot \binom{\mathbb{A}}{\square}^*$$

It is immediate to see that $\Pi(R)$ is exactly $[\exists a_0 {<} \ldots {<} a_{p-1}]_{\mathbb{A}}$. Consider the relations $R_1 := R \cdot \binom{\bullet}{\bullet} \cdot \binom{\mathbb{A}}{\square}^*$ and $R_2 := \binom{\mathbb{A}}{\square}^* \cdot \binom{\bullet}{\bullet} \cdot R$, where $\bullet$ is a fresh letter (*i.e.* not in $\mathbb{A}$ nor in $\mathbb{B}$).

To conclude the proof of Lemma 15, we will use a fairly technical fact. It may be seen as an abstract generalisation of the technique used in the proof of Lemma 3 in [12].

▶ **Fact 16.** *Let R be a relation over a product alphabet $\mathbb{A} \times \mathbb{B}$, i.e. $R \subseteq \big(\mathbb{A} \times \mathbb{B}\big)^+$. Assume that $\bullet$, $\square$ are two symbols, with $\bullet \notin \mathbb{A}$. Define $R_1 := R \cdot \binom{\bullet}{\bullet} \cdot \binom{\mathbb{A}}{\square}^*$, $R_2 := \binom{\mathbb{A}}{\square}^* \cdot \binom{\bullet}{\bullet} \cdot R$, and $P := R_1 \cup R_2$. If P is in $\mathbf{L}$ then $\Pi(R)$ is in $\mathbf{L}$.*

Proof. Let $F \in \mathbf{L}$ be a uniformisation of the above relation P, and let $\langle S, \alpha, T\rangle$ be a triple recognising F, with $S \in \mathbf{S}$.

Let β be the homomorphism from $\mathbb{A}^+$ to S defined by $\beta(a) = \alpha(\binom{a}{\square})$, for all $a \in \mathbb{A}$. Put $L := \Pi(R)$. Notice that if for all words w_1, w_2 in $\mathbb{A}^+$, the equality $\beta(w_1) = \beta(w_2)$ implies the equivalence $w_1 \in L \Leftrightarrow w_2 \in L$, then in fact L is in $\mathbf{L}$, because $\beta^{-1}\big(\beta(L)\big) = L \in \mathbf{L}$.

We show now that this implication holds for all w_1, w_2. Suppose that there exist $w_1 \in L$ and $w_2 \in L^{\mathrm{c}}$ such that $\beta(w_1) = \beta(w_2)$, in order to provide a contradiction.

Let w be the word $w_1 \cdot \bullet \cdot w_1$, over the alphabet $\mathbb{A} \sqcup \{\bullet\}$. This word w is in $\Pi(R_1) \cap \Pi(R_2)$, let π the unique word over $\mathbb{B} \cup \{\bullet, \square\}$ such that $\binom{w}{\pi} \in F$.

We suppose for instance that $\binom{w}{\pi} \in R_1$ (the case $\binom{w}{\pi} \in R_2$ is symmetric). Because $\bullet \notin \mathbb{A}$, π is necessarily of the shape $\pi_1 \cdot \bullet \cdot \square^{|w_1|}$, with $\pi_1 \in \mathbb{B}^{|w_1|}$.

Let now w' be the word $w_2 \cdot \bullet \cdot w_1$ and let π' be the unique word such that $\binom{w'}{\pi'} \in F$. Again, π' is of the shape $\square^{|w_2|} \cdot \bullet \cdot \pi_1'$, with $\pi_1' \in \mathbb{B}^{|w_1|}$. Since $\beta(w_2) = \beta(w_1)$, we know that $\alpha(\binom{w'}{\square^{|w_2|} \cdot \bullet \cdot \pi_1'}) = \alpha(\binom{w}{\square^{|w_1|} \cdot \bullet \cdot \pi_1'})$. The latter value does not belong to T because $\binom{w}{\square^{|w_1|} \cdot \bullet \cdot \pi_1'} \notin F$ – we know that F is a uniformisation and $\pi \neq \square^{|w_1|} \cdot \bullet \cdot \pi_1'$. This means that $\binom{w'}{\pi'}$ is not in F, contradicting the assumption, and concluding the proof of this fact. ◀

Now we go back to the proof of Lemma 15. The letters $\binom{a_0}{\triangle_0}, \ldots, \binom{a_{p-1}}{\triangle_{p-1}}, \binom{\bullet}{\bullet}$ are all pairwise distinct, and none of them is in the alphabet $\mathbb{A} \times \{\square\}$. Therefore, Lemma 14 tells us that R_1 and R_2 are in $\mathbf{L}$. This means that $P := R_1 \cup R_2$ is in $\mathbf{L}$, and we can conclude with Fact 16 that $[\exists a_0 {<} \ldots {<} a_{p-1}]_{\mathbb{A}} = \Pi(R)$ is in $\mathbf{L}$.

▶ **Corollary 17.** *Let $\mathbb{A}_0, \ldots, \mathbb{A}_{p-1}$ be pairwise disjoint alphabets contained in $\mathbb{A}$. Then the language $L = \mathbb{A}_0^* \cdot \mathbb{A}_1^* \ldots \mathbb{A}_{p-1}^* \setminus \{\epsilon\}$ is in L.*

Proof. It is enough to observe that $L = \bigcap_{i \in j \in p} \bigcap_{a_i \in \mathbb{A}_i} \bigcap_{a_j \in \mathbb{A}_j} [\exists a_j < a_i]_{\mathbb{A}}^{\mathrm{c}}$ (where $\mathbb{A}$ is the union of the $\mathbb{A}_i$'s). ◀

3.6 Polynomials

We will now prove a variant of Lemma 15 for *polynomials.* A *monomial* is a language of the shape $L_0 \cdot L_1 \cdots L_{p-1}$, where each L_i is either of the form $\mathbb{A}_i^*$, or a set of single-letter words over $\mathbb{A}_i$, and such that at least one of the L_i's is of the latter kind. An example of a monomial is the language

$$\{a,b\}^* \cdot \{x,y\} \cdot \{x\}^*.$$

Notice that the alphabets $\mathbb{A}_i$ are not required to be pairwise disjoint in that definition. A *polynomial* is a finite union of monomials. This section is devoted to a proof of the following lemma.

▶ **Lemma 18.** *Any polynomial is in* $\mathbf{L}$.

First notice that $\mathbf{L}$ is closed under unions and therefore it is enough to prove the lemma for monomials. Consider a monomial L over an alphabet $\mathbb{A}$, *i.e.* $L = \mathbb{A}_0^{\xi_0} \cdot \mathbb{A}_1^{\xi_1} \cdots \mathbb{A}_{p-1}^{\xi_{p-1}}$, where each ξ_i is either $*$ or 1 (because $\mathbb{A}_i^1 = \mathbb{A}_i$). Take $\mathbb{A}' := \mathbb{A} \sqcup \{\bullet\}$, $\mathbb{B} := \{\Delta_0, \ldots, \Delta_{p-1}, \bullet, \square\}$. Let $R := \binom{\mathbb{A}_0}{\Delta_0}^{\xi_0} \cdots \binom{\mathbb{A}_{p-1}}{\Delta_{p-1}}^{\xi_{p-1}}$, $R_1 := R \cdot \binom{\bullet}{\bullet} \cdot \binom{\mathbb{A}}{\square}^*$, and $R_2 := \binom{\mathbb{A}}{\square}^* \cdot \binom{\bullet}{\bullet} \cdot R$.

Notice that

$$R_1 = \binom{\mathbb{A}_0}{\Delta_0}^* \cdots \binom{\mathbb{A}_{p-1}}{\Delta_{p-1}}^* \cdot \binom{\bullet}{\bullet}^* \cdot \binom{\mathbb{A}}{\square}^* \;\cap\; \Big[\exists^{=1}\binom{\bullet}{\bullet}\Big]_{\mathbb{A}' \times \mathbb{B}} \;\cap\; \bigcap_{i \in p} \Xi_i, \tag{3}$$

where each Ξ_i is either: $\big(\mathbb{A}' \times \mathbb{B}\big)^+$ if $\xi_i = *$; or $\big[\exists^{=1} \mathbb{A}_i \times \{\Delta_i\}\big]_{\mathbb{A}' \times \mathbb{B}}$ if $\xi_i = 1$. Now, the first ingredient on the right-hand side of (3) is as in Corollary 17 and thus belongs to $\mathbf{L}$. The other ingredients also belong to $\mathbf{L}$, see Lemma 10, and Corollary 13. Similarly we know that $R_2 \in \mathbf{L}$. Thus, Fact 16 implies that $L = \Pi(R) \in \mathbf{L}$, which concludes the proof of Lemma 18.

▶ Remark 19. The family of polynomials is closed under union and concatenation.

3.7 Semigroups

We can now conclude the proof of Theorem 2. Let L be a regular language over some alphabet $\mathbb{A}$, that is recognised by a tuple $\langle S, \alpha, T\rangle$ with a finite semigroup S that may *a priori* not belong to $\mathbf{S}$. Our aim is to prove that $L \in \mathbf{L}$.

Consider a word $\binom{w}{\sigma} \in \big(\mathbb{A} \times S\big)^+$ of length n. We say that such a word is an *evaluation* if for every $i \in n$ we have $\sigma(i) = \alpha\big(w(0) \cdots w(i)\big)$. Notice that in that case $w \in L$ if and only if $\sigma(n{-}1) \in T$. Let E be the set of words $\binom{w}{\sigma} \in \big(\mathbb{A} \times S\big)^+$ that are evaluations and $\sigma(n{-}1) \in T$.

▷ Claim 20. Using the above notions, we have $\Pi(E) = L$.

Proof. Every word $w \in \mathbb{A}^n$ with $n \geq 1$ admits a unique word $\sigma \in S^n$ such that $\binom{w}{\sigma}$ is an evaluation. In that case $\alpha(w) = \sigma(n{-}1)$. Thus, $w \in L$ iff. $\sigma(n{-}1) \in T$ iff. $\binom{w}{\sigma} \in E$. ◁

Our aim is to show that a variant of the set of evaluations E belongs to $\mathbf{L}$ and then invoke Fact 16 to project away the S coordinate of the evaluations.

Consider $a, b \in \mathbb{A}$ and $r, s \in S$ and define

$$I_{a,r} := \binom{a}{r} \cdot \binom{\mathbb{A}}{S}^*, \quad M_{a,r,b,s} := \binom{\mathbb{A}}{S}^* \cdot \binom{a}{r} \cdot \binom{b}{s} \cdot \binom{\mathbb{A}}{S}^*, \quad F_{a,r} := \binom{\mathbb{A}}{S}^* \cdot \binom{a}{r}.$$

Let W be the union of the languages: $I_{a,r}$ ranging over $a \in \mathbb{A}$, $r \in S$ such that $\alpha(a) \neq r$; $M_{a,r,b,s}$ ranging over those $a, b \in \mathbb{A}$, $r, s \in S$ such that $r \cdot \alpha(b) \neq s$; and $F_{a,r}$ ranging over $r \notin T$. Notice that W defined that way is a polynomial.

▷ Claim 21. The complement of W equals E.

Proof. Clearly $E \cap W = \emptyset$. Thus, it is enough to prove that if $\binom{w}{\sigma} \notin W$ then $\binom{w}{\sigma} \in E$. Let $n = |w| = |\sigma|$. Since $\binom{w}{\sigma} \notin W$, we know that $\sigma(n{-}1) \in T$ (see the languages $F_{a,r}$), thus it is enough to show that $\binom{w}{\sigma}$ is an evaluation. It is done inductively, for $i = 0, 1, \ldots, n{-}1$. The fact that $\sigma(0) = \alpha\big(w(0)\big)$ follows from the assumption that $\binom{w}{\sigma} \notin W$ (see the languages $I_{a,r}$).

Take $i < n{-}1$ and assume that $\sigma(i) = \alpha\big(w(0) \cdots w(i)\big)$. Observe that $\sigma(i{+}1)$ must equal $\sigma(i) \cdot \alpha\big(w(i{+}1)\big)$ (see the languages $M_{a,r,b,s}$). Thus, $\sigma(i{+}1) = \alpha\big(w(0) \cdots w(i{+}1)\big)$. ◁

Consider fresh letters • and •, and the alphabets $\mathbb{A}' := \mathbb{A} \sqcup \{\bullet\}$, $S' := S \sqcup \{\bullet, \Delta\}$. Let:

$$R_1 := W \cdot \binom{\bullet}{\bullet} \cdot \binom{\mathbb{A}}{\Delta}^*, \qquad R_2 := \binom{\mathbb{A}}{\Delta}^* \cdot \binom{\bullet}{\bullet} \cdot W,$$
$$R_1' := R_1^{\mathrm{c}} \;\cap\; \binom{\mathbb{A}}{S}^* \cdot \binom{\bullet}{\bullet} \cdot \binom{\mathbb{A}}{\Delta}^*, \quad R_2' := R_2^{\mathrm{c}} \;\cap\; \binom{\mathbb{A}}{\Delta}^* \cdot \binom{\bullet}{\bullet} \cdot \binom{\mathbb{A}}{S}^*, \quad P := R_1' \cup R_2'.$$

Notice that both R_1 and R_2 are polynomials (see Remark 19) and therefore all the five relations defined above belong to $\mathbf{L}$.

▷ Claim 22. Using the above notions, we have

$$R_1' = E \cdot \binom{\bullet}{\bullet} \cdot \binom{\mathbb{A}}{\Delta}^*, \quad R_2' = \binom{\mathbb{A}}{\Delta}^* \cdot \binom{\bullet}{\bullet} \cdot E.$$

Proof. These equalities follow directly from the definition and Claim 21. ◁

Therefore, Fact 16 applied to P guarantees that $\Pi(E) \in \mathbf{L}$. Thus, by Claim 20 we know that $L \in \mathbf{L}$. This concludes the proof of Theorem 2.

4 Conclusions

The main result of this work shows that among pseudo-varieties of semigroups, only **REG** =**MSO** is strong enough to have the uniformisation property over finite words. It seems that exactly the same techniques work also for infinite words and finite trees, however the technical details are more involved there. This means that, to be able to choose witnesses in a definable way, one needs to have access to the unrestricted quantification over these witnesses (*i.e. monadic* quantifiers).

The actual arguments used in the presented proof are rather direct: they boil down to finding an appropriate relation R that is definable in the considered formalism, such that any uniformisation of R must provide some added expressive power. However, the difficulty of that reasoning lies in the deliberate design of the relations R. From this perspective, the proof can be read as a collection of instances showing how uniformisability (or generally ability to choose witnesses) leads to an increased expressive power.

The results of this work are in a sense negative, showing that all formalisms below **MSO** do not admit uniformisation. However, still some relations can be uniformised within the limited expressive power. This leads to the following decision problem:

▶ **Problem 23.** *Given a regular language R over a product alphabet $\mathbb{A} \times \mathbb{B}$, decide if R admits an* **FO**$[<]$*-definable uniformisation.*

Note that even if R itself is not **FO**$[<]$-definable, it might be the case that there is an **FO**$[<]$-definable uniformisation of R. The status of this problem is open at the moment.

References

1 Mikołaj Bojańczyk. Effective characterizations of tree logics. In *PODS*, pages 53–66, 2008.

2 Julius Richard Büchi. Weak Second-Order Arithmetic and Finite Automata. *Mathematical Logic Quarterly*, 6(1–6):66–92, 1960.

3 Julius Richard Büchi and Lawrence H. Landweber. Solving Sequential Conditions by Finite-State Strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

4 Arnaud Carayol and Christof Löding. MSO on the infinite binary tree: Choice and order. In *CSL*, pages 161–176, 2007.

5 Volker Diekert, Paul Gastin, and Manfred Kufleitner. A Survey on Small Fragments of First-Order Logic over Finite Words. *Int. J. Found. Comput. Sci.*, 19(3):513–548, 2008. doi:10.1142/S0129054108005802.

6 Samuel Eilenberg. *Automata, languages, and machines.* Pure and Applied Mathematics. Elsevier Science, 1974.

7 Calvin C. Elgot. Decision Problems of Finite Automata Design and Related Arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51, 1961.

8 Yuri Gurevich and Saharon Shelah. Rabin's uniformization problem. *Journal of Symbolic Logic*, 48(4):1105–1119, 1983.

9 Alexander Kechris. *Classical descriptive set theory.* Springer-Verlag, New York, 1995.

10 Shmuel Lifsches and Saharon Shelah. Uniformization and Skolem Functions in the Class of Trees. *Journal of Symbolic Logic*, 63(1):103–127, 1998.

11 Robert McNaughton and Seymour Papert. *Counter-free automata.* M.I.T. Press research monographs. M.I.T. Press, 1971.

12 Vincent Michielini. Uniformization Problem for Variants of First Order Logic over Finite Words. In *DLT*, pages 516–528, 2018.

13 Dominique Perrin and Jean-Éric Pin. *Infinite Words: Automata, Semigroups, Logic and Games.* Elsevier, 2004.

14 Jean-Éric Pin and Howard Straubing. Some results on C-varieties. *ITA*, 39(1):239–262, 2005. doi:10.1051/ita:2005014.

15 Michael Oser Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

16 Michael Oser Rabin and Dana Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2):114–125, April 1959.

17 Alexander Rabinovich. On decidability of monadic logic of order over the naturals extended by monadic predicates. *Information and Computation*, 205(6):870–889, 2007.

18 Jan Reiterman. The Birkhoff theorem for finite algebras. *Algebra Universalis*, 14(1):1–10, 1982.

19 Marcel Paul Schützenberger. On Finite Monoids Having Only Trivial Subgroups. *Information and Control*, 8(2):190–194, 1965.

20 Dirk Siefkes. The recursive sets in certain monadic second order fragments of arithmetic. *Arch. Math. Logik*, 17(1–2):71–80, 1975.

21 Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222, 1975.

22 Imre Simon. Word Ramsey theorems. In B. Bollobas, editor, *Graph Theory and Combinatorics*, pages 283–291. Academic Press, London, 1984.

23 Denis Thérien and Thomas Wilke. Over Words, Two Variables Are as Powerful as One Quantifier Alternation. In *STOC*, pages 234–240, 1998.

24 Wolfgang Thomas. Star-Free Regular Sets of omega-Sequences. *Information and Control*, 42(2):148–156, 1979.

25 Boris A. Trakhtenbrot. Finite automata and the monadic predicate calculus. *Siberian Mathematical Journal*, 3(1):103–131, 1962.

New Pumping Technique for 2-Dimensional VASS

Wojciech Czerwiński
University of Warsaw, Poland
wczerwin@mimuw.edu.pl

Sławomir Lasota
University of Warsaw, Poland
sl@mimuw.edu.pl

Christof Löding
RWTH Aachen, Germany
loeding@informatik.rwth-aachen.de

Radosław Piórkowski
University of Warsaw, Poland
r.piorkowski@mimuw.edu.pl

Abstract

We propose a new pumping technique for 2-dimensional vector addition systems with states (2-VASS) building on natural geometric properties of runs. We illustrate its applicability by reproving an exponential bound on the length of the shortest accepting run, and by proving a new pumping lemma for languages of 2-VASS. The technique is expected to be useful for settling questions concerning languages of 2-VASS, e.g., for establishing decidability status of the regular separability problem.

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

Keywords and phrases vector addition systems with states, pumping, decidability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.62

Related Version `https://arxiv.org/abs/1906.10494`

Funding This research was supported by Polish NCN grants “Separation problems in automata theory” (2016/21/D/ST6/01376), “Automatic analysis of concurrent systems” (2017/27/B/ST6/02093) and “Symbolic computations on first-order definable objects” (2016/21/B/ST6/01505).

1 Introduction

Vector addition systems [8] are a widely accepted model of concurrency equivalent to Petri nets. Another equivalent model, called vector addition systems with states (VASS) [7], is an extension of finite automata with integer counters, on which the transitions can perform operations of increment or decrement (but no zero tests), with the proviso that counter values are 0 initially and must stay non-negative along a run. The number of counters d defines the *dimension* of a VASS. For brevity, we call a VASS of dimension d a d-VASS. Formally, every transition of a d-VASS V has adjoined a vector $v \in \mathbb{Z}^d$ describing the effect of executing this transition on counter values; thus a transition is a triple $(q, v, q') \in Q \times \mathbb{Z}^d \times Q$, where Q is the set of control states of V. A finite *path*, i.e., a sequence of transitions of the form $\pi = (q_0, v_1, q_1), (q_1, v_2, q_2), \ldots, (q_{n-1}, v_n, q_n)$, induces a run if the counter values stay non-negative, i.e., $v_1 + \ldots + v_i \in \mathbb{N}^d$ for every i.

In this paper we concentrate on *pumping*, i.e., techniques exploiting repetitions of states in runs. Pumping is an ubiquitous phenomenon which typically provides valuable tools in proving short run properties, or showing language inexpressibility results. It seems to be particularly relevant in case of VASS, as even the core of the seminal decision procedure for

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 62; pp. 62:1–62:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the reachability problem in VASS by Mayr and Kosaraju [11, 9] is fundamentally based on pumping: briefly speaking, the decision procedure decomposes a VASS into a finite number of VASS, each of them admitting a property that every path can be pumped up so that it induces a run. Pumping techniques are used even more explicitly when dealing with subclasses of VASS of bounded dimension. The PSpace upper bound for the reachability problem in 2-VASS [2] relies on various un-pumping transformations of an original run, leading to a simple run of at most exponential length, in the form of a short path with adjoined short disjoint cycles. A smart surgery on those simple runs was also used to obtain a stronger upper bound (NL) in case when the transition effects are represented in unary [6]. Un-pumping is also used in [3] to provide a quadratic bound on the length of the shortest run for 1-VASS, also known as one counter automata without zero tests, and for unrestricted one counter automata. See also [1, 10] for pumping techniques in one counter automata.

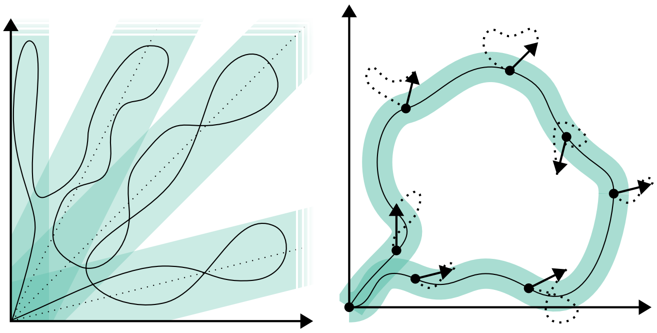


Figure 1 Thin (above) and thick run (below). Points correspond to counter values, and control states along a run are ignored.

Contribution. The above-mentioned techniques are mostly oriented towards reachable sets, and henceforth may ignore certain runs as long as the reachable set is preserved. In consequence, they are not very helpful in solving decision problems formulated in terms of the whole language accepted by a VASS, like the regular separability problem (cf. the discussion below). Our primary objective is to design a pumping infrastructure applicable to *every* run of a 2-VASS. Therefore, as our main technical contribution we perform a thorough classification of runs, in the form of a dichotomy (see the illustrations on the right): for every run π of a 2-VASS, whose initial and final values of both counters are 0,

- either π is *thin*, by which we mean that the counter values along the run stay within belts, whose direction and width are all bounded polynomially in the number of states and the largest absolute value of vectors of the 2-VASS;
- or π is *thick*, by which we mean that a number of cycles is enabled along the run, the effect vectors of these cycles span (slightly oversimplifying) the whole plane, and furthermore the lengths of cycles and the extremal factors of π are all bounded polynomially in M and exponentially in n. (For the sake of simplicity some details are omitted here; the fully precise statement of the dichotomy is Theorem 3.1 in Section 3).

The dichotomy immediately entails a pumping lemma for 2-VASS by using, essentially, the pumping scheme of 1-VASS in case of thin runs, and the cycles enabled along a run in case of thick runs (cf. Theorem 4.1). As a more subtle application of the dichotomy, we derive an alternative proof of the exponential run property (shown originally in [2]), which immediately implies PSpace-membership of the reachability problem (cf. Theorem 4.2).

Further applications. We envisage other possible applications of the dichotomy. One important case can be the regular separability problem: given two *labeled* 2-VASS V_1 and V_2, decide if there is a regular language separating languages of V_1 and V_2, i.e., including one of them and disjoint from the other. The problem is decidable in PSpace for 1-VASS [5] while the decidability status for 2-VASS is still open. A cornerstone of the decision procedure of [5] is a well-behaved over-approximation of a language of a 1-VASS V by a sequence of regular languages $(V_n)_{n\in\mathbb{N}}$, where the precision of approximation increases with increasing n. In case of 1-VASS, the language V_n is obtained by abstraction of V *modulo* n; on the other hand, as argued in [5], the very same approach necessarily fails for dimensions larger than 1. It seems that our dichotomy classification of runs of a 2-VASS prepares the ground for the right definition of abstraction V_n modulo n. Indeed, intuitively speaking, as long as the run stays within belts, 1-dimensional counting modulo n along the direction of a belt is sufficient; otherwise, a 2-dimensional abstraction modulo n can be applied as soon as a sufficient number of pumpable cycles has been identified along a run.

As our approach builds on natural geometric properties of runs, we believe that it can be generalized to dimensions larger than 2. However, one should not expect efficient length bounds from this generalization itself, as already in dimension 3 the prefix of a run preceding the first pumpable cycle has non-elementary length (the length can be as large as tower of n exponentials in the composition of n copies of the Hopcroft and Pansiot example [7]).

2 Preliminaries

2-dimensional vector addition systems with states. We use standard symbols $\mathbb{Q}, \mathbb{Z}, \mathbb{N}$ for the sets of rationals, integers, and non-negative integers, respectively. Whenever convenient we use subscripts to specify subsets, e.g., $\mathbb{Q}_{\geq 0}$ for non-negative rationals. We refer to elements of $\mathbb{Z}^2$ briefly as *vectors*. *Non-negative* vectors are elements of $\mathbb{N}^2$, and *positive* vectors are elements of $\mathbb{Z}^2_{>0}$. A vector with only non-negative coordinates and at least one positive coordinate is called *semi-positive*; it is either positive, or *vertical* of the form $(0, a)$, or *horizontal* of the form $(a, 0)$, for $a \in \mathbb{Z}_{>0}$.

A 2-dimensional *vector addition system with states* (2-VASS) V consists of a finite set of control states Q and a finite set of transitions $T \subseteq Q \times \mathbb{Z}^2 \times Q$. We refer to the vector v as the *effect* of a transition (p, v, q). A *path* in V from control state p to q is a sequence of transitions $\pi = (q_0, v_1, q_1), (q_1, v_2, q_2), \ldots, (q_{n-1}, v_n, q_n) \in T^*$ where $p = q_0$ and $q = q_n$; it is called a *cycle* whenever the starting and ending control states coincide ($q_0 = q_n$). The *effect* of a path is defined as $\mathrm{eff}(\pi) = v_1 + \ldots + v_n \in \mathbb{Z}^2$, and its *length* is n. A cycle is called non-negative, semi-positive or positive, if its effect is so.

A *configuration* of V is an element of $\mathrm{Conf} = Q \times \mathbb{N}^2$. A transition $t = (p, v, q)$ is *enabled* in a configuration $c = (p', u)$ if $p = p'$ and $u + v \in \mathbb{N}^2$. Analogously, a path π is enabled in a configuration $c = (p', u)$ if $q_0 = p'$ and $u_i = u + v_1 + \ldots + v_i \in \mathbb{N}^2$ for every i. In such case we say that π induces a *run* of the form

$$\rho = (c_0, t_1, c_1), (c_1, t_2, c_2), \ldots, (c_{n-1}, t_n, c_n) \in (\mathrm{Conf} \times T \times \mathrm{Conf})^*$$

with intermediate configurations $c_i = (q_i, u_i)$, from the *source* configuration $\mathrm{src}(\rho) = c_0$ to the *target* one $\mathrm{trg}(\rho) = c_n$. If the source configuration c_0 is clear from the context, we do not distinguish between a path enabled in c_0 and a run with source c_0, and simply say that the path *is* the run. A $(0, 0)$-*run* is a run whose source and target are $(0, 0)$-*configurations*, i.e., a configuration whose vector is $(0, 0)$.

We will sometimes relax the non-negativeness requirement on some coordinates: For $j \in \{1, 2\}$, we say that a path π is $\{j\}$-*enabled* in a configuration $c = (p', u)$ if $q_0 = p'$ and $(u + v_1 + \ldots + v_i)[j] \in \mathbb{N}$ for every i. We also say that π is $\emptyset$-*enabled* in c if just $q_0 = p'$.

The *reversal* of a 2-VASS $V = (Q, T)$, denoted $\mathrm{rev}(V)$, is a 2-VASS with the same control states and with transitions $\{(q, -v, p) \mid (p, v, q) \in T\}$. We sometimes speak of the reversal $\mathrm{rev}(\rho)$ of a run ρ of V, implicitly meaning a run in the reversal of V.

As the *norm* of $v = (v_1, v_2) \in \mathbb{Q}^2$, we take the largest of absolute values of v_1 and v_2, $\|v\| := \max\{|v_1|, |v_2|\}$. By the norm of a configuration $c = (q, v)$ we mean the norm of its vector v, and by the norm $\|V\|$ of a 2-VASS V we mean the largest among norms of effects of transitions.

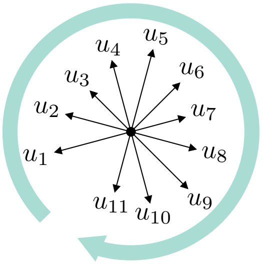


Figure 2 Above $u_1 \circlearrowright u_2 \circlearrowright \ldots \circlearrowright u_{11} \circlearrowright u_1$. Also, $u_4 \circlearrowright u_9$, but $u_4 \not\circlearrowright u_{11}$ and $u_{11} \circlearrowright u_4$. Pairs of vectors u_i, u_{i+6} are contralinear, for $i = 1, \ldots, 5$.

Sequential cones. For a vector $v \in \mathbb{Z}^2$, define the half-line induced by v as $\ell_v := \mathbb{Q}_{\geq 0} \cdot v = \{\alpha v \mid \alpha \in \mathbb{Q}_{\geq 0}\}$. We call two vectors v, w *colinear* if $\ell_v = \ell_w$, and *contralinear* if $\ell_v = \ell_{-w}$. For two vectors $u, v \in \mathbb{Z}^2 \setminus \{(0, 0)\}$, define the *angle* $\measuredangle[u, v] \subseteq \mathbb{Q}^2$ as the union of all half-lines which lie clock-wise between ℓ_u and ℓ_v, including the two half-lines themselves. In particular, $\measuredangle[v, v] = \ell_v$. Analogously we define the sets $\measuredangle[u, v)$, $\measuredangle(u, v]$ and $\measuredangle(u, v)$ which exclude one or both of the half-lines. We refer to an angle of the form $\measuredangle[v, -v]$ as *half-plane*. We write $v \circlearrowright u$ when $u \in \measuredangle(v, -v)$, i.e., u is oriented clock-wise with respect to v (see Figure 2 for an illustration). Note that $\circlearrowright$ defines a total order on pairwise non-colinear non-negative vectors.

By the *cone* of a finite set of vectors $\{v_1, \ldots, v_k\} \subseteq \mathbb{Z}^2$ we mean the set of all non-negative rational linear combinations of these vectors:

$$\textsc{Cone}(v_1, \ldots, v_k) := \{\Sigma_{j=1}^{k} a_j v_j \in \mathbb{Q}^2 \mid a_1, \ldots, a_k \in \mathbb{Q}_{\geq 0}\}.$$

We call the cone of a single vector $\textsc{Cone}(v) = \ell_v$ *trivial*, and the cone of zero vectors $\textsc{Cone}(\emptyset) = \{(0, 0)\}$ *degenerate*. Two non-zero vectors v_1 and v_2 can be in four distinct relations: (I) they are colinear, (II) they are contralinear, (III) $v_1 \circlearrowright v_2$ and hence $\textsc{Cone}(v_1, v_2) = \measuredangle[v_1, v_2]$, (IV) $v_2 \circlearrowright v_1$ and hence $\textsc{Cone}(v_1, v_2) = \measuredangle[v_2, v_1]$.

▶ **Lemma 2.1.** *Every cone either equals the whole plane $\mathbb{Q}^2$, or is included in some half-plane.*

Proof. Assume, w.l.o.g. that the vectors $v_1, \ldots, v_k$ are non-zero and include no colinear pair. Suppose there is a contralinear pair v_i, v_j among $v_1, \ldots, v_k$. If all other vectors v_h satisfy $v_i \circlearrowright v_h \circlearrowright v_j$ then $\textsc{Cone}(v_1, \ldots, v_k)$ is included in the half-plane $\measuredangle[v_i, v_j]$. Otherwise $\textsc{Cone}(v_1, \ldots, v_k)$ is the whole plane.

Now suppose there is no contralinear pair among $v_1, \ldots, v_k$. If some three v_i, v_j, v_h of them satisfy $v_i \circlearrowright v_j \circlearrowright v_h \circlearrowright v_i$ then $\textsc{Cone}(v_1, \ldots, v_k)$ includes the three angles $\measuredangle[v_i, v_j]$, $\measuredangle[v_j, v_h]$ and $\measuredangle[v_h, v_i]$, the union of which is the whole plane. Otherwise, the relation $\circlearrowright$ is transitive and hence defines a (strict) total order on $\{v_1, \ldots, v_k\}$. The minimal and maximal element v_i and v_j w.r.t. the order satisfy $v_i \circlearrowleft v_j$, and hence $\textsc{Cone}(v_1, \ldots, v_k) = \measuredangle[v_i, v_j]$ is included in the half-plane $\measuredangle[v_i, -v_i]$. ◀

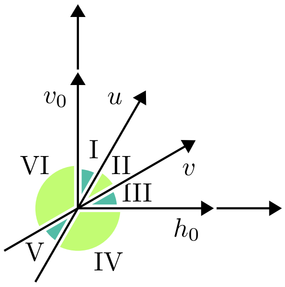
The *sequential cone* of vectors $v_1, \ldots, v_k \in \mathbb{Z}^2$ imposes additional non-negativeness conditions, namely for every i, the partial sum $a_1 v_1 + \ldots + a_i v_i$ must be non-negative (this is required later, when pumping cycles in a run whose effects are $v_1, \ldots, v_k$ in that order):

$$\textsc{SeqCone}(v_1, \ldots, v_k) := \{\Sigma_{j=1}^k a_j v_j \in \mathbb{Q}^2_{\geq 0} \mid a_1, \ldots, a_k \in \mathbb{Q}_{\geq 0}, \forall_i \ \Sigma_{j=1}^i a_j v_j \in \mathbb{Q}^2_{\geq 0}\}.$$

Note that v_1 may be assumed w.l.o.g. to be semi-positive, but other vectors v_i are not necessarily non-negative; and that every sequential cone is a subset of the non-negative orthant $\mathbb{Q}^2_{\geq 0}$. Importantly, contrarily to cones, the order of vectors $v_1, \ldots, v_k$ matters for sequential cones. In fact, sequential cones are just convenient syntactic sugar for cones of pairs of non-negative vectors:

▶ **Lemma 2.2.** *For all vectors $v_1, \ldots, v_k$, the sequential cone $\textsc{SeqCone}(v_1, \ldots, v_k)$, if not degenerate, equals $\textsc{Cone}(u, v)$, for two non-negative vectors u, v, and each of them either belongs to $\{v_1, \ldots, v_k\}$, or is horizontal, or vertical.*

Proof. We proceed by induction on k. For $k = 1$ we have $\textsc{SeqCone}(v_1) = \ell_{v_1} = \textsc{Cone}(v_1, v_1)$. Let v_0 and h_0 denote some fixed vertical and horizontal vector, respectively. For the induction step we assume $\textsc{SeqCone}(v_1, \ldots, v_{k-1}) = \textsc{Cone}(u, v)$ for non-negative vectors u, v; and compute the value of $\textsc{SeqCone}(v_1, \ldots, v_k)$, separately in each of the following distinct cases (assume w.l.o.g. $u \circlearrowleft v$):



$$\textsc{SeqCone}(v_1, \ldots, v_k) = \begin{cases} \textsc{Cone}(v_k, v) & \text{if } v_k \in \measuredangle[v_0, u) \\ \textsc{Cone}(u, v) & \text{if } v_k \in \measuredangle[u, v] \\ \textsc{Cone}(u, v_k) & \text{if } v_k \in \measuredangle(v, h_0] \\ \textsc{Cone}(u, h_0) & \text{if } v_k \in \measuredangle(h_0, -u] \\ \textsc{Cone}(v_0, h_0) & \text{if } v_k \in \measuredangle(-u, -v) \\ \textsc{Cone}(v_0, v) & \text{if } v_k \in \measuredangle[-v, v_0). \end{cases}$$

◀

3 Thin-Thick Dichotomy

The main result of this section (cf. Theorem 3.1 below) classifies $(0, 0)$-runs in a 2-VASS into *thin* and *thick* ones. Throughout this section we consider an arbitrary fixed 2-VASS $V = (Q, T)$.
Let $n = |Q|$ and $M = \|V\|$.

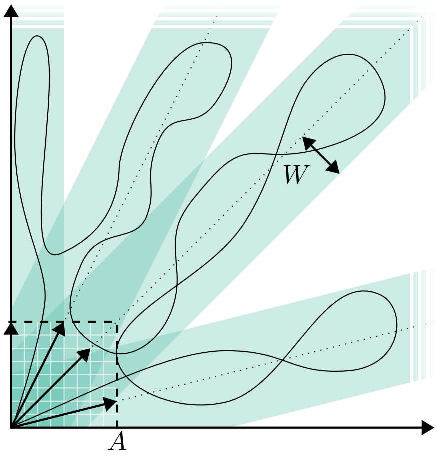


Figure 3 Thin run within belts $B_{v,W}$.

Thin runs. The *belt* of *direction* $v \in \mathbb{N}^2$ and *width* W is the set

$$\mathcal{B}_{v,W} = \{u \in \mathbb{N}^2 \mid \mathrm{dist}(u, \ell_v) \leq W\},$$

where $\mathrm{dist}(u, \ell_v)$ denotes the Euclidean distance between the point u and the half-line ℓ_v. For $A \in \mathbb{N}$, we call $\mathcal{B}_{v,W}$ an A-*belt* if $\|v\| \leq A$ and $W \leq A$. We say that a run ρ of V is A-*thin* if for every configuration c in ρ there exists an A-belt B such that $c \in Q \times B$.

Thick runs. Let $A \in \mathbb{N}$. Four cycles $\pi_1, \pi_2, \pi_3, \pi_4 \in T^*$ are A-*sequentially enabled* in a run ρ if their lengths are at most A, and the run ρ factors into $\rho = \rho_1\, \rho_2\, \rho_3\, \rho_4\, \rho_5$ so that (denote by v_1, v_2, v_3, v_4 the effects of $\pi_1, \pi_2, \pi_3, \pi_4$, respectively):

- The effect v_1 is semi-positive, the cycle π_1 is enabled in $c_1 := \mathrm{trg}(\rho_1)$, and both coordinates are bounded by A along ρ_1.
- If v_1 is positive then π_2 is $\emptyset$-enabled in $c_2 := \mathrm{trg}(\rho_2)$. Otherwise (let j be the coordinate s.t. $v_1[j] = 0$) π_2 is $\{j\}$-enabled in $c_2 := \mathrm{trg}(\rho_2)$, and jth coordinate is bounded by A along ρ_2.
- The cycle π_i is $\emptyset$-enabled in $c_i := \mathrm{trg}(\rho_i)$, for $i = 3, 4$.

We also say that the four vectors v_1, v_2, v_3, v_4 are A-sequentially enabled in ρ, quantifying the cycles existentially. A $(0,0)$-run τ is called A-*thick* if it partitions into $\tau = \rho\, \rho'$ so that

1. some vectors v_1, v_2, v_3, v_4 are A-sequentially enabled in ρ,
2. some vectors v_1', v_2', v_3', v_4' are A-sequentially enabled in $\mathrm{rev}(\rho')$,
3. $\textsc{SeqCone}(v_1, v_2, v_3, v_4) \cap \textsc{SeqCone}(v_1', v_2', v_3', v_4')$ is non-trivial.

Figure 4 illustrates the geometric ideas underlying these three conditions for A-thick runs. Concerning condition 1, a cycle π_1 depicted by a dotted line, with vertical effect v_1, can be used to increase the second (vertical) coordinate arbitrarily, which justifies the relaxed requirement that a cycle π_2 with effect v_2 is only $\{1\}$-enabled. Note that the norm of the configuration enabling π_1, as well as the first coordinate of the configuration enabling π_2, are bounded by A. Concerning condition 2, a cycle π_1' with positive effect v_1' can be used to increase both coordinates arbitrarily; therefore a cycle π_2' with effect v_2' is only required to be $\emptyset$-enabled, and no coordinate of the configuration enabling π_2' is required to be bounded by A. In the illustrated example, vectors v_3' and v_4' are not needed; formally, one can assume $v_2' = v_3' = v_4'$ and $\rho_3' = \rho_4' = \varepsilon$. Condition 3 ensures that the cycles

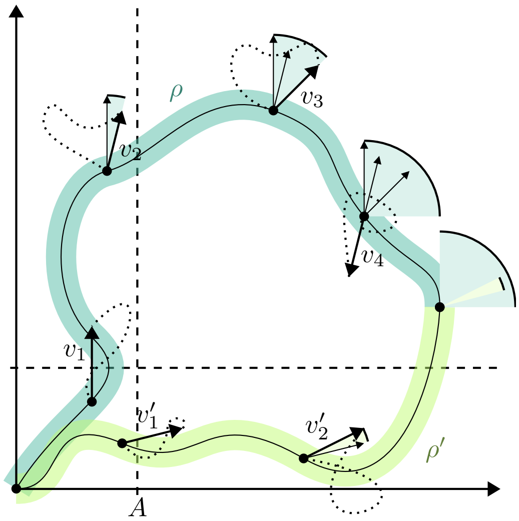


Figure 4 Thick run. Blue angles denote sequential cones $\textsc{SeqCone}(v_1, v_2)$, $\textsc{SeqCone}(v_1, v_2, v_3)$ and $\textsc{SeqCone}(v_1, v_2, v_3, v_4)$, respectively, and green angle denotes $\textsc{SeqCone}(v'_1, v'_2)$.

$\pi_1, \ldots, \pi_4$ and $\pi'_1, \ldots, \pi'_4$ can be pumped such that the pumped versions of ρ and ρ' are still connected. In the illustrated example, observe that $\textsc{SeqCone}(v_1, v_2) \cap \textsc{SeqCone}(v'_1) = \emptyset$. Intuitively, both coordinates in the target of ρ can be increased arbitrarily using v_1 and v_2, and similarly both coordinates of the target of $\text{rev}(\rho')$ can be increased arbitrarily using v'_1, but "directions of increase" are non-crossing. Adding v_3 and v'_2 is not sufficient, as still $\textsc{SeqCone}(v_1, v_2, v_3) \cap \textsc{SeqCone}(v'_1, v'_2) = \emptyset$. When vector v_4 is adjoined, condition 3 holds as $\textsc{SeqCone}(v_1, v_2, v_3, v_4) = \mathbb{Q}^2_{\geq 0}$. Finally, the four vectors are really needed here, e.g., vector v_3 can not be omitted as $\textsc{SeqCone}(v_1, v_2, v_4) = \textsc{SeqCone}(v_1, v_2)$.

Here is the main result of this section:

▶ **Theorem 3.1** (Thin-Thick Dichotomy). *There is a polynomial p such that every $(0,0)$-run in a 2-VASS V is either $p(nM)^n$-thin or $p(nM)^n$-thick.*

For the proof of Theorem 3.1 we need the following core fact (for space reasons, the proof is only in the full version of the paper):

▶ **Lemma 3.2** (Non-negative Cycle Lemma). *There is a polynomial P such that every run ρ in V from a $(0,0)$-configuration to a target configuration of norm larger than $P(nM)^n$, contains a configuration enabling a semi-positive cycle of length at most $P(nM)$.*

Proof of Theorem 3.1. Let P be the polynomial from Lemma 3.2. The polynomial p required in Theorem 3.1 can be chosen arbitrarily as long as $p(x) \geq \sqrt{2} \cdot \big(P(x) + (x+1)^3\big) \cdot x$. for all x; note that the following inequality follows:

$$p(nM)^n \geq \sqrt{2} \cdot \big((P(nM))^n + (nM+1)^3 \big) \cdot nM. \tag{1}$$

In the sequel we deliberately confuse configurations $c = (q, v)$ with *their vectors* v: whenever convenient, we use c to denote the vector v, hoping that this does not lead to any confusion.

Let τ be a $(0,0)$-run of V which is not $p(nM)^n$-thin, i.e., τ contains therefore a configuration t which lies outside of all the $p(nM)^n$-belts. We need to demonstrate points 1–3 in the definition of thick run. To this aim we split τ into $\tau = \rho\,\rho'$ where $\mathrm{trg}(\rho) = t = \mathrm{src}(\rho')$, and are going to prove the following two claims (a) and (a'). Let $D := P(nM)^n + (nM+1)^3$. For $x, y \in \mathbb{Q}^2$, let $\mathrm{dist}(x, y)$ denote their Euclidean distance.

(a) Some vectors v_1, v_2, v_3, v_4 are $P(nM)^n$-sequentially enabled in ρ, and the sequential cone $\textsc{SeqCone}(v_1, v_2, v_3, v_4)$ contains a point $u \in \mathbb{Q}^2_{\geq 0}$ with $\|u - t\| \leq D$.

(a') Some vectors v'_1, v'_2, v'_3, v'_4 are $P(nM)^n$-sequentially enabled in $\mathrm{rev}(\rho')$, and the sequential cone $\textsc{SeqCone}(v'_1, v'_2, v'_3, v'_4)$ contains a point $u \in \mathbb{Q}^2_{\geq 0}$ with $\|u - t\| \leq D$.

In simple words, instead of proving point 3, we prove that both sequential cones contain a point v which is sufficiently close to t.

▷ Claim 3.3. The conditions (a) and (a') guarantee that τ is thick.

Indeed, points 1–2 in the definition of thick run are immediate as $P(nM) \leq p(nM)$. For point 3, observe that the inequality (1) implies $p(nM)^n \geq \sqrt{2} \cdot D$, which guarantees that the circle $\{u \in \mathbb{Q}^2_{\geq 0} \mid \mathrm{dist}(u, t) \leq \sqrt{2} \cdot D\}$ does not touch any half-line ℓ_w induced by a non-negative vector w with $\|w\| \leq p(nM)^n$. In consequence, neither does the square $X := \{u \in \mathbb{Q}^2_{\geq 0} \mid \|u - t\| \leq D\}$ inscribed in the circle, and hence X lies between two consecutive half-lines ℓ_w induced by a non-negative vector w with $\|w\| \leq p(nM)^n$. Hence, as $\textsc{SeqCone}(v_1, v_2, v_3, v_4)$ contains some point of X, by Lemma 2.2 it includes the whole X, and likewise $\textsc{SeqCone}(v'_1, v'_2, v'_3, v'_4)$. In consequence, the whole X is included in $\textsc{SeqCone}(v_1, v_2, v_3, v_4) \cap \textsc{SeqCone}(v'_1, v'_2, v'_3, v'_4)$ which entails point 3. Claim 3.3 is thus proved.

As condition (a') is fully symmetric to (a), we focus exclusively on proving condition (a), i.e., on constructing sequentially enabled vectors v_1, v_2, v_3, v_4.

Vector t lies outside of $p(nM)^n$-belts, hence outside of all the $P(nM)^n$-belts, therefore its norm $\|t\| > P(nM)^n$. Relying on Lemma 3.2, let c_1 be the first configuration in the run ρ which enables a semi-positive cycle π_1 of length bounded by $P(nM)$, and let $v_1 = \mathrm{eff}(\pi_1)$. We start with the following obvious claim (let v_0 be some vertical vector, e.g. $v_0 = (0, 1)$):

▷ Claim 3.4. $\textsc{SeqCone}(v_0)$ contains a point $u \in \mathbb{Q}^2_{\geq 0}$ such that $\|u - c_1\| \leq P(nM)^n + nM$.

Indeed, due to Lemma 3.2 we may assume $\|c_1\| \leq P(nM)^n + M$ and hence $u = (0, 0)$ does the job.

Recall that the relation $\circlearrowleft$ defines a total order on pairwise non-colinear non-negative vectors.

▷ Claim 3.5. We can assume w.l.o.g. that $v_1 \circlearrowleft t$.

Indeed, if v_1 and t were colinear then $t \in \textsc{Cone}(v_1)$ and hence condition (a) would hold.

Split ρ into the prefix ending in c_1 and the remaining suffix: $\rho = \rho_1\,\sigma$, where $\mathrm{trg}(\rho_1) = c_1 = \mathrm{src}(\sigma)$. As the next step we will identify a configuration c_2 in σ which satisfies Claim 3.6 (which will serve later as the basis of induction) and enables a cycle π_2 with effect v_2 (as stated in Claim 3.7).

▷ Claim 3.6. $\textsc{SeqCone}(v_0, v_1)$ contains a point $u \in \mathbb{Q}^2_{\geq 0}$ such that $\|u - c_2\| \leq P(nM)^n + 2nM$.

The proof of Claim 3.6 depends on whether v_1 is positive. If v_1 is so, we simply duplicate the first cycle: $c_2 := c_1$ and $\pi_2 := \pi_1$, and use Claim 3.4. Otherwise v_1 is vertical due to Claim 3.5. If $t[1] \leq W = P(nM)^n + (n+1)M$ then condition (a) holds immediately as $\textsc{SeqCone}(v_1) = \ell_{v_1}$ contains a point $u \in \mathbb{Q}^2_{\geq 0}$ with $\|u - t\| \leq P(nM)^n + (n+1)M \leq D$. Therefore suppose $t[1] > P(nM)^n + (n+1)M$, and define the sequence $d_1, \ldots, d_m$ of configurations as follows (cf. Figure 5): let $d_1 := c_1$, and let d_{i+1} be the first configuration in

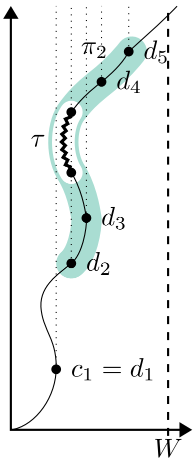


Figure 5 Proof of Claim 3.6.

σ with $d_{i+1}[1] > d_i[1]$. Recall that $d_1[1] \leq P(nM)^n + M$, and observe that $d_{i+1}[1] \leq d_i[1] + M$. Thus by the pigeonhole principle $m > n$ and hence for some $i < j \leq n+1$ the configurations d_i and d_j must have the same control state. The infix σ_{ij} of the path σ from d_i to d_j is thus a cycle, enabled in d_i, whose effect is positive on the first (horizontal) coordinate. Let $c_2 := d_i$. As $c_2[1] \leq P(nM)^n + (n+1)M$, $\textsc{SeqCone}(v_0, v_1) = \ell_{v_0}$ contains necessarily a point $u \in \mathbb{Q}^2_{\geq 0}$ such that $\|u - c_2\| \leq P(nM)^n + (n+1)M$, which proves Claim 3.6.

▷ Claim 3.7. The configuration c_2 $\{1\}$-enables a cycle π_2 of length bounded by $p(nM)^n$, such that the first coordinate of $\mathrm{eff}(\pi_2)$ is positive.

Recalling the proof of the previous claim, observe that the first (horizontal) coordinate in the infix σ_{ij} is bounded by $P(nM)^n + (n+1)M$, and think of the second (vertical) coordinate as irrelevant. Let π_2 be the path inducing σ_{ij}. For bounding the length of π_2, as long as π_2 contains a cycle α with vertical effect $(0, w)$, remove α from π_2. This process ends yielding a cycle π_2 of length at most $(P(nM)^n + (n+1)M) \cdot n$, and hence at most $p(nM)^n$ (by the inequality (1)), which is $\{1\}$-enabled in c_2, but not necessarily enabled. Let $v_2 := \mathrm{eff}(\pi_2)$.

▷ Claim 3.8. We can assume w.l.o.g. that $v_2 \circlearrowleft t$.

Indeed, if $v_1 = v_2$ then Claim 3.5 does the job; otherwise v_1 is vertical and then $t \circlearrowleft v_2$ (or t colinear with v_2) would imply $t \in \textsc{SeqCone}(v_1, v_2)$, hence condition (a) would hold again.

Split σ further into the prefix ending in c_2 and the remaining suffix: $\sigma = \rho_2\sigma'$, where $\mathrm{trg}(\rho_2) = c_2 = \mathrm{src}(\sigma')$. If σ' contains a configuration which $\emptyset$-enables a simple cycle whose effect w belongs to $\measuredangle[t, -v_2)$ then $t \in \textsc{SeqCone}(v_2, w)$ and hence condition (a) holds. We aim at achieving this objective incrementally (cf. Figure 6).

For $i \geq 2$, let c_{i+1} be the first configuration in σ' after c_i that $\emptyset$-enables a simple cycle π_{i+1} with effect $v_{i+1} \in \measuredangle(v_i, -v_i)$. As discussed above, if $v_{i+1} \in \measuredangle[t, -v_i)$ for some i then $t \in \textsc{SeqCone}(v_i, v_{i+1})$ and hence condition (a) holds. Assume therefore that the sequence $v_1, \ldots, v_m$ so defined satisfies $v_{i+1} \in \measuredangle(v_i, t)$ for all $i \geq 2$. Let $c_{m+1} := t$. As vectors $v_3, \ldots, v_m$ are pairwise different, semi-positive and, being effects of simple cycles, have norms at most nM, we know that $m \leq (nM+1)^2 + 1$.

▷ Claim 3.9. For every $i = 1, \ldots, m$, $\textsc{SeqCone}(v_0, v_i)$ contains a point $u \in \mathbb{Q}^2_{\geq 0}$ such that $\|u - c_{i+1}\| \leq P(nM)^n + (i+1)nM$.

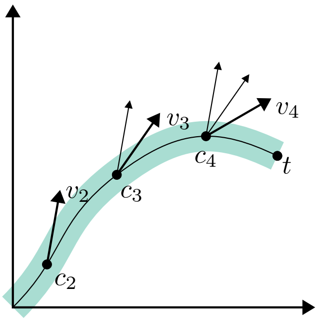


Figure 6 Incremental construction of $v_1, \ldots, v_m$.

Proof. By induction on i. The induction base is exactly Claim 3.6. For the induction step, we are going to show that $\textsc{SeqCone}(v_0, v_i)$ contains a vector u such that $\|u - c_{i+1}\| \leq P(nM)^n + (i+1)nM$. Decompose the infix of σ' which starts in c_i and ends in c_{i+1} into simple cycles, plus the remaining path $\bar{\rho}$ of length at most n. The norm of the effect $\bar{v}$ of $\bar{\rho}$ is hence bounded by nM, and we have

$$c_{i+1} \;=\; c_i + s + \bar{v},$$

where s is the sum of effects of all the simple cycles. By the definition of v_{i+1}, the effects of all the simple cycles belong to the half-plane $\measuredangle[-v_i, v_i]$, and hence there belongs s. By induction assumption there is $u' \in \textsc{SeqCone}(v_0, v_{i-1})$ such that $\|u' - c_i\| \leq P(nM)^n + inM$. As $v_{i-1} \circlearrowright v_i$, we also have $u' \in \textsc{SeqCone}(v_0, v_i)$. Consider the point

$$u \;:=\; u' + s$$

which necessarily belongs to the half-plane $\measuredangle[-v_i, v_i]$ but not necessarily to $\textsc{SeqCone}(v_0, v_i) = \measuredangle[-v_i, v_i] \cap \mathbb{Q}^2_{\geq 0}$. Ignoring this issue, by routine calculations we get

$$\|u - c_{i+1}\| \;=\; \|u' + s - c_i - s - \bar{v}\| \;\leq\; \|u' - c_i\| + \|\bar{v}\| \leq \|u' - c_i\| + nM \leq P(nM)^n + (i+1)nM$$

as required for the induction step. Finally, if $u \notin \mathbb{Q}^2_{\geq 0}$, translate u towards c_{i+1} until it enters the non-negative orthant $\mathbb{Q}^2_{\geq 0}$; clearly, the translation can only decrease the value of $\|u - c_{i+1}\|$. ◀

Applying the claim to $i = m$, and knowing that $m \leq (nM+1)^2 + 1$, we get some point $u \in \textsc{SeqCone}(v_0, v_m)$ such that $\|u - t\| \leq P(nM)^n + ((nM+1)^2+1) \cdot nM \leq P(nM)^n + (nM+1)^3$. Furthermore, relying on the assumptions that t lies outside of all $p(nM)^n$-belts and that $v_1 \circlearrowright t$ we prove, similarly as in the proof of Claim 3.3, that $v_1 \circlearrowright u$ and hence the point u belongs also to $\textsc{SeqCone}(v_1, v_m)$. This completes the proof of Theorem 3.1. ◀

4 Dichotomy in Action

This section illustrates applicability of Theorem 3.1. As before, we use symbols n and M for the number of control states, and the norm of a 2-VASS, respectively. As the first corollary we provide a pumping lemma for 2-VASS: in case of thin runs apply, essentially, pumping schemes of 1-VASS, and in case of thick runs use the cycles enabled along a run. As another application, we derive an alternative proof of the exponential run property for 2-VASS.

▶ **Theorem 4.1** (Pumping). *There is a polynomial p such that every $(0,0)$-run τ in a 2-VASS of length greater that $p(nM)^n$ factors into $\tau = \tau_0 \tau_1 \ldots \tau_k$ $(k \geq 1)$, so that for some non-empty cycles $\alpha_1, \ldots, \alpha_k$ of length at most $p(nM)^n$, the path $\tau_0 \alpha_1^i \tau_1 \alpha_2^i \ldots, \alpha_k^i \tau_k$ is a $(0,0)$-run for every $i \in \mathbb{N}$. Furthermore, the lengths of τ_0 and τ_k are also bounded by $p(nM)^n$.*

▶ **Theorem 4.2** (Exponential run). *There is a polynomial p such that for every $(0,0)$-run τ in a 2-VASS, there is a $(0,0)$-run of length bounded by $p(nM)^n$ with the same source and target as τ.*

We fix from now on a 2-VASS $V = (Q,T)$ and the polynomial p of Theorem 3.1. Let $A = p(nM)^n$. Both proofs proceed separately for thin and thick runs τ. The former (fairly standard) case is treated in the full version of the paper, so assume below τ to be A-thick. The polynomials required in Theorems 4.1 and 4.2 can be read out from the constructions.

We rely on the standard tool, cf. Prop. 2 in [4] (the norm of a system of inequalities is the largest absolute value of its coefficient, and likewise we define the norm of a solution):

▶ **Lemma 4.3.** *Let $\mathcal{U}$ be a system of d linear inequalities of norm M with k variables. Then the smallest norm of a non-negative-integer solution of $\mathcal{U}$ is in $\mathcal{O}(k \cdot M)^d$.*

Consider a split $\tau = \rho\rho'$, where $\rho = \rho_1 \rho_2 \rho_3 \rho_4 \rho_5$ and $\rho' = \rho_5' \rho_4' \rho_3' \rho_2' \rho_1'$, as well as cycles $\pi_1, \ldots, \pi_4$ and $\pi_1', \ldots, \pi_4'$ given by the definition of thick run. Let $v_1, \ldots, v_4$ and $v_1', \ldots, v_4'$ be the respective effects of $\pi_1, \ldots, \pi_4$ and $\pi_1', \ldots, \pi_4'$. For $j = 1, \ldots, 4$ let $c_j = \mathrm{trg}(\rho_j)$ and for $j = 2, \ldots, 4$ let $e_j \in \mathbb{N}^2$ be the minimal non-negative vector such that the configuration $c_j + e_j$ enables cycle π_j. We define the following system $\mathcal{U}$ of linear inequalities with 6 variables a_1, a_2, a_3, a_4, x, y (max is understood point-wise):

$$a_1 v_1 \geq e_2 \tag{2}$$

$$a_1 v_1 + a_2 v_2 \geq \max(e_2, e_3) \tag{3}$$

$$a_1 v_1 + a_2 v_2 + a_3 v_3 \geq \max(e_3, e_4) \tag{4}$$

$$a_1 v_1 + a_2 v_2 + a_3 v_3 + a_4 v_4 = (x,y) \geq e_4 \tag{5}$$

(Observe that when $v_1[j] = 0$, i.e., in case when v_1 is vertical or horizontal, $e_j = 0$ and therefore one of the two first inequalities is always satisfied, namely $a_1 v_1[j] \geq e_2[j]$.) Likewise, we have a system of inequalities $\mathcal{U}'$ with 6 variables $a_1', a_2', a_3', a_4', x', y'$. Observe that the sequential cone $\textsc{SeqCone}(v_1, v_2, v_3, v_4)$ contains exactly (projections on (x,y) of) non-negative rational solutions of the modified system $\mathcal{U}^{(0,0)}$ obtained by replacing all the right-hand sides with $(0,0)$. Likewise we define $\mathcal{U}'^{(0,0)}$. Finally, we define the compound system $\mathcal{C}$ by enhancing the union of $\mathcal{U}$ and $\mathcal{U}'$ with two additional equalities (likewise we define the system $\mathcal{C}^{(0,0)}$)

$$(x,y) = (x',y'). \tag{6}$$

▷ Claim 4.4. $\mathcal{C}$ admits a non-negative integer solution $(a_1, a_2, a_3, a_4, x, y, a_1', a_2', a_3', a_4', x', y')$.

Proof. The system $\mathcal{C}^{(0,0)}$ admits a non-negative rational solution as the intersection of the cones $\textsc{SeqCone}(v_1, v_2, v_3, v_4)$ and $\textsc{SeqCone}(v_1', v_2', v_3', v_4')$ is non-empty by assumption. As intersection of cones is stable under multiplications by non-negative rationals, the solution can be scaled up arbitrarily, to yield a non-negative integer one, and even a non-negative integer solution of the stronger system $\mathcal{C}$. ◀

▷ Claim 4.5. For every non-negative integer solution of $\mathcal{C}$, for the cycles defined as $\alpha_j := \pi_j^{a_j}$ and $\alpha_j' := (\pi_j')^{a_j'}$, for $j = 1, 2, 3, 4$, the following path is a $(0,0)$-run:

$$\rho_1\ \alpha_1\ \rho_2\ \alpha_2\ \rho_3\ \alpha_3\ \rho_4\ \alpha_4\ \rho_5\ \rho_5'\ \alpha_4'\ \rho_4'\ \alpha_3'\ \rho_3'\ \alpha_2'\ \rho_2'\ \alpha_1'\ \rho_1'.$$

Proof. The first two inequalities (2) enforce that the first cycle π_1 is repeated sufficiently many a_1 times so that π_2 is enabled in configuration $\mathrm{trg}(\rho_1\,\alpha_1\,\rho_2)$. Then the next two inequalities (3) enforce that π_1 and π_2 are jointly repeated sufficiently many a_1, a_2 times so that π_2 is still enabled after its last repetition (which guarantees that every of intermediate repetitions of π_2 is also enabled), and that π_3 is enabled in configuration $\mathrm{trg}(\rho_1\,\alpha_1\,\rho_2\,\alpha_2\,\rho_3)$. Likewise for (4). Finally, the inequalities (5) enforce that $\pi_1, \ldots, \pi_4$ are jointly repeated sufficiently many times so that π_4 is still enabled after its last repetition. Analogous argument, but in the reverse order, applies for the repetitions of $\pi'_4, \ldots, \pi'_1$. Finally, equalities (6) ensure that the total effect of $\alpha_1, \ldots, \alpha_4$ is precisely compensated by the total effect of $\mathrm{rev}(\alpha'_1), \ldots, \mathrm{rev}(\alpha'_4)$. ◀

Proof of Theorem 4.1. Consider a solution of $\mathcal{C}$. In particular the sum $\mathrm{eff}(\alpha_1)+\ldots+\mathrm{eff}(\alpha_j)$, as well as $\mathrm{eff}(\mathrm{rev}(\alpha'_1)) + \ldots + \mathrm{eff}(\mathrm{rev}(\alpha'_j))$, is necessarily non-negative for every $j = 1, \ldots, 4$. Therefore, as a direct corollary of Claim 4.5, for every $i \in \mathbb{N}$ the path

$$\rho_1\,\alpha_1^i\,\rho_2\,\alpha_2^i\,\rho_3\,\alpha_3^i\,\rho_4\,\alpha_4^i\,\rho_5\,\rho'_5\,(\alpha'_4)^i\,\rho'_4\,(\alpha'_3)^i\,\rho'_3\,(\alpha'_2)^i\,\rho'_2\,(\alpha'_1)^i\,\rho'_1$$

is also a $(0,0)$-run. For bounding the lengths of cycles we use Claim 4.4 and apply Lemma 4.3 to $\mathcal{C}$, to deduce that $\mathcal{C}$ admits a non-negative integer solution of norm polynomial in $A = p(nM)^n$. This, together with the bounds on lengths of cycles $\pi_1, \ldots, \pi_4$ and $\pi'_1, \ldots, \pi'_4$ in the definition of A-thick run, entails required bounds on the lengths of the pumpable cycles. Finally, the lengths of the extremal factors ρ_1 and ρ'_1 can be also bounded: if ρ_1 (resp. ρ'_1) is long enough it must admit a repetition of configuration, we add one more cycle determined by the first (resp. last) such repetition, thus increasing k from 8 to 10. ◀

For proving Theorem 4.2 we will need a slightly more elaborate pumping. By the definition of thick run, both coordinates are bounded by A along ρ_1 and ρ'_1. W.l.o.g. assume that no configuration repeats in each of the two runs, and hence their lengths are bounded by A^2.

Let $\mathcal{C}_\delta$ denote the union of of $\mathcal{U}$ and $\mathcal{U}'$ enhanced, this time, by the two equalities

$$(x, y) + (\delta_x, \delta_y) = (x', y').$$

The two additional variables δ_x, δ_y describe, intuitively, possible differences between the total effect of $\pi_1^{a_1}, \ldots, \pi_4^{a_4}$ and the total effect of $\mathrm{rev}(\pi'_1)^{a'_1}, \ldots, \mathrm{rev}(\pi'_4)^{a'_4}$. The projection of any solution of $\mathcal{C}_\delta$ on variables (δ_x, δ_y) we call below a *shift*.

▷ Claim 4.6. For some non-negative integer m bounded polynomially with respect to A, all the four vectors $(0, m)$, $(m, 0)$, $(0, -m)$ and $(-m, 0)$ are shifts.

Proof. We reason analogously as in the proof of Claim 4.4, but this time we rely on the assumption that intersection of the cones $\textsc{SeqCone}(v_1, v_2, v_3, v_4)$ and $\textsc{SeqCone}(v'_1, v'_2, v'_3, v'_4)$ is non-trivial, and hence contains, for some $v \in \mathbb{Q}^2_{>0}$ and $a \in \mathbb{Q}_{>0}$, the points v and $v + (0, a)$. By scaling we obtain an integer point $v' \in \mathbb{N}^2$ and a non-negative integer $m_1 \in \mathbb{N}$ so that v' and $v' + (0, m_1)$ both belong to the intersection of cones. Therefore the vector $(0, m_1)$ is a shift. Likewise we obtain three other non-negative integers $m_2, m_3, m_4 \in \mathbb{N}$ such that $(m_2, 0)$, $(0, -m_3)$ and $(-m_4, 0)$ are all shifts. Each of the integers $m_1, \ldots, m_4$ can be bounded polynomially in A using Lemma 4.3. As shifts are stable under multiplication by non-negative integers, it is enough to take as m the least common multiple of the four integers. ◀

Proof of Theorem 4.2. We use m from the last claim to modify all factors of τ except for ρ_1 and ρ'_1, in order to reduce their lengths to at most $n \cdot m^2$. W.l.o.g. assume m to be larger than A (take a sufficient multiplicity of m otherwise); this assumption allows us to

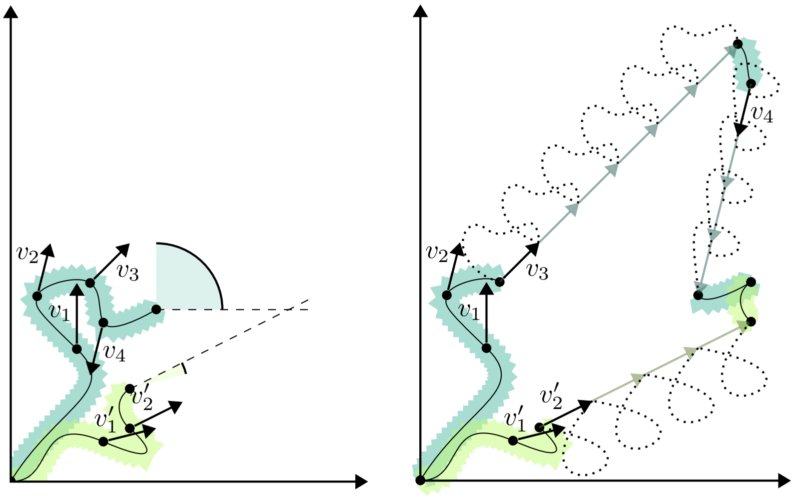


Figure 7 Contracted paths $\widetilde{\rho}, \widetilde{\rho}'$ (left) and reconstructed $(0,0)$-run $\bar{\tau} = \bar{\rho}\,\bar{\rho}'$ (right).

proceed uniformly, irrespectively whether v_1 is positive or not. Observe that any path longer than $n \cdot m^2$ must contain two configurations with the same control state whose vectors are coordinate-wise congruent modulo m. As long as this happens, we remove the infix; note that this operation changes the effect of the whole path by a multiplicity of m on every coordinate. If this operation is performed on factors ρ_2, ρ_3, ρ_4, ρ_5, ρ_5', ρ_4', ρ_3', ρ_2', the paths ρ, ρ' are transformed into contracted paths (see the left picture in Figure 7) of the form:

$$\widetilde{\rho} \;=\; \rho_1\, \widetilde{\rho}_2\, \widetilde{\rho}_3\, \widetilde{\rho}_4\, \widetilde{\rho}_5, \qquad \widetilde{\rho}' \;=\; \widetilde{\rho}_5'\, \widetilde{\rho}_4'\, \widetilde{\rho}_3'\, \widetilde{\rho}_2'\, \sigma_1,$$

each of total length at most $5\,n \cdot m^2$. Importantly, their effects $\mathrm{eff}(\widetilde{\rho})$ and $\mathrm{eff}(\widetilde{\rho}')$ are bounded polynomially in A, and their difference is (coordinate-wise) divisible by m:

$$\mathrm{eff}(\widetilde{\rho}) - \mathrm{eff}(\mathrm{rev}(\widetilde{\rho}')) \;=\; (am, bm) \qquad \text{for some integers } a, b \in \mathbb{Z} \text{ polynomial in } A.$$

Our aim is now to pump up the cycles $\pi_1, \ldots, \pi_4$ and $\mathrm{rev}(\pi_1'), \ldots, \mathrm{rev}(\pi_4')$ (see the right picture in Figure 7), to finally end up with the paths of the form

$$\bar{\rho} \;=\; \rho_1\, \pi_1^{a_1}\, \widetilde{\rho}_2\, \pi_2^{a_2}\, \widetilde{\rho}_3\, \pi_3^{a_3}\, \widetilde{\rho}_4\, \pi_4^{a_4}\, \widetilde{\rho}_5, \quad \bar{\rho}' \;=\; \widetilde{\rho}_5'\, (\pi_4')^{a_4'}\, \widetilde{\rho}_4'\, (\pi_3')^{a_3'}\, \widetilde{\rho}_3'\, (\pi_2')^{a_2'}\, \widetilde{\rho}_2'\, (\pi_1')^{a_1'}\, \rho_1', \tag{7}$$

such that $\bar{\tau} = \bar{\rho}\,\bar{\rho}'$ is a $(0,0)$-run. In other words, we aim at $\mathrm{eff}(\bar{\rho}) = \mathrm{eff}(\mathrm{rev}(\bar{\rho}'))$. We are going to use Lemma 4.3 twice. For $j = 2, \ldots, 5$ let $c_j := \mathrm{eff}(\rho_1 \widetilde{\rho}_2 \ldots \widetilde{\rho}_j) \in \mathbb{Z}^2$, and let f_j be the minimal non-negative vector such that the configuration $c_{j-1} + f_j$ enables $\widetilde{\rho}_j$. For $j = 2, \ldots, 4$ let $e_j \in \mathbb{N}^2$ be the minimal non-negative vector such that the configuration $c_j + e_j$ enables π_j. Finally, let e_5 be the minimal non-negative vector such that $c_5 + e_5 \geq (0,0)$. Analogously to the system $\mathcal{U}$ (2)–(5), we define the system $\widetilde{\mathcal{U}}$ of linear inequalities:

$$\begin{aligned} a_1 m v_1 &\;\geq\; \max(e_2, f_2) \\ a_1 m v_1 + a_2 m v_2 &\;\geq\; \max(e_2, e_3, f_3) \\ a_1 m v_1 + a_2 m v_2 + a_3 m v_3 &\;\geq\; \max(e_3, e_4, f_4) \\ a_1 m v_1 + a_2 m v_2 + a_3 m v_3 + a_4 m v_4 &\;\geq\; \max(e_4, e_5, f_5) \end{aligned}$$

In words, $\widetilde{\mathcal{U}}$ requires that every prefix of $\bar{\rho}$ is enabled in the source $(0,0)$-configuration, and that the number of repetitions of every cycle π_i is divisible by m. Clearly $\widetilde{\mathcal{U}}$ has a non-negative integer solution, as v_1 is either positive, or vertical or horizontal in which case

v_2 is positive on the relevant coordinate. Likewise we define a system of inequalities $\widetilde{\mathcal{U}}'$ that requires that every prefix of $\mathrm{rev}(\bar{\rho}')$ is enabled in the target $(0,0)$-configuration. Consider some fixed solutions of $\widetilde{\mathcal{U}}$ and $\widetilde{\mathcal{U}}'$ bounded, by the virtue of Lemma 4.3, polynomially in A. We have thus two fixed runs $\bar{\rho}$ and $\mathrm{rev}(\bar{\rho}')$ of the form (7), with source vector $(0,0)$; the number of repetitions of each cycles is divisible by m, and the difference of their effects is (coordinate-wise) divisible by m:

$$\mathrm{eff}(\bar{\rho}) - \mathrm{eff}(\mathrm{rev}(\bar{\rho}')) \;=\; (am, bm) \qquad \text{for some integers } a, b \in \mathbb{Z} \text{ polynomial in } A.$$

As shifts are closed under addition, by Claim 4.6 we know that (am, bm) is a shift. Substituting (am, bm) for (δ_x, δ_y) in the system $\mathcal{C}_\delta$ yields a system which admits, again by Lemma 4.3, a solution bounded polynomially in A. We use such a solution to increase the numbers of repetitions of respective cycles $a_1, \ldots, a_4$ and $a'_4, \ldots, a'_1$ in $\bar{\rho}$ and $\bar{\rho}'$, respectively. This turns the path $\bar{\tau} = \bar{\rho}\,\bar{\rho}'$ into a $(0,0)$-run of length bounded polynomially in A. ◀

References

1 Mohamed Faouzi Atig, Dmitry Chistikov, Piotr Hofman, K. Narayan Kumar, Prakash Saivasan, and Georg Zetzsche. The complexity of regular abstractions of one-counter languages. In *Proc. LICS'16*, pages 207–216, 2016.

2 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in Two-Dimensional Vector Addition Systems with States Is PSPACE-Complete. In *Proc. LICS'15*, pages 32–43, 2015.

3 Dmitry Chistikov, Wojciech Czerwinski, Piotr Hofman, Michal Pilipczuk, and Michael Wehar. Shortest Paths in One-Counter Systems. In *Proc. of FOSSACS'16*, pages 462–478, 2016.

4 Dmitry Chistikov and Christoph Haase. The Taming of the Semi-Linear Set. In *Proc. ICALP'16*, pages 128:1–128:13, 2016.

5 Wojciech Czerwinski and Slawomir Lasota. Regular separability of one counter automata. In *LICS'17*, pages 1–12, 2017.

6 Matthias Englert, Ranko Lazic, and Patrick Totzke. Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-Complete. In *Proc. of LICS '16*, pages 477–484, 2016.

7 John E. Hopcroft and Jean-Jacques Pansiot. On the Reachability Problem for 5-Dimensional Vector Addition Systems. *Theor. Comput. Sci.*, 8:135–159, 1979. doi:10.1016/0304-3975(79)90041-0.

8 Richard M. Karp and Raymond E. Miller. Parallel Program Schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.

9 S. Rao Kosaraju. Decidability of Reachability in Vector Addition Systems (Preliminary Version). In *STOC'82*, pages 267–281, 1982.

10 Michel Latteux. Langages à un Compteur. *J. Comput. Syst. Sci.*, 26(1):14–33, 1983.

11 Ernst W. Mayr. An Algorithm for the General Petri Net Reachability Problem. In *STOC'81*, pages 238–246, 1981.

Computational Complexity of Synchronization under Regular Constraints

Henning Fernau
Fachbereich 4 - Abteilung Informatikwissenschaften, Universität Trier, Germany
fernau@uni-trier.de

Vladimir V. Gusev
Leverhulme Research Centre for Functional Materials Design, University of Liverpool, UK
https://www.liverpool.ac.uk/chemistry/staff/vladimir-gusev/
vladimir.gusev@liverpool.ac.uk

Stefan Hoffmann
Fachbereich 4 - Abteilung Informatikwissenschaften, Universität Trier, Germany
hoffmanns@uni-trier.de

Markus Holzer
Institut für Informatik, Universität Gießen, Germany
holzer@informatik.uni-giessen.de

Mikhail V. Volkov
Institute of Natural Sciences and Mathematics, Ural Federal University, Yekaterinburg, Russia
http://csseminar.imkn.urfu.ru/volkov/
m.v.volkov@urfu.ru

Petra Wolf
Fachbereich 4 - Abteilung Informatikwissenschaften, Universität Trier, Germany
wolfp@uni-trier.de

Abstract

Many variations of synchronization of finite automata have been studied in the previous decades. Here, we suggest studying the question if synchronizing words exist that belong to some fixed constraint language, given by some partial finite automaton called constraint automaton. We show that this synchronization problem becomes PSPACE-complete even for some constraint automata with two states and a ternary alphabet. In addition, we characterize constraint automata with arbitrarily many states for which the constrained synchronization problem is polynomial-time solvable. We classify the complexity of the constrained synchronization problem for constraint automata with two states and two or three letters completely and lift those results to larger classes of finite automata.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Problems, reductions and completeness

Keywords and phrases Finite automata, synchronization, computational complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.63

Funding The financial support of the last two authors by the DFG-funded project FE560/9-1 is gratefully acknowledged. V. V. Gusev is supported by the Leverhulme Trust. M. V. Volkov is supported by the Ministry of Science and Higher Education of the Russian Federation, project no. 1.580.2016, and the Competitiveness Enhancement Program of Ural Federal University.

Acknowledgements This project started during the workshop “Modern Complexity Aspects of Formal Languages” that took place at Trier University on February 11–15, 2019.

1 Introduction

Synchronization is an important concept for many applied areas: parallel and distributed programming, system and protocol testing, information coding, robotics, etc. At least some aspects of synchronization are captured by the notion of a synchronizing automaton; for

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 63; pp. 63:1–63:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

instance, synchronizing automata adequately model situations in which one has to direct a certain system to a particular state without a priori knowledge of its current state. We only refer to some survey papers [24, 28], as well as to Chapter 13 in [17], that also report on some of these applications. An automaton is called synchronizing if there exists a word that brings it to a known state independently of the starting state. This concept is quite natural and has been investigated intensively in the last six decades. It is related to the arguably most famous open combinatorial question in automata theory, formulated by Černý in [8]. The Černý conjecture states that every n-state synchronizing automaton can be synchronized by a word of length smaller or equal $(n-1)^2$. Although this bound was proven for several classes of finite-state automata, the general case is still widely open. The currently best upper bound on this length is cubic, and only very little progress has been made, basically improving on the multiplicative constant factor in front of the cubic term, see [25, 27].

Due to the importance of this notion of synchronizing words, quite a large number of generalizations and modifications have been considered in the literature. We only mention four of these in the following. Instead of synchronizing the whole set of states, one could be interested in synchronizing only a subset of states. This and related questions were first considered by Rystsov in [23]. Instead of considering deterministic finite automata (DFAs), one could alternatively study the notion of synchronizability for nondeterministic finite automata [12, 21]. The notion of synchronizability naturally transfers to partially defined transition functions where a synchronizing automata avoiding undefined transitions is called *carefully synchronizing*, see [9, 20, 21]. To capture more adaptive variants of synchronizing words, synchronizing strategies have been introduced in [19]. Recall that the question of synchronizability (without length bounds) is solvable in polynomial time for complete DFAs [28]. However, in all of the mentioned generalizations, this synchronizability question becomes even PSPACE-complete. This general tendency can also be observed in the generalization that we introduce in this paper, which we call *regular constraints*. These constraints are defined by some (fixed) finite automaton describing a regular language R, and the question is, given some DFA A, if A has some synchronizing word from R. This notion explicitly appeared in [13] as an auxiliary tool: it was shown that the synchronization problem of every automaton $A = (\Sigma, Q, \delta)$ whose letters σ have ranks at most r, i.e., $|\delta(Q, \sigma)| \leq r$, is equivalent to the synchronization of an r-state automaton A' under some regular constraints.

The main research question that we look into is to understand for which regular constraints the question of synchronizability is solvable in polynomial time (as it is for $R = \Sigma^*$), or for which it is hard. Furthermore, it would be interesting to see complexity classes different from P and PSPACE to show up (depending on R). In our paper, we give a complete description of the complexity status for constraints that can be described by partial 2-state deterministic automata on alphabets with at most three letters. In this case, indeed, we only observe P and PSPACE situations. However, we also find 3-state automata (on binary input alphabets) that exhibit an NP-complete synchronization problem when considered as constraints. We describe several ways how to generalize our results to larger constraint automata. Moreover, we identify several classes of constraint automata that imply feasible synchronization problems. We motivate our study of synchronization under regular constraints by the following example.

A motivating result. In the theory of synchronizing automata, one normally allows the directing instruction to be an arbitrary word over the input language of the corresponding automaton. In reality, however, available commands might be subject to certain restrictions; for instance, it is quite natural to assume that a directing instruction should always start and end with a specific command that first switches the automaton to a “directive” mode and then returns the automaton to its usual mode. In its simplest form, the switching between "normal mode" and "directive" (synchronization) mode can be modeled as ab^*a. This scenario produces an NP-complete synchronization problem. In order to state our first result formally, we make use of some (standard) notions defined in Section 2.

▶ **Proposition 1.** *The following problem is NP-complete: Given a deterministic finite complete automaton A with $L(A) \subseteq \{a, b\}^*$, is there a synchronizing word $w \in ab^*a$ for A?*

Notice that this contrasts with the complexity of the synchronizability question for complete DFAs, which can be solved in quadratic time; see, e.g., [24, 28]. Also, it contrasts the complexity of synchronizability for partial DFAs, which is PSPACE-complete; see [21].

The constraint automaton describing ab^*a has three states. As we will see below, only P- and PSPACE-results can be observed for two-state constraint automata over binary and ternary input alphabets. Hence, in a sense, Proposition 1 is a minimal example of a complexity status inbetween P and PSPACE. A proof sketch of Proposition 1 is given below.

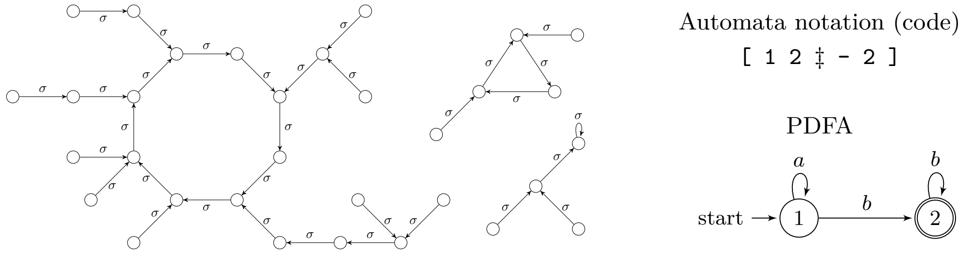
Rystsov [23] considered a problem that he called Global Inclusion Problem for Non-initial Automata. As we will see below, this problem (together with a variation) will be the key problem in our reductions. Looking at the proof of [23, Theorem 2.1], we can observe the following refined result. We consider the next problem that we call $\mathcal{P}_\Sigma$ for brevity. Given a complete DFA A with state set Q and input alphabet Σ, with $a \in \Sigma$, as well as a designated state subset S, is there some word $w \in \{a\}(\Sigma \setminus \{a\})^*$ such that w drives A into S, irrespectively of where A starts processing w? Trivially, $\mathcal{P}_\Sigma$ is in P if $|\Sigma| = 1$.

▶ **Theorem 2.** *$\mathcal{P}_\Sigma$ is NP-hard if $|\Sigma| = 2$, and PSPACE-hard if $|\Sigma| > 2$.*

In particular, the case distinction between binary input alphabets and larger input alphabets (concerning hardness results) comes from the fact that the reduction of Rystsov uses DFA-Intersection Nonemptiness, the non-emptiness of intersection problem for deterministic finite automata on the alphabet $\Sigma \setminus \{a\}$, using Theorem 6.1 in [26] (more details in [11, 18]). In Rystsov's reduction, the state set of the automaton A consists of a part $Q_\cap$, which just copies the n automata A_i (over alphabet $\Sigma \setminus \{a\}$) of a DFA-Intersection Nonemptiness instance, together with n new states $t_i \in Q_\rightarrow$ that move on input a into the initial state s_i of A_i. Likewise, from any state q_i of A_i, letter a leads to s_i. All transitions not yet defined are self-loops. Set S collects all final states of all A_i. Hence, a word $w \in (\Sigma \setminus \{a\})^*$ is accepted by all of the A_i iff aw drives A into S, starting out from any state. The promised proof sketch follows. Modify A to obtain an automaton A' such that A' has a synchronizing word awa, with $w \in (\Sigma \setminus \{a\})^*$ iff aw drives A into S as follows: add a new state s where all letters loop; for all $q \in S$, replace the a-transitions leading from q into s_i by a-transitions leading into s. For more details (membership in NP for $\Sigma = \{a, b\}$ is non-trivial), see Theorem 19.

2 Preliminaries and Definitions

Throughout the paper, we consider deterministic finite automata (DFAs). Recall that a DFA A is a tuple $A = (\Sigma, Q, \delta, q_0, F)$, where the alphabet Σ is a finite set of input symbols, Q is the finite state set, with start state $q_0 \in Q$, and final state set $F \subseteq Q$. The transition function $\delta : Q \times \Sigma \to Q$ extends to words from Σ^* in the usual way. The function δ can be further extended to sets of states in the following way. For every set $S \subseteq Q$ with $S \neq \emptyset$ and $w \in \Sigma^*$, we set $\delta(S, w) := \{\, \delta(q, w) \mid q \in S \,\}$. We sometimes refer to the function δ as a relation and we identify a transition $\delta(q, \sigma) = q'$ with the tuple (q, σ, q'). We call A *complete* if δ is defined for every $(q, a) \in Q \times \Sigma$; if δ is undefined for some (q, a),



```
[ 1 2 ‡ - 2 ]
```

(a) Unary automaton A_σ consisting of 3 sun-structures.

(b) Automata notation and PDFA using its standard interpretation.

Figure 1 Illustration of sun-structures and of the notation of PDFAs.

the automaton A is called *partial*. If $|\Sigma| = 1$, we call A a *unary* automaton. The set $L(A) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \in F \,\}$ denotes the language accepted by A. A semi-automaton is a finite automaton without a specified start state and with no specified set of final states. Notice that $(\Sigma, \binom{Q}{\leq k}, \delta)$ can be viewed as a semi-automaton for each $k \leq |Q|$, when $\binom{Q}{\leq k}$ is the set formed by all subsets of Q of cardinality at most k. The properties of being *deterministic*, *partial*, and *complete* of semi-automata are defined as for DFA. When the context is clear, we call both deterministic finite automata and semi-automata simply *automata*. We call a deterministic complete semi-automaton a DCSA and a partial deterministic finite automaton a PDFA for short. If we want to add an explicit initial state r and an explicit set of final states S to a DCSA A or change them in a DFA A, we use the notation $A_{r,S}$.

An automaton A is called *synchronizing* if there exists a word $w \in \Sigma^*$ with $|\delta(Q, w)| = 1$. In this case, we call w a *synchronizing word* for A. For a word w, we call a state in $\delta(Q, w)$ an *active* state. We call a state $q \in Q$ with $\delta(Q, w) = \{q\}$ for some $w \in \Sigma^*$ a *synchronizing state*. A state from which some final state is reachable is called *co-accessible*. For a set $S \subseteq Q$, we say S is *reachable* from Q or Q is synchronizable to S if there exists a word $w \in \Sigma^*$ such that $\delta(Q, w) = S$. An automaton A is called *returning*, if for every state $q \in Q$, there exists a word $w \in \Sigma^*$ such that $\delta(q, w) = q_0$, where q_0 is the start state of A.

▶ **Fact 1.** *[28] For any DCSA, we can decide if it is synchronizing in polynomial time $O(|\Sigma||Q|^2)$. Additionally, if we want to compute a synchronizing word w, then we need time $O(|Q|^3 + |Q|^2|\Sigma|)$ and the length of w will be $O(|Q|^3)$.*

The following obvious remark will be used frequently without further mentioning.

▶ **Lemma 3.** *Let $A = (\Sigma, Q, \delta)$ be a DCSA and $w \in \Sigma^*$ be a synchronizing word for A. Then for every $u, v \in \Sigma^*$, the word uwv is also synchronizing for A.*

For an automaton A over the alphabet Σ, we denote with $A_{\Sigma'}$ for every $\Sigma' \subset \Sigma$ the restriction of A to the alphabet Σ'. Automaton $A_{\Sigma'}$ is obtained from A by deleting all transitions with labels in $\Sigma \setminus \Sigma'$. We will identify $A_{\{\sigma\}}$ with A_σ for every $\sigma \in \Sigma$. For a complete deterministic automaton A_σ, each connected component of A_σ consists of exactly one cycle and some tails leading into the cycle (see Figure 1a). A cycle is a sequence of states $q_1, q_2, \ldots, q_k$, for $k \in \mathbb{N}$ such that $\delta(q_i, \sigma) = q_{i+1}$ and $\delta(q_k, \sigma) = q_1$. In particular, a cycle may consist of one single state only. The tails are only leading into the cycle since A is deterministic. We call components of this form *sun-structures* as illustrated in Figure 1a.

We call two automata A and A' isomorphic if one automaton can be obtained from the other one by renaming states and alphabet-symbols. Notice that the number of non-isomorphic automata can be quite huge even for small number of states and alphabet sizes;

see [3, 10, 14]. In order to address the presented automata in a compact way, we introduce a short notation motivated by [1, 2, 22], where we assume some order is given on the alphabet and on the state set. Each automaton is denoted by a tuple of size $|Q| \cdot |\Sigma|$ where for each state the mapping of this state with each alphabet symbol is listed. The states themselves are separated by ‡-signs. For example, the first entry of the tuple denotes the transition of the first state under the first symbol (and "-" for an undefined transition), while the second entry denotes the transition of the first state by the second symbol, and so on. We will always assume the first state in the ordering of the states to be the start state of the automaton. See Figure 1b for an example. Final states are not part of this coding.
For a fixed PDFA $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$, we define the *constrained synchronization problem*:

▶ **Definition 4.** $L(\mathcal{B})$-CONSTR-SYNC
Input*: DCSA* $A = (\Sigma, Q, \delta)$*.*
Question*: Is there a synchronizing word* w *for* A *with* $w \in L(\mathcal{B})$*?*

The automaton $\mathcal{B}$ will be called the *constraint automaton*. If an automaton A is a yes-instance of $L(\mathcal{B})$-CONSTR-SYNC we call A *synchronizing with respect to* $\mathcal{B}$. Occasionally, we do not specify $\mathcal{B}$ and rather talk about L-CONSTR-SYNC. We are going to inspect the complexity of this problem for different (small) constraint automata. We assume the reader to have some basic knowledge in computational complexity theory and formal language theory, as contained, e.g., in [15]. For instance, we make use of regular expressions to describe languages. We also identify singleton sets with its elements. We also make use of complexity classes like P, NP, or PSPACE. At one point, we also mention the parameterized complexity class XP. With $\leq_m^{\log}$ we denote a logspace many-one reduction. If for two problems L_1, L_2 it holds that $L_1 \leq_m^{\log} L_2$ and $L_2 \leq_m^{\log} L_1$, then we write $L_1 \equiv_m^{\log} L_2$.

For establishing some of our results, we need the following computational problems taken from [6], which are PSPACE-complete problems for at least binary alphabets, also see [23, 24].

▶ **Definition 5.** SYNC-FROM-SUBSET
Input*: DCSA* $A = (\Sigma, Q, \delta)$ *and* $S \subseteq Q$*.*
Question*: Is there a word* w *with* $|\delta(S, w)| = 1$*?*

▶ **Definition 6.** SYNC-INTO-SUBSET
Input*: DCSA* $A = (\Sigma, Q, \delta)$ *and* $S \subseteq Q$*.*
Question*: Is there a word* w *with* $\delta(Q, w) \subseteq S$*?*

▶ Remark 7. The terminology is not homogeneous in the literature. For instance, SYNC-INTO-SUBSET has different names in [6] and in [23].

3 Placing Constrained Problems Within Complexity Classes

In this section we present several criteria for L which lead to the membership of L-CONSTR-SYNC in different complexity classes, starting by studying unary languages.

▶ **Lemma 8.** *Let* $A = (\{\sigma\}, Q, \delta)$ *be a unary synchronizing DCSA. For all* $i \geq |Q| - 1$*, we have* $\delta(Q, \sigma^i) = \delta(Q, \sigma^{i+1})$*. A shortest word* w *synchronizing* $S \subseteq Q$ *obeys* $|w| \leq |Q| - 1$*.*

▶ **Corollary 9.** *If* $L(\mathcal{B}) \subseteq \{\sigma\}^*$ *for a PDFA* $\mathcal{B}$*, then* $L(\mathcal{B})$-CONSTR-SYNC $\in$ *P.*

▶ **Theorem 10.** *If* L *is regular, then* L-CONSTR-SYNC *is contained in PSPACE.*

We continue with 1-state constraint automata and unions of constraint languages.

▶ **Lemma 11.** *Let $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ be a PDFA. If $L(\mathcal{B}) = L(\mathcal{B}_{\Sigma\setminus\{\sigma\}})$ for some $\sigma \in \Sigma$, then $L(\mathcal{B})$-*Constr-Sync *$\equiv_m^{\log} L(\mathcal{B}_{\Sigma\setminus\{\sigma\}})$-*Constr-Sync*.*

▶ **Corollary 12.** *$L(\mathcal{B})$-*Constr-Sync *$\in$ P for every one-state constraint automaton $\mathcal{B}$.*

▶ **Lemma 13.** *If L is a finite union of languages $L_1, L_2, \ldots, L_n$ such that for each $1 \leq i \leq n$ the problem L_i-*Constr-Sync *$\in$ P, then L-*Constr-Sync *$\in$ P.*

▶ **Theorem 14.** *Let $L \subseteq \Sigma^*$. If $\{ v \in \Sigma^* \mid \exists u, w \in \Sigma^* : uvw \in L \} = \Sigma^*$, L-*Constr-Sync *$\in$ P.*

Proof. Let $A = (\Sigma, Q, \delta)$ be a DCSA. To decide if A has a synchronizing word from L, simply test if A is synchronizing at all, cf. Fact 1. Assume $v \in \Sigma^*$ is a synchronizing word for A. By assumption, $uvw \in L$ for some $u, w \in \Sigma^*$. Moreover, uvw also synchronizes A. ◀

With the same type of reasoning, one can show:

▶ **Theorem 15.** *Let $L \subseteq L' \subseteq \Sigma^*$. If $L' \subseteq \{ v \in \Sigma^* \mid \exists u, w \in \Sigma^* : uvw \in L \}$, then L-*Constr-Sync *$\equiv_m^{\log} L'$-*Constr-Sync*.*

In [28], the unconstrained synchronization problem can be decided in polynomial time by verifying that every pair of states can be synchronized. In essence, we generalize this algorithm here for returning constraint automata.

▶ **Lemma 16.** *Let $A = (\Sigma, Q, \delta)$ be a DCSA. If the constraint automaton $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ is returning, then A is synchronizing with respect to $\mathcal{B}$ if and only if for all $q, q' \in Q$ we find $w \in \Sigma^*$ with $\mu(p_0, w) = p_0$ such that $\delta(q, w) = \delta(q', w)$.*

Proof sketch. If $u \in L(\mathcal{B})$ is a synchronizing word for A, u collapses any pair of states, but this is also true for $w = uv$ with $\mu(p_0, w) = p_0$ that must exist as $\mathcal{B}$ is returning. Repeatedly using this idea to prolong collapsing words, the somewhat more involved argument for proving the converse implication can be adapted from the unconstrained setting. ◀

As we just have to check pairs of states we can devise a polynomial-time algorithm to decide $L(\mathcal{B})$-Constr-Sync of a returning automaton $\mathcal{B}$.

▶ **Theorem 17.** *If $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ is returning, then $L(\mathcal{B})$-*Constr-Sync *$\in$ P.*

Proof. Let $A = (\Sigma, Q, \delta)$ be an DCSA with $n = |Q|$. Let $m = |P|$. From A, we construct the DCSA $A^{\leq 2} = (\Sigma, \binom{Q}{\leq 2}, \delta')$. Then, for each two-element set $\{q_1, q_2\} \in \binom{Q}{\leq 2}$, define $A' = A^{\leq 2}_{\{q_1,q_2\},Q}$, identifying Q with all 1-element state sets. We check for each two-element set $\{q_1, q_2\} \in \binom{Q}{\leq 2}$ if $L(A') \cap L(\mathcal{B}_{p_0,\{p_0\}}) \neq \emptyset$. By Lemma 16, the DCSA A is synchronizing with respect to $\mathcal{B}$ if and only if each intersection is non-empty. Each of the $\binom{n}{2}$ intersections can be checked by using the product-automaton construction in time $\mathcal{O}\left((n + \binom{n}{2})m\right)$. ◀

Our considerations can be turned into a polynomial-time algorithm for computing a synchronizing word for A with respect to $\mathcal{B}$, which implies the following result.

▶ **Corollary 18.** *If the constraint automaton $\mathcal{B}$ is returning, a shortest synchronizing word with respect to $\mathcal{B}$ is polynomially bounded in the size of the input automaton.*

▶ **Theorem 19.** *Let $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ be a PDFA. Then, $L(\mathcal{B})$-*Constr-Sync *$\in$ NP if there is a $\sigma \in \Sigma$ such that for all states $p \in P$, if $L(\mathcal{B}_{p,\{p\}})$ is infinite, then $L(\mathcal{B}_{p,\{p\}}) \subseteq \{\sigma\}^*$.*

Proof. By assumption, the letters in $\Sigma \setminus \{\sigma\}$ do not appear in any pumpable substring. Hence, their number in a word in $L(\mathcal{B})$ is bounded by $|P| = m$. Therefore, any word $w \in L(\mathcal{B})$ can be partitioned into at most $2m-1$ substrings $w = u_1 v_1 \dots u_{m-1} v_{m-1} u_m$ with $u_i \in \{\sigma\}^*$ and $v_i \in (\Sigma \setminus \{\sigma\})^*$ for all $i \leq m$. Note that $|v_i| \leq m-1$, for all $i < m$. Let $A = (\Sigma, Q, \delta)$ be a yes-instance of $L(\mathcal{B})$-Constr-Sync with $|Q| = n$. Let k be the number of sun-structures in A_σ. Let $w \in L(\mathcal{B})$ be a synchronizing word for A partitioned as mentioned above.

▷ Claim 1. If for some $i \leq m$, $|u_i| > (mn)^k + n$, then we can replace u_i by some $u_i' \in \{\sigma\}^*$ with $|u_i'| \leq (mn)^k + n$, yielding a word $w' \in L(\mathcal{B})$ that synchronizes A.

We will now show that we can decide whether A is synchronizing with respect to $\mathcal{B}$ in polynomial time using nondeterminism despite the fact that an actual synchronizing word might be exponentially large. This problem is circumvented by some preprocessing based on modulo arithmetics. This allows us to guess a binary representation $\mathrm{bin}(u_i)$ of $|u_i|$ instead of u_i itself. Hence, we guess $w_{\mathrm{bin}} = \mathrm{bin}(u_1)v_1 \dots \mathrm{bin}(u_{m-1})v_{m-1}\,\mathrm{bin}(u_m)$. Since all u_i are single exponential in size, the length of w_{bin} is polynomially bounded in the size of A.

▷ Claim 2. For each $q \in Q$, one can compute in polynomial time numbers $\ell(q), \tau(q) \leq n$ such that, given some number x in binary, based on $\ell(q), \tau(q)$, one can compute in polynomial time a number $y \leq n$ such that $\delta(q, a^x) = \delta(q, a^y)$.

As $\mathcal{B}$ is even fixed, we can do a similar preprocessing also for $\mathcal{B}$ in polynomial time.

The NP-machine guesses w_{bin} part-by-part, keeping track of the set S of active states of A and of the current state p of $\mathcal{B}$. Initially, $S = Q$ and $p = p_0$. When guessing the number $x_i = |u_i|$ in binary, by Claim 1 we guess $\log(|u_i|) \leq \log((mn)^k + n) \in O(n \log n)$ many bits. By Claim 2, we can update $S := \delta(S, \sigma^{x_i})$ and $p := \mu(p, \sigma^{x_i})$ in polynomial time. After guessing v_i, we can simply update $S := \delta(S, v_i)$ and $p := \mu(p, v_i)$ by simulating this input, as $|v_i| \leq m = |P|$, which is a constant in our setting. Finally, check if $|S| = 1$ and if $p \in F$. ◀

Observe that our NP-algorithm was guessing at most $b(n,k) \in O((k+1)\log(nm))$ many bits with $b(n,k) \leq m^2 \log(|\Sigma|) + (m-1)(k+1)(\log(m) + \log(n))$. As m and $|\Sigma|$ are constants and as $n, k < n$ depend on A, we can determinize this algorithm by testing $b(n,k)$ many bits.

▶ **Corollary 20.** *Under the assumptions of Theorem 19, $L(\mathcal{B})$-Constr-Sync is in XP with parameter k counting the number of sun-structures in A_σ for an input DCSA A.*

When $k = 1$, we face a one-cluster automaton, see [4].

After these more general thoughts, we focus on two-state constraint automata $\mathcal{B}$, giving a complete picture of the complexity of $L(\mathcal{B})$-Constr-Sync over alphabets Σ with $|\Sigma| \leq 3$.

4 Constraint Automata with Two States and Two or Three Letters

There are already very many 2-state PDFA. We explain why we need to consider only one automaton for each automaton code listed in Tables 1 and 2. Here, we consider 1 as the start state and $\{2\}$ as the set of final states and call this the *standard interpretation* of a code.

▶ **Lemma 21.** *Let $\mathcal{B} = (\Sigma, P, \mu)$ be some partial deterministic semi-automaton with two states, i.e., $P = \{1, 2\}$. Then, for each $p_0 \in P$ and each $F \subseteq P$, either $L(\mathcal{B}_{p_0,F})$-Constr-Sync $\in$ P, or $L(\mathcal{B}_{p_0,F})$-Constr-Sync $\equiv_m^{\log}$ $L(\mathcal{B}')$-Constr-Sync for a PDFA $\mathcal{B}' = (\Sigma, P', \mu', 1, \{2\})$.*

Hence, we only need to specify $\mathcal{B} = (\Sigma, \{1, 2\}, \mu)$ in the following. Let $\Sigma_{ij} := \{\, a \in \Sigma \mid \mu(i, a) = j \,\}$ for $1 \leq i, j \leq 2$. As $\mathcal{B}$ is deterministic, $\Sigma_{i1} \cap \Sigma_{i2} = \emptyset$. Consider easy cases first.

Table 1 List of all PDFAs with two states and a binary alphabet, with $\Sigma_{1,2} = \{a, b\}$ or $\Sigma_{1,2} = \{b\}$.

Automaton code	Why in P?	Automaton code	Why in P?
`[ * 2 ‡ 1 * ]`	$a \in \Sigma_{2,1}$ Propos. 22, (2)	`[ 2 2 ‡ 2 - ]`	Theorem 24
`[ * 2 ‡ * 1 ]`	$b \in \Sigma_{2,1}$ Propos. 22, (2)	`[ 2 2 ‡ - 2 ]`	Isomorphic to `[ 2 2 ‡ 2 - ]`
`[ * 2 ‡ 2 2 ]`	$\Sigma_{1,1} \cup \Sigma_{1,2} = \Sigma_{2,2}$ Propos. 22, (3)	`[ {2,-} 2 ‡ - - ]`	$\Sigma_{1,1} \cup \Sigma_{2,2} = \emptyset$ Propos. 22, (4)
`[ 1 2 ‡ {-,2} - ]`	Theorem 24	`[ - 2 ‡ 2 - ]`	Theorem 24
`[ 1 2 ‡ - 2 ]`	Theorem 24	`[ - 2 ‡ - 2 ]`	$\Sigma_{1,1} \cup \Sigma_{1,2} = \Sigma_{2,2}$ Propos. 22, (3)

▶ **Proposition 22.** *If one of the following conditions hold, then $L(\mathcal{B}_{1,\{2\}})$-*Constr-Sync *$\in$ P: (1) $\Sigma_{1,2} = \emptyset$, (2) $\Sigma_{2,1} \neq \emptyset$, (3) $\Sigma_{1,1} \cup \Sigma_{1,2} \subseteq \Sigma_{2,2}$, or (4) $\Sigma_{1,1} \cup \Sigma_{2,2} = \emptyset$.*

Proof. (1) If $\Sigma_{1,2} = \emptyset$ means $L(\mathcal{B}_{1,\{2\}}) = \emptyset$. (2) If $\Sigma_{2,1} \neq \emptyset$, then $\mathcal{B}$ is returning (Theorem 17). (3) Lemma 11 and Theorem 14 cover this case. (4) Now, $L(\mathcal{B}_{1,\{2\}})$ is finite. ◀

For $\mathcal{B} = (\Sigma, \{1, 2\}, \mu)$ and $x \in \Sigma_{1,2}$, let $\mathcal{B}^x$ denote the variation with transition function μ^x defined by $\mu^x = \mu \cap (\{\, (p, y, p) \mid p \in \{1, 2\}, y \in \Sigma \,\} \cup \{(1, x, 2)\})$. Then Lemma 13 implies:

▶ **Lemma 23.** *If $L(\mathcal{B}^x_{1,\{2\}})$-*Constr-Sync *$\in$ P for each $x \in \Sigma_{1,2}$ and if $\Sigma_{2,1} = \emptyset$, then $L(\mathcal{B}_{1,\{2\}})$-*Constr-Sync *$\in$ P.*

Lemma 23 gives some final arguments why we only study the standard interpretation.

For 2-state constraint automata with alphabet $\Sigma = \{a, b\}$, in order to avoid isomorphic automata and by Proposition 22, we can assume that either (1) $a \in \Sigma_{1,1}$ and $b \in \Sigma_{1,2}$ and $|\Sigma_{2,2}| \leq 1$ or (2) $a \in \Sigma_{1,2}$ but $b \notin \Sigma_{1,1}$ and $|\Sigma_{2,2}| > 0$. See Table 1.

The constrained synchronization problem for constraint automata with a binary alphabet is not easy in general, as we have seen already in Proposition 1 for 3-state constraint PDFA.

▶ **Theorem 24.** *For any two-state binary PDFA $\mathcal{B}$, $L(\mathcal{B})$-*Constr-Sync *$\in$ P.*

Proof. By Table 1, we only need to show the claim for $\mathcal{B}_1$ = `[ 1 2 ‡ 2 - ]`, $\mathcal{B}_2$ = `[ 1 2 ‡ - 2 ]`, $\mathcal{B}_3$ = `[ 1 2 ‡ - - ]`, $\mathcal{B}_4$ = `[ - 2 ‡ 2 - ]`, and $\mathcal{B}_5$ = `[ 2 2 ‡ 2 - ]`. Let $A = (\Sigma, Q, \delta)$ be a DCSA with $n := |Q| - 1$. Consider the first PDFA $\mathcal{B}_1$ with $L(\mathcal{B}_1) = a^*ba^*$. Let $a^\ell ba^m$ be some synchronizing word for A, then by Lemma 8, applied to A_a, we have $\delta(Q, a^\ell) = \delta(Q, a^j)$ for some $j \leq n$, and moreover, by a similar argument, we find $k \leq n$ with $\delta(\delta(Q, a^j b), a^m) = \delta(\delta(Q, a^j b), a^k)$. So, the word $a^j ba^k$ is synchronizing and according to Lemma 3 the word $a^n ba^n$ is also synchronizing. In order to decide synchronizability with respect to $\mathcal{B}_1$, we simply have to check this last word. With the same argument, for $\mathcal{B}_2$ we only have to test the word $a^n b^n$, for $\mathcal{B}_3$ the word $a^n b$, and for $\mathcal{B}_4$ the word ba^n. As $\mathcal{B}_5$ accepts the union of $L(\mathcal{B}_4)$ and a unary regular language, the claim follows with Corollary 9 and Lemma 13. ◀

Next, we give a full classification on the complexity of the constrained synchronization problem for constraint automata with two states and a ternary alphabet. As can be verified by a case-by-case analysis, the only automaton with a constrained synchronization problem in P not covered by the generalization results in Section 3 is `[ 1 2 - ‡ - - 2 ]`.

▶ **Theorem 25.** *Let $\mathcal{B}$ = `[ 1 2 - ‡ - - 2 ]`. Then $L(\mathcal{B})$-*Constr-Sync *is in P.*

Proof. The language accepted by the constraint automaton $\mathcal{B}$ = `[ 1 2 - ‡ - - 2 ]` is a^*bc^*. Let $A = (\Sigma, Q, \delta)$ be a DCSA, $n = |Q|$. By arguments along the lines of the proof of Theorem 24, one can show that there is a synchronizing word for A with respect to $\mathcal{B}$ if and only if $a^n bc^n$ synchronizes A. This condition is easy to check. ◀

Table 2 Constraint automata (2 states, 3 letters) causing PSPACE-hard synchronization.

Case	Automaton code	Language
1	[2 - - ‡ - 2 2]	$a(b+c)^*$
	[2 2 2 ‡ 2 2 -]	$(a+b+c)(a+b)^*$
	[2 2 - ‡ 2 - 2]	$(a+b)(a+c)^*$
2	[1 1 2 ‡ - - -]	$(a+b)^*c$
	[1 1 2 ‡ 2 - -]	$(a+b)^*ca^*$
	[1 1 2 ‡ 2 2 -]	$(a+b)^*c(a+b)^*$
	[1 1 2 ‡ - - 2]	$(a+b)^*cc^*$
3	[1 2 - ‡ 2 - 2]	$a^*b(a+c)^*$
	[1 2 2 ‡ 2 2 -]	$a^*(b+c)(a+b)^*$
4	[1 2 - ‡ - 2 2]	$a^*b(b+c)^*$
	[1 1 2 ‡ - 2 2]	$(a+b)^*c(b+c)^*$
	[1 2 2 ‡ - 2 2]	$a^*(b+c)(b+c)^*$

The leftover two-state automata over a ternary alphabet are listed in Table 2. For all of them, the corresponding constrained synchronization problem is PSPACE-complete (see Theorem 26). We want to point out that there is no constraint automaton with two states and a ternary alphabet for which the $L(\mathcal{B})$-Constr-Sync is not either PSPACE-complete or contained in P, as we covered all possible automata of this kind.

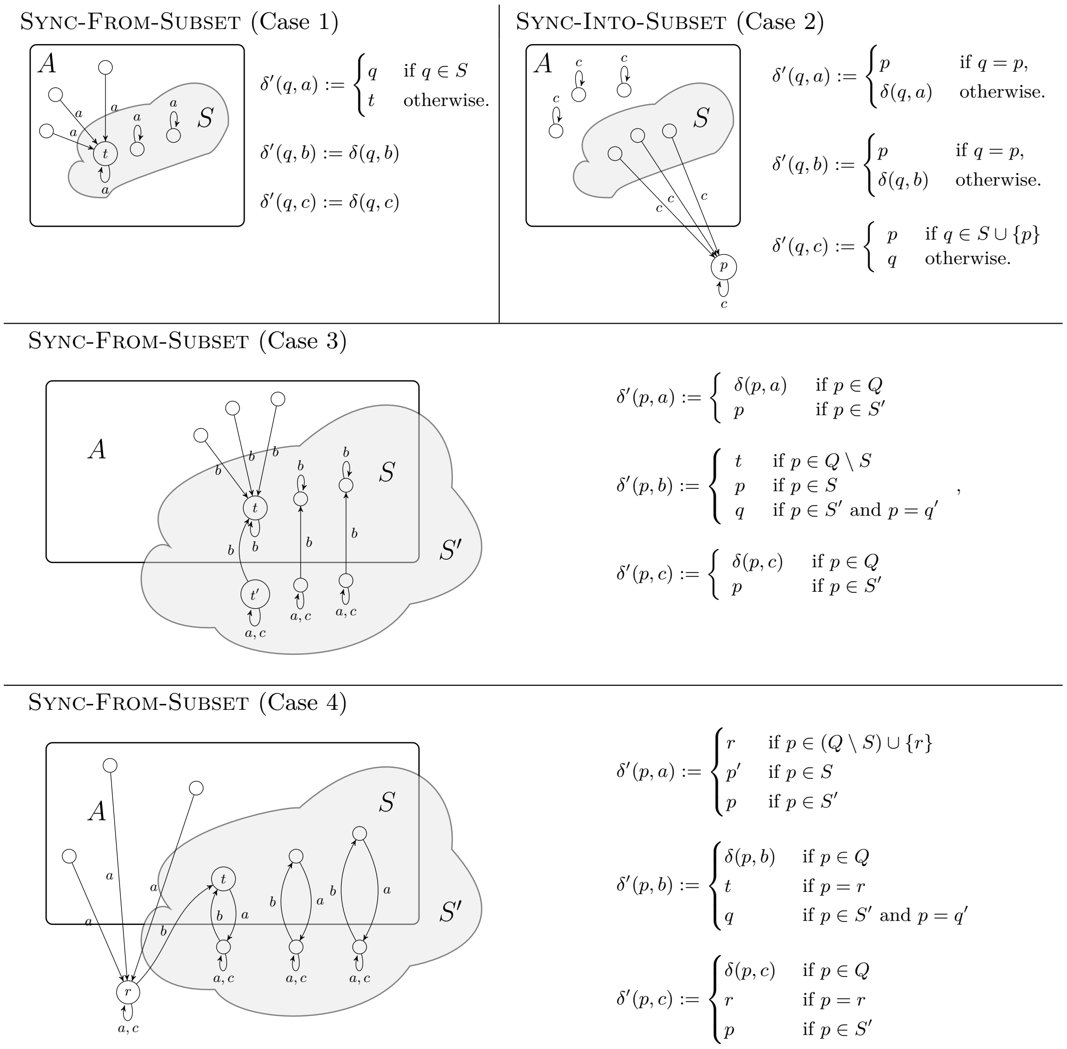
▶ **Theorem 26.** *For each constraint automaton $\mathcal{B}$ in Table 2 the problem $L(\mathcal{B})$-*Constr-Sync *is PSPACE-hard.*

Proof. We prove each case separately by giving an explicit reduction for one of the automata, the statement for the other automata of that case follows by the same argument. Our reductions are illustrated in Figure 2. Each reduction is starting out from (A, S), with $A = (\Sigma, Q, \delta)$ being a DCSA and $S \subseteq Q$. Depending on the considered case, (A, S) is either an instance of Sync-From-Subset (or of Sync-Into-Subset, resp.). We construct from A a DCSA $A' = (\Sigma', Q', \delta')$, with $\Sigma' = \Sigma \cup \{\sigma\}$ for an appropriately chosen letter $\sigma \notin \Sigma$, such that there exists $w \in \Sigma^*$ with $|\delta(S, w)| = 1$ (or $\delta(Q, w) \subseteq S$, resp.) if and only if A' is synchronizing with respect to $\mathcal{B}$. This construction is described and illustrated in Figure 2.

Case 1: Consider the first automaton $\mathcal{B} =$ [2 - - ‡ - 2 2]. Then, $L(\mathcal{B}) = a(b+c)^*$. We reduce from the PSPACE-complete problem Sync-From-Subset for the binary alphabet $\Sigma = \{b, c\}$. Since the constraint automaton forces us to read an a as the first letter, we start synchronizing A' with $\delta'(Q, a) = S$. After the first a, we are allowed to read any letter from Σ. Hence, if $|\delta(S, w)| = 1$ by a word $w \in \Sigma^*$, then $aw \in L(\mathcal{B})$ synchronizes A'. Conversely, if there exists a word v that synchronizes A' with respect to $\mathcal{B}$, then v must be of the form $v = au$ with $u \in \{b, c\}^*$. By the definition of δ', $|\delta(S, u)| = 1$.

The PSPACE-hardness of constrained synchronization with respect to the PDFA [2 2 2 ‡ 2 2 -] with the language $(a + b + c)(a + b)^*$ follows with the same reduction with the letters a and c interchanged. The same idea applies to [2 2 - ‡ 2 - 2].

Case 2: The language accepted by $\mathcal{B} =$ [1 1 2 ‡ - - -] is $L(\mathcal{B}) = (a+b)^*c$. We reduce from Sync-Into-Subset. Note that by construction if A' is synchronizing, p must be the unique synchronization state. The state p can only be reached by a transition with the letter c, but the constraint automaton only allows us to read one single c as the last letter of the synchronizing word. Hence, if there exists a synchronizing word w for A' with respect to $\mathcal{B}$, it is of the form uc with $u \in \{a, b\}^*$. Since $\delta'(Q, w) = p$, $\delta'(Q, u) \subseteq \{\, q \in Q \mid \delta'(q, c) = p \,\}$; by definition of δ', this equals the set $S \cup \{p\}$. Hence, u synchronizes the automaton A into a subset of S. Conversely, if w is a word that synchronizes A to a subset of S, by the construction of δ', the word wc synchronizes A' to $\{p\}$ and since $w \in \{a, b\}^*$, $wc \in L(\mathcal{B})$.



Sync-From-Subset (Case 1)

$$\delta'(q,a) := \begin{cases} q & \text{if } q \in S \\ t & \text{otherwise.} \end{cases}$$

$$\delta'(q,b) := \delta(q,b)$$

$$\delta'(q,c) := \delta(q,c)$$

Sync-Into-Subset (Case 2)

$$\delta'(q,a) := \begin{cases} p & \text{if } q = p, \\ \delta(q,a) & \text{otherwise.} \end{cases}$$

$$\delta'(q,b) := \begin{cases} p & \text{if } q = p, \\ \delta(q,b) & \text{otherwise.} \end{cases}$$

$$\delta'(q,c) := \begin{cases} p & \text{if } q \in S \cup \{p\} \\ q & \text{otherwise.} \end{cases}$$

Sync-From-Subset (Case 3)

$$\delta'(p,a) := \begin{cases} \delta(p,a) & \text{if } p \in Q \\ p & \text{if } p \in S' \end{cases}$$

$$\delta'(p,b) := \begin{cases} t & \text{if } p \in Q \setminus S \\ p & \text{if } p \in S \\ q & \text{if } p \in S' \text{ and } p = q' \end{cases},$$

$$\delta'(p,c) := \begin{cases} \delta(p,c) & \text{if } p \in Q \\ p & \text{if } p \in S' \end{cases}$$

Sync-From-Subset (Case 4)

$$\delta'(p,a) := \begin{cases} r & \text{if } p \in (Q \setminus S) \cup \{r\} \\ p' & \text{if } p \in S \\ p & \text{if } p \in S' \end{cases}$$

$$\delta'(p,b) := \begin{cases} \delta(p,b) & \text{if } p \in Q \\ t & \text{if } p = r \\ q & \text{if } p \in S' \text{ and } p = q' \end{cases}$$

$$\delta'(p,c) := \begin{cases} \delta(p,c) & \text{if } p \in Q \\ r & \text{if } p = r \\ p & \text{if } p \in S' \end{cases}$$

Figure 2 Schematic illustration of our reductions. Transitions inherited from A are not shown.

We can only reach the synchronizing state by reading the letter c and for each automaton of this case we are only allowed to read one single letter c. Therefore, allowing additional letters a and b in the synchronizing word after reading the letter c does not change the synchronizability of A' and hence the same construction works for the constraint automata `[ 1 1 2 ‡ 2 - - ]` and `[ 1 1 2 ‡ 2 2 - ]`. The same holds if we allow only additional letters c (and no a or b) after the first c. In A', c only leads in the synchronization state from states in S and is the identity on other states. Therefore, $\delta(q, cc) = \delta(q, c)$ for any state q and the construction of Case 2 also works for the constraint automaton `[ 1 1 2 ‡ - - 2 ]`.

Case 3: The language accepted by $\mathcal{B}$ = `[ 1 2 - ‡ 2 - 2 ]` is $L(\mathcal{B}) = a^*b(a+c)^*$. We reduce from Sync-From-Subset for $\Sigma = \{a, c\}$ similar to the one in Case 1, but we have to ensure that the whole set S is active after reading the letter b, since a preceding a might already merge some states in S. The idea is to add for each state $q \in S$ a new state q' for which we stay in q' with the letters a, c and go to q with the letter b. Therefore, we ensure that $\delta(Q, a^i b) = S$ for every integer i. Since, starting from the whole state set,

A' is precisely in the state set S after the first and only b letter, the rest of the argument follows as in Case 1. For the constraint automaton [1 2 2 ‡ 2 2 -], accepting the language $a^*(b+c)(a+b)^*$, the same idea applies.

Case 4: The language accepted by $\mathcal{B} =$ [1 2 - ‡ - 2 2] is $L(\mathcal{B}) = a^*b(b+c)^*$. Here, we do not have a special letter which appears exactly once in a word from $L(\mathcal{B})$. We will use the optional a letters in order to jump into S, since a word from $L(\mathcal{B})$ that does not contain any a must synchronize the whole state set and therefore also the subset S. We reduce from the problem Sync-From-Subset for the alphabet $\Sigma = \{b, c\}$. We decompose the state set Q with the letter a in S and $Q \setminus S$. The states in S are stored in an annotated copy S' of S. The other states are gathered up in a new state r. With the first b, we restore the set S as the set of active states. The remainder of a synchronizing word then synchronizes S.

If the set S in A is synchronizable to a single state by a word $u \in \{b,c\}^*$, then the word $abu \in L(\mathcal{B})$ synchronizes A' since $\delta'(Q', ab) = S$. For the other direction, assume A' is synchronizing with respect to $\mathcal{B}$ by a word w. Then w is of the form ubv with $u \in a^*$, $v \in \{b,c\}^*$. If $u \neq \epsilon$, then $\delta(Q', ub) = S$ and v synchronizes S to a single state. If $u = \epsilon$, then $S \subseteq \delta'(Q', b) \subseteq Q$ since we never synchronized any states into r and we leave all states in the set $S' \cup \{r\}$ with b and are not able to reach them again. In particular $\delta'(S', b) = S$. Therefore, bv synchronizes S to a single state without ever leaving Q. The same idea can be applied to the constraing automata [1 1 2 ‡ - 2 2] and [1 2 2 ‡ - 2 2]. ◀

5 Generalizations to Lift Results

In this section, we aim for more general results, either by lifting existing cases by homomorphic images, or by identifying common patterns. For a map $\varphi : \Sigma \to \Gamma^*$ we identify it with its natural homomorphism extension $\varphi : \Sigma^* \to \Gamma^*$ without further mentioning.

▶ **Theorem 27.** *Let $L \subseteq \Gamma^*$ and $\varphi : \Sigma^* \to \Gamma^*$ be an homomorphism such that $\varphi(\varphi^{-1}(L)) = L$. Then L-Constr-Sync $\leq_m^{\log}$ $\varphi^{-1}(L)$-Constr-Sync.*

A typical application of the preceding theorem is to lift hardness results from smaller to bigger alphabets; e.g., knowing PSPACE-hardness for the constraint language $a(b+c)^*$ lifts to PSPACE-hardness for the constraint language $a(b+c+d)^*$ via $\varphi : a \mapsto a, b \mapsto b, c \mapsto c, d \mapsto c$.

▶ Remark 28. It is impossible to further generalize the previous result from homomorphisms to mappings induced by deterministic gsm. Such a machine allows to map $(a+b)(a+b)^*$ to $a(b+c)^*$, but the constraint $(a+b)(a+b)^*$ yields a synchronization problem in P.

▶ **Theorem 29.** *Let $L \subseteq \Sigma^*$. Let $\varphi : \Sigma \to \Gamma^*$ be an homomorphism such that $\varphi(\Sigma)$ is a prefix code. Let $c \in \Gamma$ with $\{c\}\Gamma^* \cap \varphi(\Sigma) = \emptyset$. Let $k := \max\{\, \ell \geq 0 \mid \exists u, v \in \Gamma^* : uc^\ell v \in \varphi(\Sigma) \,\}$. Then L-Constr-Sync $\leq_m^{\log}$ $\{c^{k+1}\}\varphi(L)$-Constr-Sync.*

In the special case where c does not occur in $\varphi(\Sigma)$ at all, it is sufficient to choose $k = 0$, i.e., to consider the language $\{c\}\varphi(L)$ as constraint language. With Theorem 29, we can transfer hardness results with constraint language L over arbitrary alphabets to hardness results with constraint languages over a binary alphabet.

▶ Remark 30. With the construction presented in Corollary 1 (see pp. 220-221) in [5] we can lift our hardness results for constrained synchronization with constraint automata with two states and a ternary alphabet to constrained synchronization problems with 6 states and a binary alphabet. More generally we can reduce the alphabet size of a constraint automata from $k = |\Sigma|$ to 2 by enlarging the size of its state set from $n = |Q|$ to $k \cdot n$ without affecting the hardness of the associated constrained synchronization problem.

Up to this point, we did not make use of the fact that constraint languages considered in this paper are given by finite automata. This changes from here onward.

It is hardness-preserving to plug sub-automata of some kind in front of an automata with a hard constrained synchronization problem. A partial automaton A is called *carefully synchronizing* if there exists a synchronizing word w for A such that the transition function of A is defined for w on every state of A.

▶ **Theorem 31.** *Let $\mathcal{B} = (\Sigma^{\mathcal{B}}, P^{\mathcal{B}}, \mu^{\mathcal{B}}, p_0^{\mathcal{B}}, F)$ and $\mathcal{C} = (\Sigma^{\mathcal{C}}, P^{\mathcal{C}}, \mu^{\mathcal{C}}, p_0^{\mathcal{C}}, \emptyset)$ be PDFAs with $P^{\mathcal{B}} \cap P^{\mathcal{C}} = \emptyset$. For $p_x \in P^{\mathcal{C}}$ let $\nu \subseteq \{p_x\} \times (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}) \times \{p_0^{\mathcal{B}}\}$ define the automaton $\mathcal{B}' = (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}, P^{\mathcal{B}} \cup P^{\mathcal{C}}, \mu^{\mathcal{B}} \cup \mu^{\mathcal{C}} \cup \nu, p_0^{\mathcal{C}}, F)$. If the following three conditions are satisfied:*

1. *automaton $\mathcal{B}'$ is deterministic,*
2. *automaton $\mathcal{C}' = (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}, P^{\mathcal{C}} \cup \{p_0^{\mathcal{B}}\}, \mu^{\mathcal{C}} \cup \nu \cup \{p_0^{\mathcal{B}}\} \times (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}) \times \{p_0^{\mathcal{B}}\})$ is carefully synchronizing, and*
3. *there exists a synchronizing word $v = v_1 \dots v_n$ for $\mathcal{C}'$ such that $v_1 \dots v_{n-1} \in L(\mathcal{C}_{p_x})$, where $\mathcal{C}_{p_x}$ results from $\mathcal{C}$ by adding p_x to the set of final states,*

then $L(\mathcal{B})$-Constr-Sync $\leq L(\mathcal{B}')$-Constr-Sync.

Proof. Note that the start state of $\mathcal{B}'$ is the start state of $\mathcal{C}$, but the final states of $\mathcal{B}'$ are the ones from $\mathcal{B}$. Let $A = (\Sigma^{\mathcal{B}}, Q, \delta)$ be a DCSA. We extend A to a DCSA $A' = (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}, Q', \delta')$ in the following way. For every state $q \in Q$ we add a copy of $\mathcal{C}$ to A', where a self-loop is added for every yet undefined transition in A'. The $\mathcal{C}$-copy is connected to q with the transitions in ν where the target $p_0^{\mathcal{B}}$ is replaced by q. Since the automaton $\mathcal{B}'$ is deterministic by condition (1), the $\mathcal{C}$-copies, which are added to A, are also deterministic and so is A'.

It remains to show that A is synchronizing with respect to $\mathcal{B}$ if and only if A' is synchronizing with respect to $\mathcal{B}'$. For the only if direction, assume $w \in L(\mathcal{B})$ is a synchronizing word for A. Considering A', condition (2) states that there exists a word v that, applied to all states of a copy of $\mathcal{C}$, leads every state of this copy through the exit state p_x into the original states of A. Further, condition (3) specifies that the last state leaves through p_x with the last letter of v and that this last state is the image of the start state. Hence v is the label of a path from $p_0^{\mathcal{C}}$ to $p_0^{\mathcal{B}}$ in $\mathcal{B}'$ and $vw \in L(\mathcal{B}')$. Starting in all states of A', the active states in each $\mathcal{C}$-copy act synchronously. Hence, $\delta'(Q', v) = Q$. Note that no state of a $\mathcal{C}$-copy is reachable by a state of Q. Since A' acts like A on Q, $|\delta'(Q, w)| = 1$ and vw is a synchronizing word for A' with respect to $\mathcal{B}'$. For the other direction, we refer to the long version of this paper. ◀

As an illustration, we apply Theorem 31 to a family of languages.

▶ **Corollary 32.** *Let the language-family $\mathcal{L}$ consists of languages $L_i := (b^*a)^i$ with $i \geq 2$. The constrained synchronization problem for all languages in $\mathcal{L}$ is NP-complete.*

6 Conclusions and Prospects

We have commenced a study of synchronization under regular constraints. The complexity landscape of 2-state constraint automata with at most ternary input alphabets is completely understood. In particular, binary alphabets yield polynomial-time solvable synchronization problems, while ternary alphabets split the constrained synchronization problems into polynomial-time solvable and PSPACE-complete cases. As already seen in the introduction, this picture changes with 3-state automata, giving an NP-complete scenario with binary alphabets. Our general results also imply PSPACE-complete synchronization problems for binary constraint automata with at least six states. In the following theorem, we present a three state constraint automaton with a binary alphabet for which the associated constrained synchronization problem is PSPACE-complete, also because of Theorem 10.

▶ **Theorem 33.** *Let* $\mathcal{B} = $ `[ - 2 ‡ 3 3 ‡ 2 - ]` *be a three state PDFA over the alphabet* $\{0, 1\}$ *with start state* 1 *and final state* 3*. The problem* $L(\mathcal{B})$-Constr-Sync *is PSPACE-hard.*

Hence, binary 3-state constraint automata offer easy synchronization problems as well as problems complete for NP and for PSPACE. However, we have no complete complexity picture here, giving a natural research question. Motivated by a remark of Rystsov [23] in a related setting, one could also ask if there are regular language constraints that define synchronization problems that are complete for other levels of the polynomial-time hierarchy. We presented several criteria for a regular language L such that L-Constr-Sync $\in P$ as well as generalization-results to transfer the obtained hardness results for fixed L to larger classes of constraint languages, but a full classification of the complexity of L-Constr-Sync for regular constraint languages L is still an open research problem.

References

1 Marco Almeida, Nelma Moreira, and Rogério Reis. Enumeration and generation with a string automata representation. *Theoretical Computer Science*, 387(2):93–102, 2007. doi:10.1016/j.tcs.2007.07.029.

2 Dmitry S. Ananichev, Mikhail V. Volkov, and Vladimir V. Gusev. Primitive digraphs with large exponents and slowly synchronizing automata. *Journal of Mathematical Sciences*, 192(3):263–278, 2013. doi:10.1007/s10958-013-1392-8.

3 Frédérique Bassino and Cyril Nicaud. Enumeration and random generation of accessible automata. *Theoretical Computer Science*, 381(1-3):86–104, 2007. doi:10.1016/j.tcs.2007.04.001.

4 Marie-Pierre Béal, Mikhail V. Berlinkov, and Dominique Perrin. A quadratic upper bound on the size of a synchronizing word in one-cluster automata. *International Journal of Foundations of Computer Science*, 22(2):277–288, 2011. doi:10.1142/S0129054111008039.

5 Mikhail V. Berlinkov. Approximating the minimum length of synchronizing words is hard. *Theory of Computing Systems*, 54(2):211–223, 2014. doi:10.1007/s00224-013-9511-y.

6 Mikhail V. Berlinkov, Robert Ferens, and Marek Szykuła. Complexity of preimage problems for deterministic finite automata. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 117 of *LIPIcs*, pages 32:1–32:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.32.

7 Vincent D. Blondel and Natacha Portier. The presence of a zero in an integer linear recurrent sequence is NP-hard to decide. *Linear Algebra and its Applications*, 351-352:91–98, 2002. doi:10.1016/S0024-3795(01)00466-9.

8 Ján Černý. Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis*, 14(3):208–216, 1964.

9 Michiel de Bondt, Henk Don, and Hans Zantema. Lower bounds for synchronizing word lengths in partial automata. *International Journal of Foundations of Computer Science*, 30(1):29–60, 2019. doi:10.1142/S0129054119400021.

10 Michael Domaratzki, Derek Kisman, and Jeffrey Shallit. On the number of distinct languages accepted by finite automata with n states. *Journal of Automata, Languages and Combinatorics*, 7(4):469–486, 2002. doi:10.25596/jalc-2002-469.

11 Henning Fernau and Andreas Krebs. Problems on finite automata and the exponential time hypothesis. *Algorithms*, 10(1):24, 2017.

12 Zsolt Gazdag, Szabolcs Iván, and Judit Nagy-György. Improved upper bounds on synchronizing nondeterministic automata. *Information Processing Letters*, 109(17):986–990, 2009. doi:10.1016/j.ipl.2009.05.007.

13 Vladimir V. Gusev. Synchronizing automata of bounded rank. In Nelma Moreira and Rogério Reis, editors, *Implementation and Application of Automata - 17th International Conference, CIAA*, volume 7381 of *LNCS*, pages 171–179. Springer, 2012. doi:10.1007/978-3-642-31606-7_15.

14 Michael A. Harrison. A census of finite automata. *Canadian Journal of Mathematics*, 17:100–113, 1965. doi:10.4153/CJM-1965-010-9.

15 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2nd edition, 2001.

16 Ravindran Kannan and Richard J. Lipton. Polynomial-time algorithm for the orbit problem. *Journal of the ACM*, 33(4):808–821, August 1986. doi:10.1145/6490.6496.

17 Zvi Kohavi and Niraj K. Jha. *Switching and Finite Automata Theory.* Cambridge University Press, 3rd edition, 2009.

18 Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 254–266. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.16.

19 Kim Guldstrand Larsen, Simon Laursen, and Jirí Srba. Synchronizing strategies under partial observability. In Paolo Baldan and Daniele Gorla, editors, *Concurrency Theory - 25th International Conference, CONCUR*, volume 8704 of *LNCS*, pages 188–202. Springer, 2014. doi:10.1007/978-3-662-44584-6_14.

20 Pavel Martyugin. Complexity of problems concerning reset words for some partial cases of automata. *Acta Cybernetica*, 19(2):517–536, 2009.

21 Pavel V. Martyugin. Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory of Computing Systems*, 54(2):293–304, 2014. doi:10.1007/s00224-013-9516-6.

22 Rogério Reis, Nelma Moreira, and Marco Almeida. On the representation of finite automata. *CoRR*, abs/0906.2477, 2009. URL: http://arxiv.org/abs/0906.2477.

23 Igor K. Rystsov. Polynomial complete problems in automata theory. *Information Processing Letters*, 16(3):147–151, 1983. doi:10.1016/0020-0190(83)90067-4.

24 Sven Sandberg. Homing and synchronizing sequences. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *LNCS*, pages 5–33. Springer, 2005. doi:10.1007/11498490_2.

25 Yaroslav Shitov. An improvement to a recent upper bound for synchronizing words of finite automata. Technical Report arXiv:1901.06542, Cornell University, 2019.

26 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the fifth annual ACM Symposium on Theory of Computing, STOC*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

27 Marek Szykuła. Improving the upper bound on the length of the shortest reset word. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS*, volume 96 of *LIPIcs*, pages 56:1–56:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.56.

28 Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA*, volume 5196 of *LNCS*, pages 11–27. Springer, 2008. doi:10.1007/978-3-540-88282-4_4.

A Constant-Time Colored Choice Dictionary with Almost Robust Iteration

Torben Hagerup
Institut für Informatik, Universität Augsburg, 86135 Augsburg, Germany
hagerup@informatik.uni-augsburg.de

Abstract

A (colored) choice dictionary is a data structure that is initialized with positive integers n and c and subsequently maintains a sequence of n elements of $\{0,\ldots,c-1\}$, called colors, under operations to inspect and to update the color in a given position and to return the position of an occurrence of a given color. Choice dictionaries are fundamental in space-efficient computing. Some applications call for the additional operation of dynamic iteration, i.e., enumeration of the positions containing a given color while the sequence of colors may change. An iteration is robust if it enumerates every position that contains the relevant color throughout the iteration but never enumerates a position more than once or when it does not contain the color in question. We describe the first choice dictionary that executes every operation in constant amortized time and almost robust iteration in constant amortized time per element enumerated. The iteration is robust, except that it may enumerate some elements a second time. The data structure occupies $n\log_2 c+O((\log n)^2)$ bits. The time and space bounds assume that $c=O((\log n)^{1/2}(\log\log n)^{-3/2})$.

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

Keywords and phrases Succinct data structures, space efficiency, in-place chain technique, bounded universes, amortization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.64

1 Introduction

Following similar earlier definitions [3, 5] and concurrently with that of [2], the (colored) choice dictionary was introduced by Hagerup and Kammer [10]. It appears to be fundamental in space-efficient computing and has already been shown to have a number of applications [2, 5, 10, 11, 12]. Its precise definition is as follows:

▶ **Definition 1.1.** *A (colored) choice dictionary is a data structure that can be initialized with integers $n, c \in \mathbb{N}=\{1,2,\ldots\}$ and subsequently maintains a sequence $(e_1,\ldots,e_n)$ of n integers drawn from $\{0,\ldots,c-1\}$, initially $(0,\ldots,0)$, under the following operations:*

color(i) $(i\in\{1,\ldots,n\})$: *Returns e_i.*
setcolor(j,i) $(j\in\{0,\ldots,c-1\}$ *and* $i\in\{1,\ldots,n\})$: *Replaces e_i by j.*
choice(j) $(j\in\{0,\ldots,c-1\})$: *With $S_j=\{i\in\{1,\ldots,n\}: e_i=j\}$, returns an (arbitrary) element of S_j if $S_j\neq\emptyset$, and 0 if $S_j=\emptyset$.*

We call the elements of $\{0,\ldots,c-1\}$ *colors*. The choice dictionary is similar to an array of colors that supports reading (*color*) and writing (*setcolor*), but with a crucial additional operation (*choice*) to locate an occurrence of a given color. Our terminology will sometimes pretend that the elements of $\{1,\ldots,n\}$ have colors. For $j=0,\ldots,c-1$, we define $S_j=\{i\in\{1,\ldots,n\}: e_i=j\}$ as above and call S_j a *color class*.

In some applications of choice dictionaries, notably to the computation of a breadth-first search or BFS forest of a given graph [9, 10], it is essential for a choice dictionary to support the additional operation of iteration over a given color class (while other computation takes place in an interleaved fashion). The BFS algorithms of [9, 10], for instance, repeatedly

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 64; pp. 64:1–64:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

iterate over the *gray* vertices, those with a given distance d from a start vertex, in order to identify the vertices at distance $d+1$. This is not straightforward because the algorithms – in order to save colors – also color the vertices at distance $d+1$ gray. In other words, the set of gray vertices changes while it is being iterated over. Hagerup and Kammer [10] coined the phrase *robust iteration* to describe the ideal that one would like to have in such a situation. An iteration over a dynamic set S is robust if every element that belongs to S throughout the iteration is certain to be enumerated by the iteration, while no element is enumerated more than once or at a time when it does not belong to S. Elements that belong to S during part but not all of the iteration may or may not be enumerated. We say that an iteration works in constant time if there are constant-time operations to initialize for a new iteration, to test whether more elements remain to be enumerated, and – if so – to enumerate the next element.

By the information-theoretic lower bound, a choice dictionary with parameters n and c must occupy at least $\lceil n \log c \rceil$ bits ("log" always denotes the binary logarithm function $\log_2$). Kammer and Sajenko [13] recently described a choice dictionary that occupies $n \log c + O(\log n)$ bits and executes every operation including iteration in constant time, but is severely restricted:

- The number c of colors must be a power of 2.
- Moreover, c must be a constant.
- The iteration is *static*: While an iteration is underway, no changes to colors are allowed.

We present a new choice dictionary that also executes every operation including iteration in constant time and alleviates the drawbacks listed above. The number of colors is not restricted to be a power of 2, it is not required to be a constant – but we do impose the condition $c = O((\log n)^{1/2}(\log \log n)^{-3/2})$ – and the iteration is dynamic and almost robust. By "almost robust" we mean that the definition of robust iteration is satisfied, except that some elements may be enumerated a second time (the BFS algorithms discussed above can tolerate this). Compared to the choice dictionary of [13], the new data structure has a few drawbacks of its own:

- It is more complicated.
- The operation times are amortized, not worst-case.
- The number of bits needed is $n \log c + O((\log n)^2)$, not $n \log c + O(\log n)$.

The third drawback should be seen in light of the fact that even if only the operations *color* and *setcolor* (i.e., those of an array) are to be supported in constant time, every known data structure that does not restrict c to be a power of 2 requires $n \log c + \Omega((\log n)^2)$ bits. As for the condition $c = O((\log n)^{1/2}(\log \log n)^{-3/2})$, it may be noted that the performance of comparable choice dictionaries [7, 10] also degrades sharply if the number of colors exceeds similar thresholds.

The main novelty of our work lies in the almost robust iteration. Allowing amortization changes the ground rules of iteration and was crucial to obtaining the results described here. It becomes feasible to begin a dynamic iteration over a color class S_j by an "internal" static iteration that can serve, among other things, to determine $|S_j|$. Enumerating the elements that belong to S_j at that time, i.e., at the start of the iteration, would satisfy the conditions of robustness, but the elements that are enumerated must be handed to a caller one by one, and there is no space to store them temporarily (cf. the BFS algorithms discussed above). Once S_j is allowed to change while it is being iterated over, it becomes difficult to keep track of which elements have already been enumerated and therefore to prevent elements from being enumerated repeatedly and perhaps an unbounded number of times (i.e., the iteration might not terminate). Enumerating the elements of S_j in sorted order would take care of this problem, but constant time per element is not enough to sort. Our iteration sorts a small part of S_j initially. If the changes to S_j during the iteration are relatively few, the partial sorting is sufficient to guarantee robustness. If there are many changes to S_j, on the other hand, these can "pay for" a complete sorting of what remains of S_j. An element is enumerated at most once before and at most once after the complete sorting.

The only previous choice dictionary with robust iteration was devised by Hagerup and Kammer [10], but for worse time and/or space bounds (e.g., constant time together with $n \log c + O(n/(\log n)^t)$ bits for constant c and arbitrary fixed $t \in \mathbb{N}$). It should be noted that in contrast with the robust iteration of [10], if the iteration described here is terminated early, its time bound is the same as if it had run to completion, i.e., incomplete iteration is not supported efficiently.

2 Preliminaries

Our model of computation is the standard word RAM [1, 6] with a word length of $w \in \mathbb{N}$ bits, where we assume that w is large enough to allow all memory words in use to be addressed. As part of ensuring this, we assume that $w \geq \log n$. The word RAM has constant-time operations for addition, subtraction and multiplication modulo 2^w, division with truncation ($(x, y) \mapsto \lfloor x/y \rfloor$ for $y > 0$), left shift modulo 2^w ($(x, y) \mapsto (x \ll y) \bmod 2^w$, where $x \ll y = x \cdot 2^y$), right shift ($(x, y) \mapsto x \gg y = \lfloor x/2^y \rfloor$), and bitwise Boolean operations (AND, OR and XOR (exclusive or)). The machine is also assumed to be able to compute $\lfloor \log x \rfloor$ in constant time for every given $x \in \{1, \ldots, 2^w - 1\}$.

Like all comparable data structures, the new choice dictionary depends on standard low-level word-RAM routines, some of which are conveniently collected in the following lemma.

▶ **Lemma 2.1** ([10], Lemma 3.2). *Let m and f be given integers with $1 \leq m, f < 2^w$ and suppose that a sequence $A = (a_1, \ldots, a_m)$ with $a_i \in \{0, \ldots, 2^f - 1\}$ for $i = 1, \ldots, m$ is given in the form of the (mf)-bit binary representation of the integer $\sum_{i=0}^{m-1} 2^{if} a_{i+1}$. Then the following holds:*

(a) *Let $I_0 = \{i \in \{1, \ldots, m\} : a_i = 0\}$. Then, in $O(1 + mf/w)$ time, we can test whether $I_0 = \emptyset$ and, if not, compute* $\min I_0$.

(b) *If an additional integer $k \in \{0, \ldots, 2^f - 1\}$ is given, then $O(1 + mf/w)$ time suffices to compute the integer $\sum_{i=0}^{m-1} 2^{if} b_{i+1}$, where $b_i = 1$ if $k \geq a_i$ and $b_i = 0$ otherwise for $i = 1, \ldots, m$.*

(c) *If $m < 2^f$ and an additional integer $k \in \{0, \ldots, 2^f - 1\}$ is given, then $\mathit{rank}(k, A) = |\{i \in \{1, \ldots, m\} : k \geq a_i\}|$ can be computed in $O(1 + mf/w)$ time.*

3 Informal Overview

This section tries to convey the basic intuition behind the new data structure. The necessary nitty-gritty details and calculations will be presented in the subsequent sections.

It is known from the work of Dodis, Pătraşcu and Thorup [4] that a sequence of n color values drawn from $\{0, \ldots, c - 1\}$ can be stored in a data structure that occupies $n \log c + O((\log n)^2)$ bits and supports the operations *color* and *setcolor* in constant time. Let us call this structure a *c-ary array*. Our task can be viewed as that of adding efficient *choice* and almost robust iteration to a c-ary array.

Suppose that, in addition to being represented in a c-ary array, the elements of each color class S_j, where $j \in \{0, \ldots, c-1\}$, are organized in a doubly-linked list L_j. Then executing *choice*(j) is trivial – return an arbitrary element of L_j – and with a little care we can also carry out a robust iteration over S_j via a traversal of L_j during which elements that drop out of S_j splice themselves out of L_j and elements that enter S_j are inserted at the beginning of L_j, where they will not be enumerated – they may be "trying to sneak in again" after having already been enumerated in the past.

The problem with the scheme outlined above is, of course, that the doubly-linked lists $L_0, \ldots, L_{c-1}$ would need far too much space – $\Theta(n \log n)$ bits – for two pointers for each of the n elements. In order to alleviate this problem, we divide the n elements into equal-sized groups and represent each group via a data structure called a *container.* Provided that groups are not too large, iteration within a container can be handled efficiently with techniques based on Lemma 2.1. If every container holds at least one element of a color class S_j, we can therefore iterate over S_j by iterating over all containers and enumerating the elements of S_j in each. Thus at this point the problem can be viewed as represented by the containers that are *j-free*, i.e., do not contain any occurrences of j. The list L_j that we would now like to have for each $j \in \{0, \ldots, c-1\}$ chains together those containers that are not j-free. In contrast with what was the case before, a container can belong to many lists, for which reason it may need up to $2c$ pointers rather than just two pointers.

Even though we have reduced the need for pointers from two per element to $2c$ per container, the basic problem remains that we have already used up practically all the available space and cannot afford to store even a single pointer per container. If some color is missing entirely from a container, however, $\log c$ bits per element is a tad more than what is strictly necessary to record the contents of the container. Using a more efficient encoding, we can compress the information stored in the container slightly to leave room for a couple of pointers – provided that containers are sufficiently large. The same is true if a color is not missing completely, but has very few occurrences, as we can then store the sequence of these "exceptional" occurrences explicitly while using the efficient encoding for the other elements in the container. If several colors are missing or rare in a container, we can pack the container even more tightly. As a result, the container has room for a couple of pointers for each such color, which in turn means that we can treat the colors somewhat independently.

Let $j \in \{0, \ldots, c-1\}$ and let us call a container H *j-sparse* if the number of occurrences of j in H is bounded by some suitably chosen r, and *j-abundant* otherwise. For a moment, let us make the unrealistic assumption that all j-sparse containers are to the left of (have smaller indices than) all j-abundant containers. Then we can keep in L_j those containers that are j-sparse (and therefore have room for the necessary pointers) but not j-free (since a main goal is to skip the j-free containers). To iterate over S_j, iterate both over L_j and over the j-abundant containers. The latter is easy because the j-abundant containers, by assumption, are consecutively numbered.

Even though the unrealistic assumption is unrealistic, we can still keep track of the number μ_j of j-sparse containers and imagine that the leftmost μ_j containers (say that these form the *left part*) "normally" are j-sparse. There may be containers that go against the norm, j-abundant containers in the left part – call these *j-masters* – and j-sparse containers in the right part, *j-slaves.* It is easy to see that the number of j-masters equals the number of j-slaves, and the *in-place chain technique* of Katoh and Goto [14] suggests to maintain a perfect matching between the j-masters and the j-slaves. This helps us to solve two problems: (1) A j-master, by virtue of not being in the right part, needs to belong to L_j in order to be iterated over, but has no room for pointers. (2) Because it is in the right part, a j-slave is iterated over, but may be j-free, in which case it cannot "pay for" its share of the iteration. To solve problem (1), we relocate some of the data of the master to the slave, which has room to spare. As for problem (2), even though the slave cannot "pay", its master can.

We cannot keep track of the number of occurrences of a color j in a container once that number exceeds r, and therefore replace the notions j-sparse and j-abundant by algorithmically more tractable j*-weak* and j*-strong*. A j-weak container is certain to be j-sparse, but a j-strong container may be j-sparse or j-abundant. If the number of occurrences of j in a j-weak container H grows beyond r, H is converted to being j-strong. A conversion of H back to j-weak, however, takes place only if and when H is later discovered to have fewer than r occurrences of j, and this in turn happens essentially only in connection with an iteration over H. The iteration can "pay" a constant for each element that is enumerated, and the operations that changed the colors of the other elements formerly of color j can "pay" a constant for each of these. In an amortized setting, therefore, a conversion between j-weak and j-strong can be allowed to have a cost of $O(r)$, and r is chosen accordingly. A budget of $O(r)$ also covers the conversion of a j-strong container, for the purpose of iteration over the container, from the very compact usual representation to one that supports efficient iteration.

A final problem is represented by containers that migrate from the left to the right side (as a consequence of a decrease in μ_j). Such a container might be iterated over on the left side, because it belongs to L_j, and later again as a container on the right side. In order to prevent elements from being enumerated repeatedly, the iteration is designed to enumerate elements roughly in sorted order. In more detail, first the number $|L_j|$ of containers in L_j is determined, and then the containers in L_j that belong to a "buffer zone" of width $|L_j|^2$ at the right end of the left part are sorted, which can happen in $O(|L_j|+1)$ time, and placed last in L_j. If the buffer zone is consumed entirely, i.e., migrates completely to the right part before it starts to take part in the iteration, this is evidence of so many color changes during the iteration that we can afford to sort the remaining containers in L_j. After that point in time, an element is enumerated at most once, but it may also have been enumerated once before the sorting. If some part of the buffer zone remains in the left part throughout the iteration over the part of L_j on its left, on the other hand, the order of iteration over the containers on its left is immaterial, and the iteration is robust.

4 The Data Representation

This section describes the organization of data in the choice dictionary after it has been initialized with parameters $n, c \geq 2$ with $c = O((\log n)^{1/2}(\log\log n)^{-3/2})$. We first describe a slightly bigger data structure that uses $n \log c + O(c(\log n)^2)$ bits.

For a positive integer $s = \Theta(c \log n)$, we divide the sequence $(e_1, \ldots, e_n)$ of n color values to be maintained into $N = \lfloor n/s \rfloor$ subsequences of exactly s consecutive color values each, with at most $s-1$ color values left over. The left-over color values can be handled separately in $O(s \log s) = O((\log n)^2)$ bits, essentially by keeping a doubly-linked list for each color of the positions in which the color occurs. We omit the easy details and assume in the following that n is a multiple of s. For $\ell = 1, \ldots, N$, the ℓth subsequence is stored in a data structure H_ℓ called a *container* (as anticipated in the previous section). Each container partitions the set $\{0, \ldots, c-1\}$ of colors into a set of *weak* colors and the complementary set of *strong* colors. If a color $j \in \{0, \ldots, c-1\}$ is weak in a container H, we call H j*-weak*; otherwise H is j*-strong* (again, these terms as well as notation introduced below were used already in Section 3). For $j = 0, \ldots, c-1$, we keep a global count $\mu_j \in \{0, \ldots, N\}$ of the number of j-weak containers. For $\ell \in \{1, \ldots, N\}$ and $j \in \{0, \ldots, c-1\}$, we say that H_ℓ is *j-left* if $\ell \le \mu_j$ and *j-right* if $\ell > \mu_j$. Finally, we call H_ℓ *j-free* if the sequence of color values stored in H_ℓ does not contain any occurrence of j.

Let $\ell \in \{1, \ldots, N\}$ and suppose that $(a_0, \ldots, a_{s-1})$ is the subsequence of color values stored in H_ℓ – thus $a_i = e_{(\ell-1)s+i+1}$ for $i = 0, \ldots, s-1$. If H_ℓ is j-weak, where $j \in \{0, \ldots, c-1\}$, we require the number $|\{i \in \{0, \ldots, s-1\} : a_i = j\}|$ of occurrences of j in H_ℓ to be bounded by a positive integer r with $r = \Theta(\log n/(c \log\log n))$.

If H_ℓ is j-strong and j-right for all $j \in \{0, \ldots, c-1\}$, H_ℓ is *atomic*, by which we mean that its representation, which we will call $D_{\overline{W}}$, consists of the single integer $x = \sum_{i=0}^{s-1} a_i c^i$. Provided that we keep a table of the powers $c^0, c^1, \ldots, c^{s-1}$, we can then inspect and update individual values in $(a_0, \ldots, a_{s-1})$ with a constant number of arithmetic operations. E.g., $a_i = \lfloor x/c^i \rfloor \bmod c$ for $i = 0, \ldots, s-1$. This does not necessarily translate into constant time, as we may not be able to operate on integers as large as c^s in constant time. To remedy this, we obtain s as $s_0 s_1$, where s_0 and s_1 are positive integers with $s_0 = \Theta(\log n/\log c)$ and $s_1 = \Theta(c \log c)$, and actually represent x through s_1 digits – called *big digits* – to base $C = c^{s_0} = n^{O(1)}$. We can operate on big digits in constant time, and it is easy to see that we can still inspect and update individual color values by operating only on the appropriate big digit. Now we need only the powers $c^0, \ldots, c^{s_0-1}$. For reasons that will become clear shortly, we store in fact a table of all powers bounded by C of the form i^k, where $i \in \{2, \ldots, c\}$ and $k \in \mathbb{N}$. The number of bits required for the table is easily seen to be $O(c(\log n)^2)$.

In all other cases, namely if H_ℓ is j-weak or j-left for at least one $j \in \{0, \ldots, c-1\}$, H_ℓ is *composite*, namely represented through a collection of four substructures: D_{C}, which stores information concerning the roles played by the various colors in H_ℓ; D_W and $D_{\overline{W}}$, which specify the distribution of weak and strong colors in H_ℓ, respectively; and D_{P}, which links H_ℓ to other containers. The details are as follows:

- First, for various subsets M of $\{0, \ldots, c-1\}$ we store $\mathit{rank}_M(j) = |\{i \in M : i \le j\}|$ for $j = 0, \ldots, c-1$ and $\mathit{select}_M(k) = \min\{j \in \mathbb{N} \cup \{0\} : \mathit{rank}_M(j) = k\}$ for $k = 1, \ldots, |M|$. Thus we store a table of rank_M, which numbers the elements of M consecutively, and a table of select_M, which realizes the inverse mapping. In detail, let W, $\overline{W}$, R and $\overline{R}$ be the sets of colors j for which H_ℓ is j-weak, j-strong, j-right and j-left, respectively, Then we store tables of *rank* and *select* for all of the sets W, $\overline{W}$, R and $\overline{R}$ as well as for all unions and intersections of two of these (we actually need only a few of the tables). D_{C} is the collection of these tables, which occupy $O(c \log c)$ bits and can be computed from W and R in $O(c)$ time.
- Second, D_W stores the set $\{(i, a_i) \mid i \in \{0, \ldots, s-1\}$ and $a_i \in W\}$ and, more abstractly, maintains a subset A of $\{0, \ldots, s-1\} \times \{0, \ldots, c-1\}$ and supports the following operations in constant time:
  - Given a pair $(i, j) \in A$, delete it from A.
  - Given a pair $(i, j) \in \{0, \ldots, s-1\} \times \{0, \ldots, c-1\}$, insert it in A, provided that before the operation no pair in A has first component i and fewer than r pairs in A have second component j.
  - Given $i \in \{0, \ldots, s-1\}$, return the pair of the form (i, j) in A or an indication that A contains no such pair.
  - Given $(i, j) \in \{0, \ldots, s-1\} \times \{0, \ldots, c-1\}$, return the pair of the form (i', j) in A with $i' \ge i$ for which i' is minimal or an indication that A contains no such pair.
  - Given $j \in \{0, \ldots, c-1\}$, return the number of pairs of the form (i, j) in A.

  D_W can be thought of as an embellished associative array for the weak colors. What makes its realization easy is the fact that the number of pairs in A at all times is bounded by cr, while every component of a pair is an integer of f bits, where $f = O(\log s) = O(\log\log n)$. We can simply store $|A|$, an integer of $O(\log(cr)) = O(\log\log n)$ bits, and two arrays A_1 and A_2 of cr f-bit entries each such that the first $|A|$ entries in A_1 contain the first components of the pairs in A in sorted order and the first $|A|$ entries in A_2 contain their second components in the corresponding order, i.e., so that the two components of each pair have the same index in A_1 and A_2. Each of A_1 and A_2 is stored as a single integer of $crf = O(\log n)$ bits, and without loss of generality (choose s_0 and hence s sufficiently large) we will assume that D_{C} and D_W fit together in a single big digit.

To delete a pair (i, j) from A, we can form the bitwise XOR of A_1 with an integer that contains the integer i in each of $|A|$ f-bit fields and can be obtained in constant time with a multiplication, and subsequently use the algorithm of Lemma 2.1(a) to discover the position in A_1 and A_2 of the pair (i, j) (or an indication that (i, j) does not occur in A). After subtracting 1 from the variable that stores $|A|$, we can easily carry out the remainder of the operation, namely prizing out a field from A_1 and A_2 and closing the gap, in constant time with a combination of bitwise Boolean operations, applications of bit masks and shifts.

When inserting a pair (i, j) in A, rather than an exact match of i in A_1, we want to find the correct place in which to insert i while keeping A_1 sorted. We therefore use part (c) instead of part (a) of Lemma 2.1, but can otherwise proceed similarly as in the case of a deletion. The remaining operations listed above can be implemented to work in constant time in similar ways and are left to the interested reader (the last operation can also be handled by maintaining an additional array of counts).

- Third, the distribution of strong colors in H_ℓ is recorded in $D_{\overline{W}}$ in one of several formats, chosen in dependence of $|\overline{W}|$ and c: If $|\overline{W}| = 0$, there are no strong colors in H_ℓ, and $D_{\overline{W}}$ is a dummy data structure (denoted by $\emptyset$, say) that takes up no space. If $|\overline{W}| = 1$ and $c = 2$, we also take $D_{\overline{W}} = \emptyset$, since the necessary information can be gleaned from D_W – an element whose color is stored in H_ℓ has the strong color precisely if it does not have the weak color. If $1 \le |\overline{W}| \le c - 2$, $D_{\overline{W}}$ stores the sequence $(a'_0, \ldots, a'_{s-1})$, where $a'_i = 0$ if $a_i \in W$ and $a'_i = \mathit{rank}_{\overline{W}}(a_i)$ if $a_i \in \overline{W}$, for $i = 0, \ldots, s - 1$, essentially in the form of the integer $y = \sum_{i=0}^{s-1} a'_i(|\overline{W}|+1)^i$. In other words, every weak color is encoded via a zero, the strong colors are encoded via the integers $1, \ldots, |\overline{W}|$, and the sequence of codes is represented similarly as for atomic containers. Finally, if $|\overline{W}| \ge \max\{c-1, 2\}$, we redefine a'_i as $\mathit{rank}_{\overline{W}}(a_i) - 1$ for $a_i \in \overline{W}$ and store $y = \sum_{i=0}^{s-1} a'_i|\overline{W}|^i$. Thus for $|\overline{W}| = c - 1 \ge 2$ we give up on distinguishing between the weak color and one strong color.

Informally, the idea behind D_W and $D_{\overline{W}}$ is that weak colors have only few occurrences, which can be stored in little space by listing them explicitly (in D_W), while this allows the strong colors to be represented more economically via smaller codes (in $D_{\overline{W}}$). Having two distinct conventions for y is a necessary complication: We must be able to store H_ℓ in less space even if only a single color is weak, which precludes the use of the first convention (i.e., reserving a code value for weak colors) when $|\overline{W}| \ge c - 1$. On the other hand, the second convention (not distinguishing between weak colors and the smallest strong color j_0) cannot be used when $|W| = c - |\overline{W}|$ is large, since it requires us to find the occurrences of j_0 by testing all occurrences of a zero in $(a'_0, \ldots, a'_{s-1})$ and filtering out those that correspond to weak colors – we cannot allow too many "false positives". Moreover, the case $|\overline{W}| = c - 1$ must be handled separately in $D_{\overline{W}}$ for $c = 2$ (namely not at all), because $|\overline{W}| = 1$ cannot be used as the basis of a positional system.

As was the case for x, if $1 \le |\overline{W}| \le c - 2$ or $|\overline{W}| \ge \max\{c-1, 2\}$ it is necessary to express y through a sequence of big digits but, unless $|\overline{W}| = c$, the task is now hampered by rounding issues. Assume that $|\overline{W}| \le c - 1$. The s *small digits* to base $|\overline{W}| + h$, where $h = 1$

if $1 \le |\overline{W}| \le c-2$ and $h = 0$ if $|\overline{W}| = c-1 \ge 2$, are to be distributed over a number of big digits to base $C = c^{s_0}$. A single big digit can accommodate $\lfloor (s_0 \ln c)/\ln(|\overline{W}|+h)\rfloor$ small digits. Now $\ln(|\overline{W}|+h) = \ln(c-|W|+h) = \ln c+\ln(1-(|W|-h)/c) \le \ln c-(|W|-h)/c \le \ln c - |W|/(2c)$ and

$$\frac{s_0 \ln c}{\ln(|\overline{W}|+h)} \ge \frac{s_0 \ln c}{\ln c - |W|/(2c)} = \frac{s_0}{1-|W|/(2c\ln c)} \ge s_0(1+|W|/(2c\ln c)).$$

Since $|W| \ge 1$ and $s_0/(c\ln c) = \Omega(\ln n/(c(\ln c)^2)) = \omega(1)$, we may conclude that a big digit can accommodate $s_0(1+\Omega(|W|/(c\ln c)))$ small digits (even taking the $\lfloor\cdots\rfloor$ operation into account). Moreover, since

$$\frac{s}{s_0(1+\Omega(|W|/(c\ln c)))} = \frac{s_1}{1+\Omega(|W|/(c\ln c))} = s_1(1-\Omega(|W|/(c\ln c)))$$

and $s_1 = \Theta(c\ln c)$, by choosing s_1 and hence s sufficiently large we can ensure that the total number of big digits necessary to accommodate all s small digits is at most $s_1 - 2K|W|$ for an arbitrary constant $K \in \mathbb{N}$ of our choice (but independent of s).

- Fourth and finally, D_{P} is an array that maps each color $j \in \{0,\ldots,c-1\}$ for which H_ℓ is j-weak or j-left to a *block* of three pointers, each of which points to a container or has the value *null* (points to nothing). By choosing s_0 and therefore s sufficiently large, we can assume that a block fits in a big digit. Two of the pointers stored for a color j are used to organize certain j-left containers in a doubly-linked list L_j, namely those that are j-strong or not j-free (informally, those that might contain occurrences of j). The third pointer, called a *matching pointer*, is used (has a value different from *null*) only if H_ℓ is j-left and j-strong – then H_ℓ is called a *j-master* – or H_ℓ is j-right and j-weak – then H_ℓ is called a *j-slave*. As essentially noted before, the number of j-masters always equals the number of j-slaves, and the matching pointers form a perfect matching between the j-masters and the j-slaves, with each matching pointer of a master pointing to its corresponding slave, and vice versa.

In order to reduce the space needed for tables of powers of integers to $O((\log n)^2)$ bits, we replace the set $\{c, c-1,\ldots,2\}$ of possible bases used to represent the integer y of $D_{\overline{W}}$ by the smaller set $\{c, c-1, c-2, c-4, c-8,\ldots\}$: The base $c-i$, where $i \in \{1,\ldots,c-2\}$, is replaced by $c-2^{\lfloor \log i\rfloor}$. Then y takes up more space, but the inequality $2^{\lfloor \log(|W|-h)\rfloor} \ge |W|/2$ for $0 \le h \le \min\{|W|-1, 1\}$ shows that it is still the case that y can be represented in $s_1 - 2K|W|$ big digits, where W is the set of weak colors of the container under consideration. Now we need powers of only $O(\log c)$ integers, at most one of which is smaller than $\sqrt{c}$, so the number of bits needed comes to $O(\log c \cdot s_0 \log C) = O((\log n)^2)$. This ends the description of the structure of a composite container H_ℓ.

The use of masters and slaves is an element of the in-place chain technique of Katoh and Goto [14]. Another element requires us to distinguish between *conceptual containers* (intended until this point) and *physical containers*. The number of big digits needed to store a composite container H_ℓ was bounded above by $1 + (s_1 - 2K|W|) + |W \cup \overline{R}|$, where W and $\overline{R}$ are the sets of colors j for which H_ℓ is j-weak and j-left, respectively. This number varies from one (conceptual) container to another, which is precisely the problem – we cannot store the containers in little space so as to allow constant-time access to a given container. This problem is solved by a transfer of data between containers. For each $j \in \{0,\ldots,c-1\}$ and each pair (H, H') of containers such that H and H' are a j-master and its j-slave, respectively, K big digits are relocated from H to H'. A container chooses the at most Kc big digits to relocate as a master to be among at least $s_1 - 1 - c = \omega(c)$ big digits reserved for $D_{\overline{W}}$ and not to overlap with the at most Kc digits received as a slave – if s_1 and hence s are chosen sufficiently large, this is always possible. The first condition means that no "bookkeeping information" is moved to a place where it might be difficult to find (most obviously, the pointer to a slave should not be relocated to the slave), and the second condition ensures that a master can always access one of its relocated big digits in constant time – it must follow a pointer chain of length 1 only.

With the transfer of data described above, the number of big digits needed by a composite container H_ℓ changes to

$$\begin{aligned} &1 + (s_1 - 2K|W|) + |W \cup \overline{R}| - K|\overline{W} \cap \overline{R}| + K|W \cap R| \\ &\le 1 + s_1 - K(|W| + |\overline{W} \cap \overline{R}|) + |W \cup \overline{R}| \\ &= s_1 + 1 - (K-1)|W \cup \overline{R}| \end{aligned}$$

where, again, W, $\overline{W}$, R and $\overline{R}$ are those of H_ℓ. Because H_ℓ is composite, $|W \cup \overline{R}| \ge 1$, so that the quantity above is at most $s_1 - (K-2)$. After the transfer of data, therefore, every conceptual container H can be stored in a physical container of exactly s_1 big digits, and if H is composite, it additionally offers $K-2$ big digits of freely usable *extra space*.

The $P = Ns_1$ big digits of the N physical containers are maintained in an instance of the ingenious data structure of Dodis et al. [4] embodied in the lemma below. The number of bits needed is $P \log C + O((\log P)^2) = Ns_1 s_0 \log c + O((\log(Ns_1))^2) = n \log c + O((\log n)^2)$.

▶ **Lemma 4.1** ([4], Theorem 1). *There is a data structure that, given arbitrary positive integers P and C with $C = P^{O(1)}$, can be initialized in $O(\log P)$ time and subsequently maintains an array of P elements drawn from $\{0, \dots, C-1\}$ in $P \log C + O((\log P)^2)$ bits such that individual array elements can be read and written in constant time.*

When nothing else is stated, in the following "container" means "conceptual container". A subtle point is that an atomic container must be prevented from "posing as" a composite container – there is no space for an atomic/composite bit, and every bit combination is in use for an atomic container. If an atomic container H "claims" to be j-left for some $j \in \{0, \dots, c-1\}$, such a j can be found in constant time with D_{C}, and the claim can be falsified in constant time by inspection of μ_j. Otherwise, if H "claims" to be j-weak for some $j \in \{0, \dots, c-1\}$ and hence a j-slave, the claim can be falsified in constant time by the lack of a reverse pointer to H in its purported j-master. Thus we can always test in constant time whether a given container is atomic or composite.

In addition to the "per-container" data detailed above, we store globally, for each color, a *choice buffer* and an *iteration buffer*, maintained in a special *loose representation* that is wasteful of space, but allows efficient operations corresponding to *choice* and robust iteration. Each buffer is *derived* from a container H_ℓ whose index ℓ is remembered. If the data structure $D_{\overline{W}}$ of H_ℓ stores the sequence $(a'_0, \dots, a'_{s-1})$ of codes, the loose representation of H_ℓ consists of the integer $\sum_{i=0}^{s-1} 2^{if} a'_i$, where $f = \lceil \log c \rceil$. The derivation of a choice or iteration buffer can be carried out in $O(\log s_0)$ time per big integer and $O(s_1 \log s_0) = O(c \log c \log \log n) = O(r)$ time altogether with the algorithm of Lemma 4.2 below, which is a word-parallel version (i.e., essentially independent computations take place simultaneously in different regions of a word) of a simple divide-and-conquer procedure.

▶ **Lemma 4.2** ([7], Lemma 3.3 with $p = 1$). *Given positive integers c, d, f and s with $c, d \ge 2$ and $f \ge \lceil \log_2 \max\{c, d\} \rceil$ and an integer of the form $\sum_{j=0}^{s-1} a_j c^j$, where $0 \le a_j < \min\{c, d\}$ for $j = 0, \dots, s-1$, the integer $\sum_{j=0}^{s-1} a_j d^j$ can be computed in $O(\lceil sf/w \rceil (\log s + (\log(2 + sf/w))^2))$ time.*

Because each code a_i' is readily available in an f-bit field in the loose representation, we can use the algorithm of Lemma 2.1(a), applied to a buffer for a color j and derived from a container H, to carry out a *sweep* of H that reports all elements stored in H whose color is j when they are hit by the sweep. This takes $O(1 + sf/w) = O(c \log c)$ time plus time proportional to the number of occurrences reported plus, possibly, $O(r)$ time to skip occurrences of weak colors with the same code as j, and clearly satisfies the conditions for a robust iteration over H. We generally suspend the sweep as soon as we have found one occurrence of j and resume the sweep later to find the next occurrence, for which reason we also store with each buffer how far the current sweep over the buffer has progressed. In the case of a choice buffer, the sweep is resumed one position earlier so that it reports the same element as last time unless the color of that element has changed. The space needed is $O(sf) = O(c \log c \log n)$ bits per buffer and $O(c^2 \log c \log n) = O((\log n)^2)$ bits for all $2c$ buffers.

When a buffer for a color j derived from a container H is replaced by a different buffer, j is made weak in H if this is possible, i.e., if j occurs at most r times in H. The procedure for doing this is described in the following section.

Initially all containers are 0-strong and j-weak for $j = 1, \ldots, c-1$, and there are no masters or slaves.

5 Conversion Between Weak and Strong Colors

If the number of occurrences of a color j in a container H_ℓ is bounded by r, H_ℓ may be changed from j-weak to j-strong, or vice versa. This section describes the details.

Assume first that H_ℓ is composite both before and after the change. Let the sets W, R, etc., and the substructures D_{C}, D_W, etc., be those of H_ℓ. First, W changes in the obvious way by gaining or losing an element, and D_{C} is recomputed accordingly from skratch in $O(c)$ time. Second, up to r pairs are inserted in or removed from D_W, which can happen in $O(r)$ time.

Consider now the necessary update of $D_{\overline{W}}$. Recall that in the nontrivial cases, $D_{\overline{W}}$ is represented by an integer y composed of s small digits to base $|\overline{W}|$ or $|\overline{W}| + 1$ distributed over a number of big digits. Using the algorithm of Lemma 4.2, we first convert y to the loose representation, which takes $O(r)$ time as before. Informally, we must either create a gap among the code values of the existing strong colors to make room for a new strong color or, conversely, prize out the code value of a color that stops being strong and subsequently close the gap that it leaves. Both of these can be done in $O(r)$ time with the algorithm of Lemma 2.1(b). We omit the details. For $c \geq 3$, these vary a little depending on whether or not $|\overline{W}|$ switches between $c-2$ and $c-1$, i.e., depending on whether or not the interpretation of y switches between the first and the second convention. After the creation of a gap for the code value of a new strong color, the $r+1$ occurrences of that code are “planted” one by one in $O(r)$ time. Similarly, the occurrences of the code of a color that becomes weak are replaced by the code of weak colors (i.e., zero) before the code value of the formerly strong color is prized out. When these changes have taken place, the algorithm of Lemma 4.2 is applied again to convert the loose representation back to the usual representation of $D_{\overline{W}}$ as an integer y. Altogether, the update of $D_{\overline{W}}$ can happen in $O(r)$ time.

The final task is to repair the matching between j-masters and j-slaves. In the special case in which H_ℓ switches not only between j-weak and j-strong, but simultaneously between j-left and j-right (this happens if μ_j switches between ℓ and $\ell - 1$), H_ℓ is neither a j-master nor a j-slave neither before nor after the switch, and nothing needs to be done. Otherwise let $H \neq H_\ell$ be the container that switches as above between j-left and j-right because of the change in μ_j. There are two main cases: (1) If H_ℓ switches from j-weak to j-strong and therefore μ_j decreases by 1 and H switches from j-left to j-right, H_ℓ either becomes a j-master (it is j-left) or stops being a j-slave (it is j-right), and H either stops being a j-master (it is j-strong) or becomes a j-slave (it is j-weak). In all combinations there is a master without a slave and a slave without a master, and they are matched. (2) If H_ℓ switches from j-strong to j-weak and H switches from j-right to j-left, the situation is completely analogous: H_ℓ either becomes a j-slave (it is j-right) or stops being a j-master (it is j-left), and H either stops being a j-slave (it is j-weak) or becomes a j-master (it is j-strong). Again, there is a slave without a master and a master without a slave, and they are matched. In each case the matching pointers are adjusted and the data relocated from masters to slaves is moved appropriately. In addition, the substructure D_{C} of H must be updated. All of this can be done in $O((s \log c)/w) = O(r)$ time.

If H_ℓ is atomic either before or after the conversion, the algorithmic steps are very similar, except that the bookkeeping information for H_ℓ is only implicit when H_ℓ is atomic. A similar situation obtains when H_ℓ has $D_{\overline{W}} = \emptyset$ either before or after the conversion. In all cases, the total time needed for the complete conversion is $O(r)$.

6 The Operations

This section describes how to execute the operations *color*, *setcolor* and *choice* and how to carry out an almost robust iteration.

color

In order to execute a call *color*(i), we identify the container H that stores the ith color value. The substructures D_W, $D_{\overline{W}}$ and D_{C} mentioned in the following are those of H. Accessing D_W, we can determine whether the color j to be returned is weak in H and, if so, j itself. If j is strong and $|\overline{W}| \geq 2$, we can access $D_{\overline{W}}$ to learn the code of j in H, from which j itself can be recovered using D_{C}. If j is strong and $|\overline{W}| = 1$, j is the unique element of $\overline{W}$. Once j is known, it is returned. The execution of *color* takes constant time.

setcolor

Suppose that a call of *setcolor* changes the color of some $i \in \{1, \ldots, n\}$ from j to j'. If j' is weak in the relevant container H and the operation causes the number of occurrences of j' in H to exceed r, H is first converted from j'-weak to j'-strong, as described in the previous section. The rest of the operation is straightforward. If j is weak in H, the pair (i, j) is removed from the substructure D_W of H, and if j' is weak in H, (i, j') is inserted in D_W. Similarly, if j' is strong in H and $|\overline{W}| \geq 2$, the code of j' in H is obtained from the substructure D_{C} of H, after which it is easy to carry out the appropriate update of the sequence $(a'_0, \ldots, a'_{s-1})$ stored in the substructure $D_{\overline{W}}$. A call of *setcolor* needs $O(r)$ time if it carries out a conversion of a container and constant time if not.

choice

The execution of a call *choice*(j) continues the sweep over the choice buffer for the color j, if any, until an occurrence of j is found or the sweep is complete. In the former case the position found is returned. In the latter case, as far as possible, a container H is selected that either belongs to L_j or is j-right – if there is no such container, there are no occurrences of j, and 0 is returned. If H is j-weak, we distinguish between two cases: If H belongs to L_j, it is not j-free, and an occurrence of j in H can be found in constant time and returned. If H is j-right and therefore a j-slave, it is replaced by its corresponding j-master. Now H is strong, a new choice buffer for the color j is derived from H, and the procedure is restarted recursively.

Almost Robust Iteration

Consider an iteration over a given color class S_j. In preparation for the iteration, we step through the list L_j to determine $|L_j|$. Then, using $O(\log n)$ bits of additional space, we split L_j into two lists L_j' and L_j'', with L_j' consisting of precisely those containers H_ℓ in L_j for which $\ell \leq \mu_j - |L_j|^2$. The time needed to do this is $O(|L_j| + 1)$. Since the indices ℓ of the containers in L_j'' are distinct integers in the range $\{\mu_j - |L_j|^2 + 1, \ldots, \mu_j\}$ of size $|L_j|^2$, we can sort the containers in L_j'' by their indices with 2-pass radix sorting in $O(|L_j| + 1)$ time. If K is chosen sufficiently large, the $O(|L_j| \log n)$ bits of additional space needed for this can be supplied by the extra space, discussed shortly before Lemma 4.1, of the containers H_ℓ with $\mu_j - |L_j|^2 < \ell \leq \mu_j$. At this point L_j is updated to be the concatenation of L_j' with the sorted L_j''.

Now the iteration proper can begin. We move a *token* through L_j, always robustly enumerating the elements of S_j in the *current container* H, the container that holds the token. If H is j-strong, this is done by deriving the iteration buffer for j from H and sweeping it, as described in Section 4. If H is j-weak, it is done similarly, by always remembering the last occurrence enumerated and using the fourth operation listed for D_W to enumerate the next occurrence. We will speak of a sweep also in this case. If a container H drops out of L_j because it becomes j-free while it holds the token, the token is first passed on to the successor of H in L_j, if any.

If a container H is j-strong or not j-free and H is to be inserted in L_j because it becomes j-left, then H is inserted at the end of L_j, i.e., so that it will still be swept. If a container is to be inserted in L_j because it stops being j-free, however, it is inserted at the beginning of L_j, i.e., so that it will not be swept.

Suppose that the iteration starts with $\mu_j = \widehat{\mu_j}$. If μ_j decreases all the way to $\widehat{\mu_j} - |L_j|^2$ (i.e., to the "border" between L_j' and L_j'') before all containers in L_j' have been fully swept, the enumeration is suspended and the containers in L_j whose sweep has not yet begun are sorted. Although it is not strictly necessary, this can happen "in-place" with Bubblesort in $O(|L_j|^2)$ time. Then the enumeration is resumed. When the iteration reaches the end of L_j, it proceeds to sweep the containers $H_{\overline{\mu_j}+1}, \ldots, H_N$ in that order, where $\overline{\mu_j}$ is the value of μ_j when the iteration over L_j finishes.

7 Analysis of Correctness and Execution Times

The correctness of the implementation of *color*, *setcolor* and *choice* is easy to see or has already been argued. Consider therefore an iteration over a color class S_j. Every element i that belongs to S_j throughout the iteration is stored in a container H that belongs to L_j or is j-right at the beginning of the iteration, and it is stored in H throughout any sweep over H. During the iteration H can drop out of L_j only by becoming j-right, and if H stops being j-right it is inserted at the end of L_j, where it will still be swept. Thus i is enumerated.

Let t_0 be the point in time when the iteration over L_j' ends or when what remains of L_j is sorted, whatever happens first. Until t_0 only containers in L_j' are iterated over, and any containers that enter L_j during this period are inserted at the beginning of L_j, where they will not be iterated over. Thus no element is enumerated more than once before t_0. After t_0 the enumeration happens strictly in increasing order, so again no element is enumerated more than once. Thus an element is enumerated at most once before t_0 and at most once after t_0. It is easy to see that no element is ever enumerated when it does not belong to S_j. It follows that the iteration is almost robust, as claimed.

The basic idea of the amortized timing analysis is simple: Disregarding iteration, all operations that involve only weak colors take constant time. Consider a point in time at which a container H is converted from j-weak to j-strong for some color j. At that time H contains more than r elements of color j. Because of this, before a conversion of H from j-weak to j-strong can happen again, a buffer for the color j must be derived from H, and the data structure must operate on each of the more than r elements of color j in H by enumerating the element, returning it (*choice*) or changing its color. We can "charge" all of the following to these more than r operations: The derivation of up to two buffers for color j from H, the sweep over the buffers, exclusive of the time spent reporting occurrences of j found there, and a possible conversion of H from j-strong to j-weak and back.

A problematic issue with the argument in the previous paragraph is that operations on elements in a buffer are called upon to pay for the derivation of the buffer, which happened earlier. For every color j other than 0 there is no problem here, since more than r operations – which can pay for the cost – must change the colors of elements in H to j before j can become strong in H for the first time. This argument does not apply to the color 0 because all elements have color 0 initially. However, using the structure D_W of H, a suitably represented first buffer for the color 0 can be derived from H in a time that is at most proportional to a constant plus the number of color changes executed on H since the initialization, so these operations can be charged with the cost.

Certain costs of an iteration are not covered by the analysis above (and, indeed, an iteration may temporarily "go into dept", which is why we cannot support incomplete iteration efficiently). First, there is the cost associated with sorting. The first sorting of part of L_j happens in $O(|L_j|+1)$ time, which is acceptable because every container that does not drop out of L_j before it receives the iteration token eventually contributes at least one occurrence of j to be enumerated or – if it turns out to be j-free – is converted from j-strong to j-weak, the cost of which was considered above. The second sorting of part of L_j is carried out, in $O(|L_j|^2)$ time, only after at least $|L_j|^2$ operations on elements of color j have been executed during the iteration, and the cost of the sorting can be charged to these operations. Second, there is the cost associated with sweeping j-slaves and j-weak former j-slaves that turn out to be j-free (so that the cost of the sweep cannot be charged to the occurrences found). This cost, however, can be charged to the sweeping of the corresponding masters or to the color changes that created or destroyed the matching links to these masters.

Theorem 4 of [8] furnishes a variant of the data structure of Lemma 4.1 that initializes all array elements to zero and can itself be initialized in constant time. Storing the global book-keeping information such as $\mu_0, \ldots, \mu_{c-1}$ in another instance of this data structure and interpreting the all-zero initial values appropriately allows the choice dictionary developed here to be initialized in constant time. We omit the details.

▶ **Theorem 7.1.** *There is a choice dictionary that, for arbitrary given positive integers n and c with $c = O((\log n)^{1/2}(\log\log n)^{-3/2})$, can be initialized with parameters n and c in constant time and subsequently occupies $n\log_2 c + O((\log n)^2)$ bits and executes color, setcolor and choice in constant amortized time and complete almost robust iteration (an element may be enumerated a second time) in constant amortized time per element enumerated.*

References

1 D. Angluin and L. G. Valiant. Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings. *J. Comput. Syst. Sci.*, 18(2):155–193, 1979. doi:10.1016/0022-0000(79)90045-X.

2 Niranka Banerjee, Sankardeep Chakraborty, and Venkatesh Raman. Improved Space Efficient Algorithms for BFS, DFS and Applications. In *Proc. 22nd International Conference on Computing and Combinatorics (COCOON 2016)*, volume 9797 of *LNCS*, pages 119–130. Springer, 2016. doi:10.1007/978-3-319-42634-1_10.

3 Preston Briggs and Linda Torczon. An Efficient Representation for Sparse Sets. *ACM Lett. Program. Lang. Syst.*, 2(1-4):59–69, 1993. doi:10.1145/176454.176484.

4 Yevgeniy Dodis, Mihai Pătraşcu, and Mikkel Thorup. Changing Base Without Losing Space. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 593–602. ACM, 2010. doi:10.1145/1806689.1806770.

5 Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-Efficient Basic Graph Algorithms. In *Proc. 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *LIPIcs*, pages 288–301. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

6 Torben Hagerup. Sorting and Searching on the Word RAM. In *Proc. 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1998)*, volume 1373 of *LNCS*, pages 366–398. Springer, 1998. doi:10.1007/BFb0028575.

7 Torben Hagerup. Small Uncolored and Colored Choice Dictionaries. *Computing Research Repository (CoRR)*, abs/1809.07661 [cs.DS], 2018. arXiv:1809.07661.

8 Torben Hagerup. Highly Succinct Dynamic Data Structures. In Leszek Antoni Gasieniec, Jesper Jansson, and Christos Levcopoulos, editors, *Fundamentals of Computation Theory - 22nd International Symposium, FCT 2019, Copenhagen, Denmark, August 12-14, 2019, Proceedings*, volume 11651 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2019. doi:10.1007/978-3-030-25027-0_3.

9 Torben Hagerup. Fast Breadth-First Search in Still Less Space. In *Proc. 45th International Workhop on Graph-Theoretic Concepts in Computer Science (WG 2019)*, LNCS. Springer, 2019, to appear.

10 Torben Hagerup and Frank Kammer. Succinct Choice Dictionaries. *Computing Research Repository (CoRR)*, abs/1604.06058 [cs.DS], 2016. arXiv:1604.06058.

11 Torben Hagerup, Frank Kammer, and Moritz Laudahn. Space-Efficient Euler Partition and Bipartite Edge Coloring. *Theor. Comput. Sci.*, 754:16–34, 2019. doi:10.1016/j.tcs.2018.01.008.

12 Frank Kammer, Dieter Kratsch, and Moritz Laudahn. Space-Efficient Biconnected Components and Recognition of Outerplanar Graphs. *Algorithmica*, 81(3):1180–1204, 2019. doi:10.1007/s00453-018-0464-z.

13 Frank Kammer and Andrej Sajenko. Simple 2^f-Color Choice Dictionaries. In *Proc. 29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *LIPIcs*, pages 66:1–66:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

14 Takashi Katoh and Keisuke Goto. In-Place Initializable Arrays. *Computing Research Repository (CoRR)*, abs/1709.08900 [cs.DS], 2017. arXiv:1709.08900.

Fault Tolerant and Fully Dynamic DFS in Undirected Graphs: Simple Yet Efficient

Surender Baswana
Department of Computer Science & Engineering, IIT Kanpur, Kanpur, India
sbaswana@cse.iitk.ac.in

Shiv Gupta
Department of Computer Science & Engineering, IIT Kanpur, Kanpur, India
shivguptamails@gmail.com

Ayush Tulsyan
Department of Computer Science & Engineering, IIT Kanpur, Kanpur, India
ayushtulsyan01@gmail.com

Abstract

We present an algorithm for a fault tolerant Depth First Search (DFS) Tree in an undirected graph. This algorithm is drastically simpler than the current state-of-the-art algorithms for this problem, uses optimal space and optimal preprocessing time, and still achieves better time complexity. This algorithm also leads to a better time complexity for maintaining a DFS tree in a fully dynamic environment.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Depth first search, DFS, Dynamic graph algorithms, Fault tolerant

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.65

Related Version A full version of the paper is available at `https://arxiv.org/abs/1810.01726`.

Funding *Surender Baswana*: This research work was partly done during a visit to University of Paderborn supported by a fellowship from the Alexander von Humboldt Foundation.

1 Introduction

Depth First Search (DFS) is a widely popular graph traversal method. The traversal routine, formalized by Tarjan [44] in 1972, has played a crucial role in various graph problems including reachability, bi-connectivity, topological sorting, and strongly connected components.

Given an undirected graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, DFS traversal on the graph takes $\mathcal{O}(m + n)$ time and results in a DFS tree of G.

Most of the graphs in real-world applications keep changing with time. Vertices and edges keep entering and leaving the graph at various time steps. This dynamic aspect has motivated researchers to design algorithms that can update the solution of the corresponding problem efficiently after each such change in the graph. There are two models used for solving these graph problems, namely, fault tolerant algorithms and dynamic graph algorithms.

The *fault tolerant* version of any problem $\mathcal{P}$ on a graph G is to construct a compact data structure, using which, for any given set of failed edges or vertices F, one can efficiently report the solution of $\mathcal{P}$ on $G \setminus F$. Many elegant fault tolerant algorithms have been designed for problems including connectivity [13, 21, 25], shortest paths [10, 11, 16, 20, 28], and spanners [12, 15].

The *dynamic* version of any problem $\mathcal{P}$ on G is modeled as follows. For any online sequence of updates (insertion or deletion of an edge/vertex), one has to report the solution of $\mathcal{P}$ efficiently after every update. Note that, unlike the fault tolerant version, the updates are persistent in dynamic version, i.e., after each update, the solution has to be reported taking into account all the updates made so far. Algorithms which handle both insertion and deletion of vertices/edges are called *fully dynamic* graph algorithms, whereas the algorithms that handle either insertions or deletions are called *partially dynamic* graph algorithms, more specifically *incremental* or *decremental* graph algorithms, respectively. The prominent results for dynamic graph problems include connectivity [22, 30, 32], reachability [39, 41], shortest path [19, 29, 40], matching [6, 9, 43], spanner [8, 26, 38], min cut [45], Even-Shiloach tree [23], minimum spanning tree [31, 35], and graph sparsifiers [1].

1.1 Previous Results on Fault Tolerant and Dynamic DFS

Franciosa *et al.* [24] presented an incremental algorithm for maintaining a DFS tree in a directed acyclic graph (DAG) which takes overall $\mathcal{O}(mn)$ time for any sequence of m edge insertions. For undirected graphs, Baswana and Khan [7] presented an incremental algorithm for a DFS tree which takes overall $\mathcal{O}\left(n^2\right)$ time for any sequence of edge insertions. Baswana and Choudhary [4] designed a randomized decremental algorithm for a DFS tree in a DAG with overall expected $\mathcal{O}(mn \log n)$ time for any sequence of edge deletions.

None of the partially dynamic algorithms stated above achieves an $o(m)$ bound over the worst-case complexity of a single update. Moreover, there were no fully dynamic or fault tolerant algorithms for DFS in undirected graphs until recently. In 2016, the first fault tolerant algorithm was presented that takes $\mathcal{O}\left(nk \log^4 n\right)$ time to report a DFS tree for any set of k failed vertices or edges [3]. The time complexity was further improved by Chen *et al.* [17] to $\mathcal{O}\left(nk \log^2 n\right)$. Both [3, 17] require a data structure occupying $\mathcal{O}\left(m \log^2 n\right)$ bits. Nakamura and Sadakane [34] reduced the space occupied by the data structure to $\mathcal{O}(m \log n)$ bits which is indeed optimal [1]. Using the standard technique of periodic rebuilding, the fault tolerant algorithms presented in [3, 17, 34] were also extended to $o(m)$ fully dynamic DFS algorithms (refer to Table 1 for comparison).

Recently, Chen *et al.* [18] designed an $\mathcal{O}(n)$ time incremental algorithm for DFS tree in undirected graphs, which is optimal if it is required to output the DFS tree after each update. For the hardness results for the dynamic *ordered* DFS problem, the reader may refer to the work of Reif [36, 37] and Miltersen *et al.* [33].

1.2 Familiarizing with the Fault Tolerant DFS Problem

For an undirected graph, DFS traversal results in a spanning tree rooted at the vertex from where the DFS begins. The depth first nature of the traversal ensures the following property:

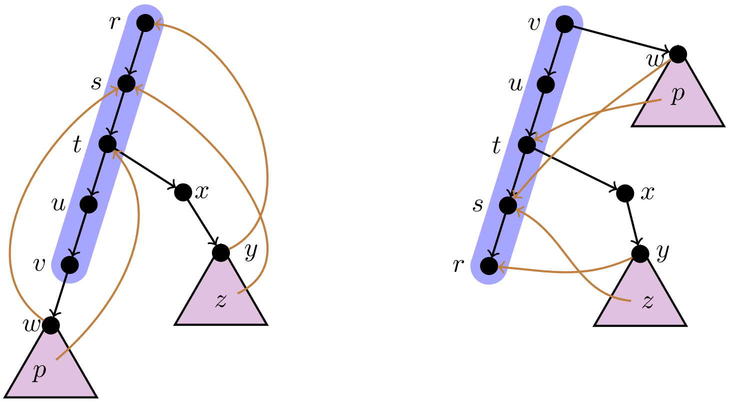
▶ **Property 1.** *(DFS Property) For each vertex $v \in V$, every neighbour of v in the graph appears either as an ancestor of v or as a descendant of v in any DFS tree.*

The DFS property implies that a non-tree edge is never a *cross edge* - an edge whose endpoints do not share an ancestor-descendant relationship. It is due to this reason that each non-tree edge is called a back edge. We now define ancestor-descendant paths.

▶ **Definition 2.** *(Ancestor-descendant Path) A path in a DFS tree is called an ancestor-descendant path if its endpoints have an ancestor-descendant relationship in the tree.*

In order to familiarize with the problem of fault tolerant DFS tree, we discuss another related, but simpler problem, namely, rerooting of a DFS tree defined as follows.

[1] Precisely, their data structure occupies $(m + o(m)) \log n$ bits by using a wavelet tree.



(a) a DFS tree rooted at r.

(b) reversing $path(r, v)$ results in transforming many back edges to cross edges.

Figure 1 Non-triviality of rerooting problem.

▶ **Problem 3.** *Preprocess an undirected graph $G = (V, E)$ to build a compact data structure so that given any vertex $v \in V$, we can report the DFS tree rooted at v efficiently.*

Let T be an initial DFS tree, rooted at a vertex, say r (see Figure 1(a)). We use $T(q)$ to represent the subtree of T rooted at the vertex q. Also, let $path(a, b)$ denote the path from vertex a to b in T. For computing a DFS tree rooted at any vertex v, the first natural idea is to just reverse the direction of the path from r to v in T. However, this may result in transforming many back edges to cross edges and hence a violation of the DFS property (see Figure 1(b)). To fix this problem, we may need to reroot various subtrees hanging from the reversed path. Along these lines, [3] presented an algorithm that takes $\mathcal{O}\left(n \log^3 n\right)$ time to compute a DFS tree rooted at any vertex.

In order to see how rerooting a DFS tree is related to the problem of fault tolerant DFS tree, consider the failure of vertex u in Figure 1(a). The subtree $T(v)$ is connected to the remaining tree through many back edges, and the back edge (p, t) is incident closest to u on the $path(r, u)$. If we reroot the subtree $T(v)$ at vertex p and hang it from the remaining tree through the edge (p, t), the resulting tree will indeed be a valid DFS tree of $G \backslash \{u\}$.

The fault tolerant DFS tree problem becomes more complex in the presence of multiple faults. However, the distribution of faults plays an important role as follows. If the failing vertices do not have any ancestor-descendant relationship in the DFS tree, they can be handled *independently*. For example, the simultaneous failure of vertices u and x in the DFS tree shown in Figure 1(a) requires rerooting the respective subtrees $T(v)$ and $T(y)$ at vertices p and z respectively. So, even for arbitrarily large number of faults, if no two of them appear on the same root to leaf path, we have to just reroot the corresponding disjoint subtrees of T to report the DFS tree avoiding those failures. However, if two or more faults indeed appear on a single root to leaf path, the problem becomes more complex. For this case, [3] presents an algorithm which is quite different from their rerooting algorithm.

1.3 Overview of the Previous Results

We now begin with an overview of the existing algorithms for fault tolerant DFS tree. For any set of k failures, the algorithm presented in [3] first partitions the original DFS tree into a pool of connected components. This pool consists of k paths (specifically, *ancestor-descendant paths*) and potentially a large number of subtrees. The algorithm treats each of these components as a *super vertex* and uses them to grow the DFS tree T^* that avoids all the failures. At a high level, the algorithm can be visualized as a traversal on these super vertices. Each traversal extracts a path from the super vertex, attaches it to T^*, and places the remaining portion of the super vertex back into the pool. In order to pursue DFS traversal further in an efficient manner, the algorithm needs to compute *minimal* adjacency lists for the vertices of the traversed path (referred to as *reduced adjacency lists*). The algorithm makes use of the following crucial property of DFS traversal.

▶ **Property 4.** *(Components Property [3]) Consider any DFS Traversal on any undirected graph $G = (V, E)$. When the traversal reaches a vertex $v \in V$, let the set of connected components induced by the unvisited vertices be C. If from any component $c \in C$, there exists two edges - e to vertex v and e' to any of the visited vertices (including v), then for building a valid DFS tree, it is sufficient to consider only the edge e during the rest of DFS traversal, that is, e' can be ignored.*

In order to use the above property to populate the reduced adjacency list, the algorithm needs a data structure to answer the following queries repeatedly.

- $Query(w, x, y)$: among all the edges from w that are incident on the $path(x, y)$ in T, return an edge that is incident nearest to x on the $path(x, y)$.
- $Query(T(w), x, y)$: among all the edges from $T(w)$ that are incident on the $path(x, y)$ in T, return an edge that is incident nearest to x on the $path(x, y)$.

It is quite obvious from the description given above that these queries are quite non-trivial, and so a sophisticated data structure is designed in [3] to answer these queries efficiently. In addition to the complex data structure, the complete difference in the processing of a path and a subtree obfuscates the algorithm and its analysis.

The subsequent results [17, 34] keep the algorithm unchanged and replace the data structure used in [3] with alternate data structures. Chen *et al.* [17] model the two queries mentioned above as Orthogonal Range Successor/Predecessor(ORS/ORP) queries and this improves the query processing time. Nakamura and Sadakane [34] compressed the data structure used in [17] using Wavelet Trees [27] to achieve optimal space. Despite these improvements, the core of the fault tolerant algorithm remains intricate and the data structure still remains complex. Recently, in an empirical study [5], it was found that this algorithm, for incremental updates, performs even worse than the static DFS algorithm for certain classes of graphs. This naturally raises the question whethere there exists a simpler algorithm for this fundamental problem.

In addition to being complex, all these algorithms fail to exploit the distribution of the faults to achieve efficiency. The running time of these algorithm is $\mathcal{O}(nk\ polylog(n))$ time irrespective of how the k faults are distributed in the DFS tree. In the extreme case, when no two faults share the ancestor-descendant relaitonship, there is a simple $\mathcal{O}(n\ polylog(n))$ time algorithm as described in Section 1.2. This raises the question whether it is possible to have a fault tolerant DFS algorithm whose time complexity depends upon the maximum number of faults lying on any ancestor-descendant path instead of the total number of faults.

1.4 Our Contribution

We take a much simpler approach as compared to the previous algorithms. We first present a new and simple rerooting algorithm based on the following ideas. After building an initial DFS tree, say T, we decompose T into a disjoint collection $\mathcal{P}$ of ancestor-descendant paths. Similar to [3], each of these paths are treated like *super vertices.* At a high level, the algorithm can still be viewed as a traversal on these super vertices. However, as the reader may also verify, the algorithm turns out to be lighter and quite different at the core. Interestingly, the original DFS tree alone acts as a powerful data structure to be used for rerooting or for the computation of another valid DFS tree in the presence of faults. The algorithm crucially exploits an implicit hierarchy among the ancestor-descendant paths in $\mathcal{P}$. This hierarchy along with the DFS property of T enables us to use much simpler queries. In particular, each query will ask only for an edge from a vertex to one of its ancestor paths in the hierarchy. The hierarchy allows us to represent T as another tree structure, called *shallow tree*. In a nutshell, our algorithm can be viewed as an efficient DFS traversal guided by this shallow tree.

This rerooting algorithm extends to the fault tolerant algorithm with very little and obvious modifications. While preserving simplicity, the fault tolerant algorithm turns out to be faster than all the previous algorithms. Moreover, our algorithm is the first to implicitly incorporate the distribution of faults to gain efficiency. We summarize our result in the following theorem.

▶ **Theorem 5.** *An undirected graph G can be preprocessed in $\mathcal{O}(m+n)$ time to build a DFS tree, say T, and a data structure of $\mathcal{O}(m+n)$ words*[2] *such that for any set F of k failed vertices or edges, a DFS tree of $G \setminus F$ can be reported in $\mathcal{O}(n(k'+\log n)\log n)$ time, where $k' \leq k$ is the maximum number of faults on any root-leaf path in the tree T.*

We now present the highlights of our algorithm.
Drastically simpler algorithm: Our algorithm is drastically simpler and more intuitive than the previous algorithm. We feel confident to defend that it can be taught even in an undergraduate course on algorithms. The pseudo-codes in Algorithm 1 and Algorithm 2 are concise and very close to the corresponding implementations.

Faster time complexity: Our algorithm takes $\mathcal{O}(n(k'+\log n)\log n)$ time, where k' is the maximum number of failures on any ancestor-descendant path of the DFS tree when k edges/vertices fail. In the worst-case k' can be as large as k. However, k' can be $o(k)$ as well. In the latter case, our result improves all the existing results significantly. Moreover, even in the case $k'=k$, our time complexity is superior to the previous best by a log factor.

Optimal preprocessing time: Our preprocessing relies upon DFS traversal only, taking $\mathcal{O}(m+n)$ time. Given a graph, in order to report the initial DFS tree, one anyway has to run a static DFS. Hence, our preprocessing time is optimal.

Optimal space and elementary data structure: In contrast to the heavy data structure used by [3, 17], our algorithm makes use of very elementary data structures which are compact as well. Each vertex keeps an array storing edges incident on it from ancestors sorted according to their levels. This data structure uses just $m+n$ words and still achieves $\mathcal{O}\left(n(k'+\log n)\log^2 n\right)$ time to report a DFS tree upon failure of any k vertices or edges. By using fractional cascading [14], we get rid of one log factor while still keeping space requirement to be $\mathcal{O}(m+n)$ words.

Faster Fully Dynamic Algorithm: Using Theorem 5 and periodic rebuilding technique used in [3], we also get the fastest algorithm for fully dynamic DFS.

▶ **Theorem 6.** *Given an undirected graph, one can maintain a DFS tree for any online sequence of insertions and deletions of vertices/edges in $\mathcal{O}\left(\sqrt{mn\log n}\right)$ worst-case time per update.*

[2] One word stores $\lceil \log m \rceil$ bits.

Table 1 Comparison of the existing and the new results. Note that k is the total number of faults, whereas k' is the maximum number of faults lying on any root-to-leaf path of the DFS tree.

	[3]	[17]	[34]	**New**
Space (in bits)	$\mathcal{O}(m\log^2 n)$	$\mathcal{O}(m\log^2 n)$	$\mathcal{O}(m\log n)$	$\mathcal{O}(m\log n)$
Preprocessing	$\mathcal{O}(m\log n)$	$\mathcal{O}(m\log n)$	$\mathcal{O}\left(m\sqrt{\log n}\right)$	$\mathcal{O}\left(m+n\right)$
Fault tolerant	$\mathcal{O}(nk\log^4 n)$	$\mathcal{O}(nk\log^2 n)$	$\mathcal{O}(nk\frac{\log^3 n}{\log\log n})$	$\mathcal{O}\left(n(k'+\log n)\log n\right)$
Dynamic DFS	$\mathcal{O}(\sqrt{mn}\log^{2.5} n)$	$\mathcal{O}(\sqrt{mn}\log^{1.5} n)$	$\mathcal{O}(\sqrt{mn}\frac{\log^{1.75} n}{\sqrt{\log\log n}})$	$\mathcal{O}(\sqrt{mn\log n})$

The new fully dynamic algorithm can be used to solve the dynamic subgraph problems discussed in [3] and improves upon their time complexity as well. Due to space constraint, we do not discuss these problems here. These can be accessed in the full version of the paper.

Table 1 offers a comparison of our results with all the previous results.

1.5 Organisation of the Paper

Section 2 introduces the notations and some well-known techniques/properties used throughout the paper. Section 3 defines the *shallow tree* representation, a concise structure which encapsulates the hierarchy of paths in the initial DFS tree. Section 4 is the core of our work. Here, we describe how a DFS tree can be rerooted efficiently. Section 5 describes how with some minor modifications to the data structure, rerooting procedure extends to a fault tolerant algorithm. We present the fully dynamic algorithm in Section 6.

2 Preliminaries

2.1 Notations

Following notations will be used throughout this paper.

- T: Any DFS tree of the original graph G.
- $dfn\,(x)$: The depth first number, i.e., the discovery time of the vertex x during the DFS traversal.
- $v\,(i)$: the vertex $x \in V$ such that $dfn(x) = i$.
- $dist\,(x, y)$: distance between the vertices x and y in the DFS tree T.

For the sake of ease of explanation, we shall assume that the graph remains connected at all times. This assumption is without loss of generality because of the following standard way of transforming the original graph right in the beginning - Introduce a dummy vertex r and connect it to all vertices of the graph. Henceforth, we maintain a DFS tree rooted at r for this augmented graph. It is easy to observe that the augmented graph remains connected throughout and the DFS tree rooted at r will be such that the subtrees rooted at the children of r constitute a DFS forest of the original graph.

2.2 Heavy-Light Decomposition

Sleator and Tarjan, in their seminal result on dynamic trees [42] introduced a technique of partitioning any rooted tree called Heavy-light decomposition. Given any rooted tree, this technique splits it into a set of vertex-disjoint ancestor-descendant paths. It marks all the tree edges either dashed or solid - a tree edge is marked solid iff the subtree of the child vertex is heaviest (in terms of number of vertices) among the subtrees of all its siblings and dashed otherwise. A maximal sequence of vertices connected through solid edges constitutes the required ancestor-descendant path. This decomposition can be carried out using DFS traversal in $\mathcal{O}(n)$ time.

2.3 Fractional Cascading

Given n sorted arrays and a value x, suppose we need to find the predecessor/successor of x in each of them. A naive way is to make a binary search on each array. Chazelle and Guibas [14] introduced a novel tool called fractional cascading using which this problem can be solved more efficiently. Also, Chen *et al.*[18] used this tool for arriving at an $\mathcal{O}(n)$ algorithm for incremental updates. We adapt a customized version of their method.

▶ **Lemma 7.** *Fractional cascading: Given n sorted arrays $\{A_i\}_{i\in[n]}$ each with $l_i = |A_i|$ elements and total $\sum_i^n l_i = m$ elements. There exists a data structure of $\mathcal{O}(m+n)$ words, which can be built in $\mathcal{O}(m+n)$ time, such that for any given x, i, and k satisfying $i, k \in [n]$ and $i + k \leq n$, we can search for x (or its predecessor/successor) in all arrays $A_i, \ldots, A_{i+k}$ using the data structure in $\mathcal{O}(k + \log m)$ time.*

3 Shallow Tree Representation

We now introduce the shallow tree representation for DFS tree T that plays a key role in our algorithm. Using heavy-light decomposition, T is broken down into a set of vertex-disjoint ancestor-descendant paths. Let's denote this set with $\mathcal{P}$. Observe that these paths are connected through dashed edges in T. These dashed edges introduce a hierarchy among paths in $\mathcal{P}$ and the shallow tree defined below captures this hierarchy.

▶ **Definition 8.** *Given a DFS tree T of an undirected graph G, let $\mathcal{P}$ be the set of paths obtained through heavy-light decomposition of T. Let H be the set of edges marked dashed during the decomposition. For tree T, its shallow tree S is a rooted tree formed by collapsing each element of $\mathcal{P}$ into a single node (super vertex). Note that, for each edge $(y, z) \in H$ with $y = parent(z)$, the node in S that contains y is the parent of the node containing z.*

Figure 2 demonstrates how a DFS tree is decomposed to form a set of ancestor-descendant paths $\mathcal{P}$ which is subsequently used to form the shallow tree S.

To avoid ambiguity, we address the vertices in the shallow tree as "nodes" and the vertices in the DFS tree as "vertices". $node(x)$ denotes the shallow tree node corresponding to the path in $\mathcal{P}$ containing vertex x.

The construction of S described above ensures the following simple but crucial properties.

- As a result of heavy-light decomposition of a tree T with n vertices, there can be at most $\log n$ dashed edges on any root to leaf path in T. Recall that each edge in S corresponds to a dashed edge. Thus, the depth of any node in tree S can't be larger than $\log n$. It is because of this small depth that we choose the name *shallow tree* for S.
- From the DFS property, neighbours of any vertex $v \in V$ in the graph are either ancestors or descendants of v in T. Consider any such neighbour u. Let p_1 be the path in $\mathcal{P}$ containing v and p_2 be the path in $\mathcal{P}$ containing u. u and v may also lie in the same path in $\mathcal{P}$. From the construction of S, the nodes corresponding to p_1 and p_2 will share an ancestor-descendant relation in S. So we can state the following lemma.

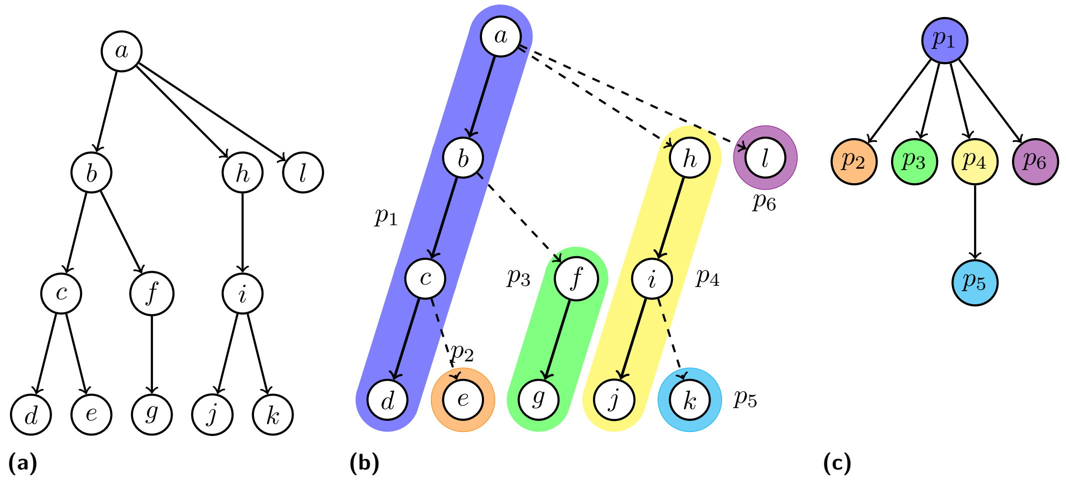


Figure 2 (a) A DFS tree T. (b) Heavy light decomposition of T and the resulting paths in $\mathcal{P}$. (c) The corresponding shallow tree.

▶ **Lemma 9.** *For a DFS tree T of an undirected graph $G = (V, E)$ with shallow tree S, any vertex $v \in V$ which lies in node $\mu \in S$ can have edges only to vertices lying in the nodes which are ancestors or descendants of μ in S.*

We require that the vertices of each solid path have consecutive dfn. This enables us to represent each path $p \in \mathcal{P}$ in a compact manner using just the smallest and largest dfn of vertices on p. For each path p, we store this pair as *PathEndPoints* at the corresponding node of the shallow tree S. Assigning consecutive dfn to each solid path can be accomplished easily - we carry out another DFS on T where for each vertex, the next vertex to be visited is its child hanging through a solid edge. Thus, the processing of T to make the shallow tree S requires only DFS on T and takes $\mathcal{O}(n)$ time.

4 Rerooting DFS Tree T

Given a DFS tree T for a graph $G = (V, E)$ and a vertex $r' \in V$, the objective is to compute a DFS tree T^* rooted at r' for the same graph G. First we compute the shallow tree representation S of T. We now describe the rerooting procedure.

4.1 Reroot Procedure

The tree T^* is empty in the beginning and is grown gradually starting from r'. To build this tree efficiently, we use the following two ideas. The first idea is to re-use the paths from $\mathcal{P}$. Observe that while rerooting, if we use the original adjacency list for scanning the neighbours, we need $\mathcal{O}(m + n)$ time. So the second idea is to populate for each vertex only a subset of its adjacency list which is small and yet sufficient to compute a DFS tree (this idea was used by [3] in the fault tolerant DFS problem). These lists we refer as *reduced adjacency lists*. We take a lazy and frugal approach to populate them.

The intuition sketched above is materialized by carrying out a DFS traversal *guided* by the shallow tree S. Note that a node of the shallow tree corresponds to a path in $\mathcal{P}$. To compute T^*, our algorithm performs a sequence of steps. Each step begins with entering a node of S through some vertex present on the path stored at the node and leaving it after traversing the path along one direction. The first node to be visited is $node(r')$. We now provide complete details of the computation involved in each step. Consider any node $\nu \in S$. Let $path(y,z)$ be the path corresponding to ν. When the DFS traversal enters ν through a vertex, say x, the following 3 simple operations are carried out.

1. *Move towards the farther end of the path.*
   We determine the vertex from $\{y,z\}$ farther from x. Let this vertex be y. DFS traversal proceeds from x to y and $path(x,y)$ is attached to the tree T^*. Next, we update the $PathEndpoints\,(\nu)$ such that it stores the endpoints of the untraversed part of $path(y,z)$. This choice of direction ensures that at least half of the path is traversed (referred to as path halving technique, also used in [2, 3]).
2. *Populate the reduced adjacency list of the path just traversed.*
   For vertices on $path(x,y)$, we populate the reduced adjacency list $\mathcal{L}$ using ancestors and descendants of ν in the shallow tree S. This will be explained below in Section 4.1.1.
3. *Continue traversal.*
   Using the reduced adjacency list $\mathcal{L}$ of the traversed path computed in step 2 above, we continue the DFS traversal along the unvisited neighbours of the vertices in the order from y to x (opposite to the direction of traversal, due to the recursive nature of DFS).

Algorithm 1 presents the complete pseudocode of the rerooting procedure based on the above 3 steps. All vertices are marked *unvisited* initially and the reduced adjacency list $\mathcal{L}$ is empty. Invoking Reroot(r') produces the DFS tree rooted at r'.

Algorithm 1 Recursive procedure to reroot the DFS Tree T.

```
Function Reroot (x)
    $(y,z) \leftarrow PathEndpoints\,(node\,(x))$ ;
    if $dist(x,z) > dist(x,y)$ then Swap $(y,z)$;      /* compute $dist$ using $dfns$ */
    Attach $path\,(x,y)$ to $T^*$;
    if $x \neq z$ then
        $w \leftarrow$ Neighbour of $x$ on $path(y,z)$ nearest to $z$ ;
        $PathEndpoints\,(node\,(x)) \leftarrow (w,z)$ ;               /* untraversed path */
    end
    $\mathcal{L} \leftarrow ReducedAL\,(\mathcal{L},(x,y))$ ;      /* Update $\mathcal{L}$ for vertices on $path(x,y)$ */
    for $i = dfn(y)$ to $dfn(x)$ do  $status(v(i)) \leftarrow visited$ ;
    for $i = dfn(y)$ to $dfn(x)$ do
        foreach $vertex\ u \in \mathcal{L}(v(i))$ do
            if  $status(u) = unvisited$ then  {add $(v\,(i)\,,u)$ to $T^*$; Reroot$(u)$} ;
        end
    end
```

4.1.1 Populating Reduced Adjacency Lists

Here we define a query $Q\,(u,(p_s,p_e))$, where $u,p_s,p_e \in V$ satisfy the following constraints:

- p_s and p_e are the endpoints of an ancestor-descendant path in T.
- u is a descendant of the highest vertex on $path\,(p_s,p_e)$, but does not lie on the path.

This query finds an edge from the vertex u that is incident to $path(p_s,p_e)$ closest to p_e. It returns *null* if no such edge exists, otherwise it returns the endpoint other than u.

Consider any node $\nu \in S$, and let $path(y,z)$ be its corresponding path in $\mathcal{P}$. When the DFS traversal enters ν through x and proceeds towards y, we populate the reduced adjacency lists of vertices on $path(x,y)$ using Lemma 9 as follows.

- *Processing Ancestors of vertices on path(x, y).*
  For each $u \in path(x, y)$ and for each ancestor μ of ν, we add $Q\left(u, PathEndpoints\left(\mu\right)\right)$ to $\mathcal{L}\left(u\right)$.
- *Processing Descendants of vertices on path(x, y).*
  Using the shallow tree S, one can list all the vertices that are descendants of vertices on $path(x, y)$ in T. From all these vertices, we query for an edge to $path(x, y)$ which is incident closest to y. For any descendant u, if the query $Q\left(u, (x, y)\right)$ returns a non-*null* value, say w, then we add u to $\mathcal{L}\left(w\right)$.

▶ Remark 10. In Algorithm 1, after visiting x, if we moved towards the leaf node, then the untraversed part of ν has to be treated as an ancestor path while populating the reduced adjacency lists of $path\left(x, y\right)$, and as a descendant path, otherwise.

Algorithm 2 Populating reduced adjacency lists of vertices on $path(x, y)$.

```
Function ReducedAL ($\mathcal{L}, (x, y)$)
    $\mu \leftarrow$ parent (node ($x$));
    while $\mu \neq NULL$ do                                   /* edges to ancestor paths */
        for $i = dfn(y)$ to $dfn(x)$ do
            $\mathcal{L}(v(i)) \leftarrow \mathcal{L}(v(i)) \cup \{Q(v(i), PathEndPoints(\mu))\}$;
        end
        $\mu \leftarrow parent(\mu)$ ;
    end
    if $dfn(x) < dfn(y)$ then $z \leftarrow x$ else $z \leftarrow y$;   /* $z$ is ancestor among two */
    $\mathcal{C} \leftarrow \mathrm{Desc}_T(z) \setminus \{v(dfn(x)), \ldots, v(dfn(y))\}$;
    foreach $u \in \mathcal{C}$ do                             /* each descendant of $path(x, y)$ */
        $w \leftarrow Q(u, (x, y))$ ;                           /* ensure $w$ is closest to $y$ */
        if $w \neq NULL$ then $\mathcal{L}(w) \leftarrow \mathcal{L}(w) \cup \{u\}$ ;
    end
    return $\mathcal{L}$
```

In Algorithm 2, we used query $Q\left(u, (p_s, p_e)\right)$ as a black box. This query can be answered very easily as follows. The constraints of $Q\left(u, (p_s, p_e)\right)$ imply that the returned edge is always an edge from u to an ancestor of u. For answering this query, we use a data structure $\mathcal{D}$ which stores the following information for each vertex u.

▶ **Definition 11.** *$\mathcal{D}\left(u\right)$ is an array that stores each ancestor of u in T to which u is a neighbour and it stores them in the increasing order of distance from the root of T.*

This data structure enables us to answer query $Q\left(u, (p_s, p_e)\right)$ using a binary search on array $\mathcal{D}\left(u\right)$ and it takes $\mathcal{O}\left(\log n\right)$ time only. Interestingly, the data structure $\mathcal{D}$ can be preprocessed very easily in $\mathcal{O}\left(m + n\right)$ time as follows. We visit vertices in the increasing order of their depth first numbers. Note that for each vertex v, the neighbours of v which have dfn larger than that of v are its descendants. For each such descendant u, we append v to $\mathcal{D}(u)$. Iterating in increasing order of dfn ensures that all arrays in $\mathcal{D}$ are sorted as needed.

▶ **Theorem 12.** *Given a DFS Tree T of an undirected graph G, it takes $\mathcal{O}\left(m + n\right)$ time to build a data structure consisting of exactly $(m + n)$ words which can answer the query $Q\left(u, (p_s, p_e)\right)$ in $\mathcal{O}\left(\log n\right)$ time.*

4.2 Time Complexity Analysis

During preprocessing, we construct the shallow tree S for the DFS tree T and build $\mathcal{D}$. This processing as shown earlier can be completed in $\mathcal{O}(m+n)$ time. The time complexity of the Reroot procedure is bounded by the time required to populate the reduced adjacency lists $\mathcal{L}$. This in turn, is bounded by the number of calls to query Q. To analyse the number of calls made from any vertex w, let ν be the node in the shallow tree S, containing w. In general, if the bound on height of S is d, ReducedAL makes worst-case d queries from w to ancestors of ν. Also when an ancestor of ν in S, say μ, is visited during Reroot procedure, ReducedAL makes a query from w to μ. The path halving technique (line 3 in Algorithm 1) ensures that any node in S (or a path in $\mathcal{P}$) is visited at most $\log n$ times. This implies that any such ancestor μ may be visited at most $\log n$ times. Thus, we can have worst-case $d(\log n+1)$ queries from w to its ancestor paths throughout the Reroot procedure. Summing over all the vertices, there can be at most $nd(\log n+1)$ calls to query Q. Therefore, populating the reduced adjacency lists $\mathcal{L}$ takes overall $\mathcal{O}\left(nd\log^2 n\right)$ time. From Section 3, we know $d \leq \log n$. Thus, using Theorem 12, we can state the following lemma.

▶ **Lemma 13.** *Given a DFS Tree T of an undirected graph G, it takes $\mathcal{O}(m+n)$ time to build a data structure of $(m+n)$ words using which we can compute a DFS Tree of G rooted at any given vertex in $\mathcal{O}\left(n\log^3 n\right)$ time.*

4.2.1 Getting rid of a log factor

Consider the moment when the Reroot procedure enters a $path(y,z)$ through the vertex x and reaches the endpoint y. In Algorithm 2, for each descendant w of $path(x,y)$, we perform query $Q(w,(x,y))$ separately. Instead, using Fractional Cascading, we can perform all these queries together in an efficient manner. Among x and y, let x be the vertex closer to the root of T. As described earlier, all the vertices of $path(x,y)$ have consecutive dfn, and so do the vertices of subtree $T(x)$. Let $last(x)$ be the vertex in $T(x)$ with the largest dfn. Since vertices on $path(x,y)$ have already been visited, we need to query for edges only from vertices with dfn between $dfn(y)+1$ and $dfn(last(x))$. Finding edges to $path(x,y)$ from these vertices can be done with a single query to the fractionally cascaded $\mathcal{D}$ (Lemma 7). It takes $\mathcal{O}(\log m + dfn(last(x)) - dfn(y))$ time to execute this query. We charge the $\log m$ part of the query time to the vertices on $path(x,y)$ and the $dfn(last(x)) - dfn(y)$ is distributed among the descendant vertices. Thus, each descendant vertex incurs a constant charge.

Note that, queries to the ancestors of $path(x,y)$ are answered using the original $\mathcal{D}$ itself. Therefore, in a shallow tree of height d, each vertex $v \in V$ incurs following charges during Reroot procedure - $\mathcal{O}(d\log n)$ when v acts as a descendant in the queries made while visiting ancestors of v, and $\mathcal{O}(d\log n+\log m)$ while visiting v itself. Overall the charge on any vertex is $\mathcal{O}(d\log n)$ and, therefore, the time complexity of Reroot procedure reduces to $\mathcal{O}(nd\log n)$. So we can state the following lemma.

▶ **Lemma 14.** *Given an undirected graph G, the DFS tree T and corresponding shallow tree of height d, it takes $\mathcal{O}(m+n)$ time to build a data structure occupying $\mathcal{O}(m+n)$ words using which the Reroot algorithm executes in $\mathcal{O}(nd\log n)$ time.*

4.3 Correctness of Reroot Procedure

As is evident from the pseudocode of Algorithm 1, besides populating the reduced adjacency lists, Reroot imitates a usual DFS traversal. So in order to show that Reroot computes a valid DFS tree, we need to prove the following. Each edge e that is not added to reduced adjacency list is indeed redundant and will appear as a back edge in the resulting DFS tree. We can show this as follows.

During Reroot procedure, consider the moment when we attach some path, say $path(x, y)$ to T^*. Lemma 9 implies that all the neighbours of vertices on the $path(x, y)$ will lie either in ancestors or in descendants of $path(x, y)$ in the shallow tree. For each descendant vertex v of $path(x, y)$, we add an edge from v incident on $path(x, y)$ closest to y (lines 12,13 of Algorithm 2). For each ancestor path of $path(x, y)$, say μ, we add an edge from each vertex of $path(x, y)$ to μ (line 5 of Algorithm 2). These ensure that at the moment $path\,(x, y)$ is visited and attached to T^*, from each connected component in the graph induced by unvisited vertices, the edge incident on $path(x, y)$ closest to vertex y is added to $\mathcal{L}$ (if exists). Using Components Property, it follows that all those edges incident on $path(x, y)$, that are not added to $\mathcal{L}$, will appear as back edges in the resulting DFS tree. Hence, the traversal performed using the reduced adjacency lists indeed results in a valid DFS tree.

5 Extension to Fault Tolerant DFS Tree

Given an undirected graph G, we build a DFS tree of G, say T, the corresponding shallow tree S, and the fractionally cascaded D. Let F be the set of failures (edges/vertices) in G, with $|F| = k$. Here, we describe how with some elementary modifications to the shallow tree of T, procedure Reroot can be utilized to report a DFS tree of $G \setminus F$. For the given set F, we update the set $\mathcal{P}$ and the shallow tree S as follows.

1. Each vertex maintains a state: *active* or *failed*. For a failed vertex $x \in F$, we toggle x's state to *failed*. Let $p \in \mathcal{P}$ be the path containing x. We remove x from p. The resulting smaller paths are added to $\mathcal{P}$ and p is removed from $\mathcal{P}$.
2. For each failed edge $e = (u, v)$, we mark the corresponding entries in the adjacency lists of u and v as *failed*. Here, we do not make any changes to fractionally cascaded $\mathcal{D}$. If failed edge e is a tree-edge and was marked *solid* during heavy-light decomposition, $\mathcal{P}$ is updated as follows. Path $p \in \mathcal{P}$ containing e is split into two smaller paths. These smaller paths are added to $\mathcal{P}$ and p is removed from $\mathcal{P}$.
3. After updating $\mathcal{P}$, shallow tree S is updated as follows. For any path $p \in \mathcal{P}$, let the vertex in p closest to root of T be x. The parent of node corresponding to p in S will be the node containing the nearest active ancestor of x.

Given the set F, we can update $\mathcal{P}$ and S in $\mathcal{O}\,(n)$ time as follows - Make a DFS traversal through T while ensuring vertices are visited in increasing order of their dfn. Update $\mathcal{P}$ and S as discussed above. For each failure, the number of nodes in S increases by at most one. After k updates, S can have at most k new nodes and may not be shallow anymore. However, the depth of a node ν in S doesn't increase due to any failure which does not lie on the path from ν to root of S. Thus, if the maximum number of failures on any root-leaf path in T is k', the height of the shallow tree will be at most $k' + \log n$.

To ensure that no *deleted* vertices/edges are traversed during Reroot procedure, we make the following modifications. Since no changes are made to fractionally cascaded $\mathcal{D}$, the result of some query $Q\,(v, (p_s, p_e))$ may be an endpoint of a *failed* edge. In such a case, we iterate in $\mathcal{D}\,(v)$ starting from the *failed* edge towards p_s until we find an edge present in $G \setminus F$. However, if we cross p_s in doing so, it implies that there is no edge between vertex v and $path\,(p_s, p_e)$, we stop and return *null*. During the procedure Reroot, we spend overall $\mathcal{O}\,(k)$ time in such iterations. Also note that each node of the updated S corresponds to a path consisting of active vertices only. So $Q\,(v, (p_s, p_e))$ never returns a failed vertex. It follows from the changes in the query and the shallow tree described above that Reroot procedure reports a valid DFS tree of $G \setminus F$. Using Lemma 14, we can conclude the following theorem.

▶ **Theorem 15.** *An undirected graph G can be preprocessed in $\mathcal{O}(m+n)$ time to build a DFS tree, say T, and a data structure of $\mathcal{O}(m+n)$ words such that for any set F of k failed vertices or edges, a DFS tree of $G \setminus F$ can be reported in $\mathcal{O}(n(k'+\log n)\log n)$ time, where $k' \leq k$ is the maximum number of faults on any root-leaf path in the tree T.*

6 Fully Dynamic DFS

We first describe how the fault tolerant DFS algorithm can handle incremental updates. Following that, we use the *overlapped periodic rebuilding technique* to arrive at a fully dynamic DFS algorithm. The ideas utilized in both of these steps were used by Baswana *et al.* [3].

Let U be the set of updates in any undirected graph G. For the edge insertions, we directly add them to the reduced adjacency lists of the endpoints of the edges. To handle insertion of a vertex, we add the new vertices in V and treat their edges as edge insertions. Observe that these modifications are sufficient to handle incremental updates. The size of the reduced adjacency lists after $|U|$ updates is at most $n(|U|+\log n)(\log n+1)$ (from Section 4.2). So the worst-case time complexity of our algorithm is $\mathcal{O}(n(|U|+\log n)\log n)$.

We can use the Reroot procedure to report a DFS tree after every update. But as $|U|$ increases, the algorithm slows down. Therefore, we need to rebuild the data structures (denoted collectively using $\mathcal{A}$) periodically to maintain the efficiency. We rebuild them after, let's say, every c updates. It takes $\mathcal{O}(m+n)$ time to rebuild the data structures. The cost of rebuilding is amortized among the c updates. This results in an amortized runtime fully dynamic algorithm.

To get a worst-case bound, the rebuilding is done in an overlapped fashion as follows. The original data structures $\mathcal{A}_0$ are used till first $2c$ updates. At the end of kc updates (for $k \geq 1$), we start building $\mathcal{A}_k$, the data structures for the graph after incorporating the first kc updates. The computation of building $\mathcal{A}_k$ is distributed evenly over the next c updates to the graph (from $(kc+1)$ to $(k+1)c$ updates). However, during these updates we use $\mathcal{A}_{k-1}$ to report the updated DFS Tree. This ensures $|U|$ is never larger than $2c$. After $(k+1)c$ updates, we have $\mathcal{A}_k$ ready for use and we can discard $\mathcal{A}_{k-1}$. This helps us manage our data structures in $\mathcal{O}(m+n)$ space. For this overlapped periodic rebuilding framework, the following lemma from [3] provides the worst-case bound on the update time we can derive.

▶ **Lemma 16.** *(Overlapped Periodic Rebuilding: Lemma 6.1 in [3]) Let $\mathcal{D}$ be a data structure that can be used to report the solution of a graph problem after a set of U updates on an input graph G. If $\mathcal{D}$ can be built in $\mathcal{O}(f)$ time and the solution for graph $G+U$ can be reported in $\mathcal{O}(h+|U|\cdot g)$time, then $\mathcal{D}$ can be used to report the solution after every update in worst-case $\mathcal{O}\left(\sqrt{fg}+h\right)$ update time, given that $f/g \leq n$.*

Substituting $f=m$, $g=n\log n$ and $h=n\log^2 n$, we obtain, $\mathcal{O}\left(\sqrt{fg}+h\right)=\mathcal{O}\left(\sqrt{mn}\log n\right)$. Hence we conclude with the following theorem.

▶ **Theorem 17.** *An undirected graph can be preprocessed in $\mathcal{O}(m+n)$ time to build a data structure occupying $\mathcal{O}(m+n)$ words, using which one can maintain a DFS tree for any online sequence of insertions and deletions of vertices/edges in $\mathcal{O}\left(\sqrt{mn}\log n\right)$ worst-case time per update.*

References

1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On Fully Dynamic Graph Sparsifiers. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 335–344, 2016. doi:10.1109/FOCS.2016.44.

2 Alok Aggarwal, Richard J. Anderson, and Ming-Yang Kao. Parallel Depth-First Search in General Directed Graphs. *SIAM J. Comput.*, 19(2):397–409, 1990. doi:10.1137/0219025.

3 Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS in Undirected Graphs: breaking the O(m) barrier. In *Symposium on Discrete Algorithms, SODA*, pages 730–739, 2016. doi:10.1137/1.9781611974331.ch52.

4 Surender Baswana and Keerti Choudhary. On Dynamic DFS Tree in Directed Graphs. In *MFCS, Proceedings, Part II*, pages 102–114, 2015. doi:10.1007/978-3-662-48054-0_9.

5 Surender Baswana, Ayush Goel, and Shahbaz Khan. Incremental DFS algorithms: a theoretical and experimental study. In *Symposium on Discrete Algorithms, SODA*, pages 53–72, 2018. doi:10.1137/1.9781611975031.4.

6 Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully Dynamic Maximal Matching in O(log n) Update Time (Corrected Version). *SIAM J. Comput.*, 47(3):617–650, 2018. doi:10.1137/16M1106158.

7 Surender Baswana and Shahbaz Khan. Incremental Algorithm for Maintaining DFS Tree for Undirected Graphs. In *ICALP, Proceedings, Part I*, pages 138–149, 2014. doi:10.1007/978-3-662-43948-7_12.

8 Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Trans. Algorithms*, 8(4):35:1–35:51, 2012. doi:10.1145/2344422.2344425.

9 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching. *SIAM J. Comput.*, 47(3):859–887, 2018. doi:10.1137/140998925.

10 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-Edge-Fault-Tolerant Approximate Shortest-Path Trees. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS*, pages 18:1–18:14, 2016. doi:10.4230/LIPIcs.STACS.2016.18.

11 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-Tolerant Approximate Shortest-Path Trees. *Algorithmica*, 80(12):3437–3460, 2018. doi:10.1007/s00453-017-0396-z.

12 Gilad Braunschvig, Shiri Chechik, David Peleg, and Adam Sealfon. Fault tolerant additive and (μ, α)-spanners. *Theor. Comput. Sci.*, 580:94–100, 2015. doi:10.1016/j.tcs.2015.02.036.

13 Timothy M. Chan, Mihai Patrascu, and Liam Roditty. Dynamic Connectivity: Connecting to Networks and Geometry. In *Symposium on Foundations of Computer Science, FOCS*, pages 95–104, 2008. doi:10.1109/FOCS.2008.29.

14 Bernard Chazelle and Leonidas J. Guibas. Fractional Cascading: I. A Data Structuring Technique. *Algorithmica*, 1(2):133–162, 1986. doi:10.1007/BF01840440.

15 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault Tolerant Spanners for General Graphs. *SIAM J. Comput.*, 39(7):3403–3423, 2010. doi:10.1137/090758039.

16 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-Sensitivity Distance Oracles and Routing Schemes. *Algorithmica*, 63(4):861–882, 2012. doi:10.1007/s00453-011-9543-0.

17 Lijie Chen, Ran Duan, Ruosong Wang, and Hanrui Zhang. Improved Algorithms for Maintaining DFS Tree in Undirected Graphs. *CoRR*, abs/1607.04913, 2016. URL: http://arxiv.org/abs/1607.04913v2, arXiv:1607.04913.

18 Lijie Chen, Ran Duan, Ruosong Wang, Hanrui Zhang, and Tianyi Zhang. An Improved Algorithm for Incremental DFS Tree in Undirected Graphs. In *SWAT*, pages 16:1–16:12, 2018. doi:10.4230/LIPIcs.SWAT.2018.16.

19 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004. doi:10.1145/1039488.1039492.

20 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for Distances Avoiding a Failed Node or Link. *SIAM J. Comput.*, 37(5):1299–1318, 2008. doi:10.1137/S0097539705429847.

21 Ran Duan. New Data Structures for Subgraph Connectivity. In *ICALP, Proceedings, Part I*, pages 201–212, 2010. doi:10.1007/978-3-642-14165-2_18.

22 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997. doi:10.1145/265910.265914.

23 Shimon Even and Yossi Shiloach. An On-Line Edge-Deletion Problem. *J. ACM*, 28(1):1–4, 1981. doi:10.1145/322234.322235.

24 Paolo Giulio Franciosa, Giorgio Gambosi, and Umberto Nanni. The Incremental Maintenance of a Depth-First-Search Tree in Directed Acyclic Graphs. *Inf. Process. Lett.*, 61(2):113–120, 1997. doi:10.1016/S0020-0190(96)00202-5.

25 Daniele Frigioni and Giuseppe F. Italiano. Dynamically Switching Vertices in Planar Graphs. *Algorithmica*, 28(1):76–103, 2000. doi:10.1007/s004530010032.

26 Lee-Ad Gottlieb and Liam Roditty. Improved algorithms for fully dynamic geometric spanners and geometric routing. In *Symposium on Discrete Algorithms, SODA*, pages 591–600, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347148.

27 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Symposium on Discrete Algorithms, SODA*, pages 841–850, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644250.

28 Manoj Gupta and Shahbaz Khan. Multiple Source Dual Fault Tolerant BFS Trees. In *44th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 127:1–127:15, 2017. doi:10.4230/LIPIcs.ICALP.2017.127.

29 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental Single-Source Shortest Paths on Undirected Graphs in Near-Linear Total Update Time. *J. ACM*, 65(6):36:1–36:40, 2018. URL: https://dl.acm.org/citation.cfm?id=3218657, doi:10.1145/3218657.

30 Monika Rauch Henzinger and Valerie King. Randomized Fully Dynamic Graph Algorithms with Polylogarithmic Time per Operation. *J. ACM*, 46(4):502–516, 1999. doi:10.1145/320211.320215.

31 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

32 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Symposium on Discrete Algorithms, SODA*, pages 1131–1142, 2013. doi:10.1137/1.9781611973105.81.

33 Peter Bro Miltersen, Sairam Subramanian, Jeffrey Scott Vitter, and Roberto Tamassia. Complexity Models for Incremental Computation. *Theor. Comput. Sci.*, 130(1):203–236, 1994. doi:10.1016/0304-3975(94)90159-7.

34 Kengo Nakamura and Kunihiko Sadakane. A Space-Efficient Algorithm for the Dynamic DFS Problem in Undirected Graphs. In *In International Workshop on Algorithms and Computation*, pages 295–307, 2017. doi:10.1007/978-3-319-53925-6_23.

35 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 950–961, 2017. doi:10.1109/FOCS.2017.92.

36 John H. Reif. Depth-First Search is Inherently Sequential. *Inf. Process. Lett.*, 20(5):229–234, 1985. doi:10.1016/0020-0190(85)90024-9.

37 John H. Reif. A Topological Approach to Dynamic Graph Connectivity. *Inf. Process. Lett.*, 25(1):65–70, 1987. doi:10.1016/0020-0190(87)90095-0.

38 Liam Roditty. Fully Dynamic Geometric Spanners. *Algorithmica*, 62(3-4):1073–1087, 2012. doi:10.1007/s00453-011-9504-7.

39 Liam Roditty and Uri Zwick. Improved Dynamic Reachability Algorithms for Directed Graphs. *SIAM J. Comput.*, 37(5):1455–1471, 2008. doi:10.1137/060650271.

40 Liam Roditty and Uri Zwick. Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs. *SIAM J. Comput.*, 41(3):670–683, 2012. doi:10.1137/090776573.

41 Piotr Sankowski. Dynamic Transitive Closure via Dynamic Matrix Inverse (Extended Abstract). In *Symposium on Foundations of Computer Science, FOCS*, pages 509–517, 2004. doi:10.1109/FOCS.2004.25.

42 Daniel Dominic Sleator and Robert Endre Tarjan. A Data Structure for Dynamic Trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

43 Shay Solomon. Fully Dynamic Maximal Matching in Constant Update Time. In *Symposium on Foundations of Computer Science, FOCS*, pages 325–334, 2016. doi:10.1109/FOCS.2016.43.

44 Robert Endre Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

45 Mikkel Thorup. Fully-Dynamic Min-Cut. *Combinatorica*, 27(1):91–127, 2007. doi:10.1007/s00493-007-0045-2.

RLE Edit Distance in Near Optimal Time

Raphaël Clifford
Department of Computer Science, University of Bristol, UK
Raphael.Clifford@bristol.ac.uk

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Tomasz Kociumaka
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
Institute of Informatics, University of Warsaw, Poland
kociumaka@mimuw.edu.pl

Daniel P. Martin[1]
The Alan Turing Institute, British Library, London, UK
dmartin@turing.ac.uk

Przemysław Uznański
Institute of Computer Science, University of Wrocław, Poland
puznanski@cs.uni.wroc.pl

Abstract

We show that the edit distance between two run-length encoded strings of compressed lengths m and n respectively, can be computed in $\mathcal{O}(mn \log(mn))$ time. This improves the previous record by a factor of $\mathcal{O}(n/\log(mn))$. The running time of our algorithm is within subpolynomial factors of being optimal, subject to the standard SETH-hardness assumption. This effectively closes a line of algorithmic research first started in 1993.

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

Keywords and phrases String algorithms, Compression, Pattern matching, Run-length encoding

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.66

Funding *Tomasz Kociumaka*: Supported by ISF grants no. 824/17 and 1278/16 and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (grant no. 683064).

1 Introduction

The edit distance is one of the most common distance measures between strings. For two strings of length M and N respectively, the edit distance counts the minimum number of single character insertions, deletions and substitutions needed to transform one string into the other. The first record of an $\mathcal{O}(MN)$ algorithm to compute the edit distance is from 1968 [14] although it was rediscovered independently a number of times subsequently. Masek and Paterson improved the running time to $\mathcal{O}(MN/\log N)$, for $N \geq M$, in 1980 and this is the fastest known algorithm to date [12]. Much more recently it has been shown that no $\mathcal{O}(MN^{1-\epsilon})$ time edit distance algorithm can exist, subject to the strong exponential time hypothesis (SETH) [4, 5]. As a result, it is likely that little further progress can be made in terms of improving its worst case complexity.

[1] Research carried out while the author was a member of the University of Bristol and the Heilbronn Institute for Mathematical Research.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 66; pp. 66:1–66:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we focus on the problem of computing the edit distance between two compressed strings. The run-length encoding (RLE) of a string compresses consecutive identical symbols into a run, denoted σ^i if the symbol σ is repeated i times. For example `aaabbbbaaa` would be compressed to $\mathtt{a}^3\mathtt{b}^4\mathtt{a}^3$. This form of compression is commonly used for image compression but also has wider applications including, for example, in image processing [9, 15] and succinct data structures [11].

In 1993 Bunke and Csirik proposed the first algorithm for computing the edit distance between RLE strings. For two strings of RLE-compressed lengths m and n respectively, their algorithm runs in $\mathcal{O}(mn)$ time in the special case where all the runs are of the same length [6]. However the running time falls back to the naive complexity of $\mathcal{O}(MN)$ time in the worst case where M and N are the uncompressed lengths of the two strings. This worst case complexity was subsequently improved to $\mathcal{O}(Nm + Mn)$ [3, 7] and then $\mathcal{O}(\min\{Nm, Mn\})$ time in 2007 [10]. Finally in 2013 the fastest solution prior to this current work was given running in $\mathcal{O}(mn^2)$ time, where $n \geq m$ [8]. This was the also the first algorithm for the RLE edit distance problem whose running time did not depend on the uncompressed lengths of the input strings.

For uncompressed strings, the longest common subsequence (LCS) problem has long been considered a close relative of the edit distance problem. This is partly due to the similarity of their dynamic programming solutions and partly because LCS is a special case of edit distance when general costs are allowed for the different mismatch and substitution operations. Moreover, the two problems have the same quadratic time upper bounds and SETH-hardness lower bounds [5]. Somewhat surprisingly, however, the history of algorithms for LCS and edit distance have *not* mirrored each other when the problems are considered on RLE strings. In particular, an $\mathcal{O}(mn\log(mn))$ time algorithm for computing the LCS on RLE strings was given in 1999 [2] which is considerably faster than has been possible up to this point for the edit distance problem. Some work has also been carried out since that date to improve the log factor in the running time complexity for the LCS problem [1, 13].

In this paper we speed up the running time for the edit distance problem on RLE strings by a factor of $\mathcal{O}(n/\log(mn))$, matching the fastest LCS algorithm to within a logarithmic factor and making it within subpolynomial factors of being optimal, assuming SETH holds. As a result, our new algorithm shows that the LCS and edit distance problems are indeed of essentially the same complexity even when the input strings are run-length encoded. This effectively closes a line of algorithmic research first started in 1993.

▶ **Theorem 1.** *Given two RLE strings of compressed length n and m respectively, there exists an algorithm to compute their edit distance which runs in $\mathcal{O}(mn\log(mn))$ time.*

2 Previous Work and Preliminaries

The classic dynamic programming solution for computing the edit distance between uncompressed strings X and Y of uncompressed lengths M and N respectively, computes the distance between all prefixes $X[1, \ldots, i]$ and $Y[1, \ldots, j]$. The key recurrence which enables us to do this efficiently is given by:

$$\mathsf{ED}(i, j) = \min(\mathsf{ED}(i-1, j-1) + \delta(X_i \neq Y_j), \mathsf{ED}(i-1, j) + 1, \mathsf{ED}(i, j-1) + 1).$$

From this the classic $\mathcal{O}(MN)$ time solution follows directly.

The previous approaches for the edit distance problem on RLE strings take this recurrence and the implied dynamic programming table as their starting point. The basic idea was introduced by Bunke and Csirk [6] whose algorithm works by dividing the dynamic programming table into "blocks", where each block is defined by a run in the original strings.

For each block the central task is to compute its bottom row and rightmost column given the bottom row of the block above and the rightmost column of the block to the left. For simplicity of terminology, we will refer to the rightmost column of the block to the left and the bottom row of the block above collectively as the *input border* of a block and the bottom row and rightmost column of a block as its *output border*. Figure 1 illustrates an example.

In [3,7] it was shown that the work needed to derive the values of all the output borders of blocks is at most linear in their length. When computing the edit distance between strings X and Y, the length of each row in the dynamic programming table is the uncompressed length of string Y and the length of each column is the uncompressed length of X. If there are m runs in string X and n runs in Y then the total time complexity for computing the edit distance using their approach is therefore $\mathcal{O}(Nm + Mn)$.

The work closest to ours is the $\mathcal{O}(mn^2)$ algorithm of Chen and Chao [8]. They observe that the borders of the blocks in the dynamic programming table are piecewise linear with gradient ±1 or 0. The borders can be therefore concisely represented by specifying their starting values as well as the positions and types of the points of changing gradient called the *turning points*. They prove that for a given block the number of turning points in an output border is at most a constant greater than in its input border. Consequently, a simple calculation shows that the total number of turning points is $\mathcal{O}(mn^2)$ (for $m \leq n$). Chen and Chao arrive at their final complexity by designing a procedure that computes the representation of the output border of a block, given the representation of its input border, in time proportional to the number of turning points. We now summarise their approach using our own notation.

There are two distinct types of blocks in the dynamic programming table. A match block corresponds to a rectangle where the corresponding symbols in the two strings match. A mismatch block corresponds to a rectangle where the corresponding symbols mismatch. Figure 1 shows both match and mismatch blocks. Borrowing notation from [8] we say that element (i, j) of the dynamic programming table ED is in *diagonal* $j - i$. Let (i_d, j_d) be the intersection of the input border with diagonal $j - i$.

▶ **Lemma 2** ([8, Lemma 1]). *For a match block,* $\mathsf{ED}(i, j) = \mathsf{ED}(i_d, j_d)$.

Lemma 2 indicates that for a match block we can simply copy the values from the corresponding position in the input border to derive the values of the output border. The main challenge is therefore how to handle mismatch blocks.

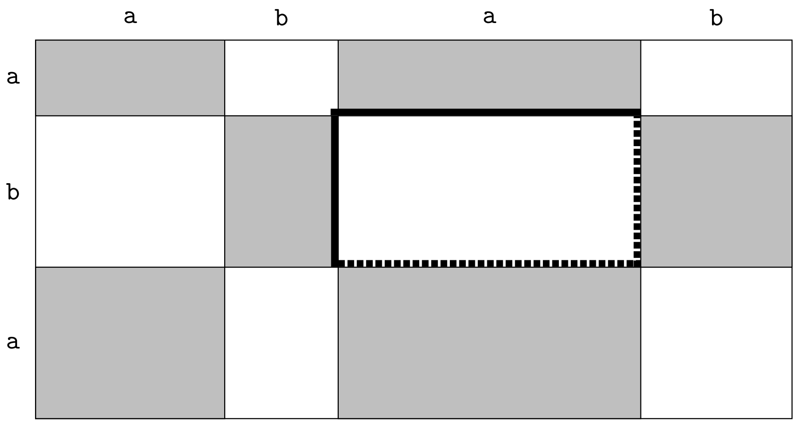


Figure 1 Match blocks are in grey. The thick line is for an input border and the dotted line an output border. The input border is contained entirely inside the neighbouring match blocks.

For mismatch blocks Chen and Chao's algorithm, in a similar manner to previous RLE edit distance algorithms, splits the problem into two parts corresponding to shortest paths that pass through the leftmost column or the top row [3, 6–8]. Consider a mismatch block of height h and width w corresponding to runs a^{h-1} and b^{w-1} such that $h \leq w$ (the other case can be processed similarly by swapping the left and the top part of the input border). $\mathrm{LEFT}[1, h]$ and $\mathrm{TOP}[1, w]$ denotes the values of the left and the top part of the input border, numbered in a bottom-to-top and a left-to-right direction, respectively. For an array $S[1, n]$ and a parameter $h \in \mathbb{Z}^+$, let $S^{(h)}[i] = \min\{S[j] \mid i - h + 1 \leq j \leq i, 1 \leq j \leq n\}$. Chen and Chao separately compute all the output border values that are derived from a value in the left part of the input border, denoted $\mathrm{OUT}_{\mathrm{LEFT}}[1, w + h - 1]$, and similarly compute all the output border values that are derived from a value in the top part of the input border, denoted $\mathrm{OUT}_{\mathrm{TOP}}[1, w + h - 1]$, as follows.

$$\mathrm{OUT}_{\mathrm{LEFT}}[i] \quad = \quad \begin{cases} \mathrm{LEFT}^{(h)}[i] + i - 1 & \text{for } i \in [1, h]; \\ \mathrm{LEFT}^{(h)}[h] + i - 1 & \text{for } i \in [h, w]; \\ \mathrm{LEFT}^{(h)}[i - w + h] + w - 1 & \text{for } i \in [w, w + h - 1]; \end{cases} \tag{1}$$

$$\mathrm{OUT}_{\mathrm{TOP}}[i] \quad = \quad \begin{cases} \mathrm{TOP}^{(h)}[i] + h - 1 & \text{for } i \in [1, w]; \\ \mathrm{TOP}^{(h)}[i] + w + h - 1 - i & \text{for } i \in [w, w + h - 1]. \end{cases} \tag{2}$$

We start with reformulating the algorithmic framework of Chen and Chao using the following notation.

▶ **Definition 3.** *Let $S[1, n]$ be a 1-indexed array of length n.*

- *For a parameter $h \in \mathbb{Z}^+$, SWM(S, h) (Sliding Window Minima) returns the array $S^{(h)}$ of length $n + h - 1$.*
- *split(S, m) returns the two subarrays $S[1, m]$, and $S[m + 1, n]$.*
- *$S \pm \overrightarrow{1}$ returns S with the gradient decreased/increased by one, or formally $S'[i] = S[i] \pm i$.*
- *For an integer constant c, $S + c$ returns S with every value increased by c.*
- *initialise(ℓ) returns an array of length ℓ initially filled with zeroes.*
- *join(S_1, S_2) simply concatenates two arrays.*

Now, equations (1) and (2) can be rephrased as Algorithms 1 and 2, respectively. The final step of the algorithm is to compute the output border as the minimum of $\mathrm{OUT}_{\mathrm{TOP}}[i]$ and $\mathrm{OUT}_{\mathrm{LEFT}}[i]$ for each index i. This is performed in linear time per block by Chen and Chao [8]. In Section 3 we will design a new implementation of both algorithms and a subtle amortised argument for this final step. The latter is based on the fundamental property of the values in an output border summarised by Lemma 4.

▶ **Lemma 4** ([8, Lemma 7]). *If there exists an i such that $\mathrm{OUT}_{\mathrm{TOP}}[i] \leq \mathrm{OUT}_{\mathrm{LEFT}}[i]$, then $\mathrm{OUT}_{\mathrm{TOP}}[j] \leq \mathrm{OUT}_{\mathrm{LEFT}}[j]$ for all $j \geq i$.*

Algorithm 1 Compute the shortest path passing through the left border.

```
$S \leftarrow \mathsf{SWM}_h(\mathrm{LEFT}[1, h], h)$;
$S^\ell, S^r \leftarrow \mathsf{split}(S, h)$;
$S_1 \leftarrow S^\ell + \overrightarrow{1} - 1$;
$S_2 \leftarrow \mathsf{initialise}(w - h) + \overrightarrow{1} + S[h] + h - 1$;
$S_3 \leftarrow S^r + w - 1$;
$\mathrm{OUT}_{\mathrm{LEFT}} \leftarrow \mathsf{join}(\mathsf{join}(S_1, S_2), S_3)$;
```

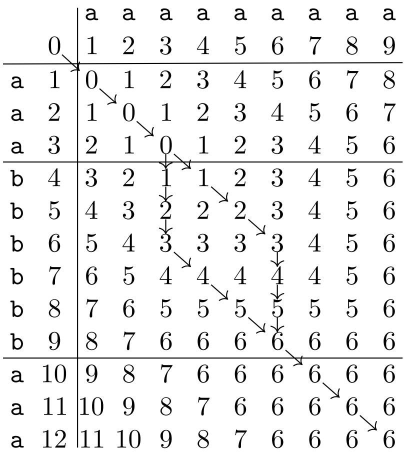
Algorithm 2 Compute the shortest path passing through the top border.

```
$S \leftarrow \mathsf{SWM}_h(\mathrm{TOP}[1,w],h)$;
$S^\ell, S^r \leftarrow \mathsf{split}(S,w)$;
$S_1 \leftarrow S^\ell + h - 1$;
$S_2 \leftarrow S^r - \overrightarrow{1} + w + h - 1$;
$\mathrm{OUT}_{\mathrm{TOP}} \leftarrow \mathsf{join}(S_1,S_2)$;
```

Before we go on to explain how we speed up the task of deriving the borders of blocks, it is worth exploring for a moment why we cannot simply apply, perhaps with some small modifications, the known $\mathcal{O}(mn\log(mn))$ time solution for LCS on RLE strings [2]. The key obstacle comes from the different nature of optimal paths in the dynamic programming table of the LCS and edit distance problems.

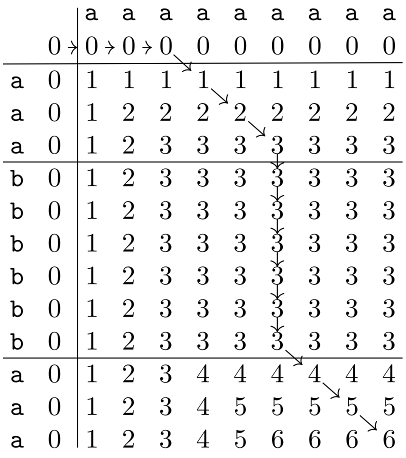
For the LCS problem on RLE strings Apostolico et al. [2] introduced two important concepts. The first is *forced paths* and the second *corner paths*. They say that a path beginning at the upper-left corner of a match block is *forced* if it traverses the block by strictly diagonal moves and, whenever the right (respectively, lower) side of an intermediate match block is reached, proceeds to the next match block by a straight horizontal (respectively, vertical) line through the mismatch blocks in between. A *corner* path is one that enters match blocks in the top left corner and exits only through the bottom right corner. They show that there is always a longest common subsequence between two strings corresponding to the concatenation of subpaths of corner and forced paths. This fact greatly reduces the number of different paths that have to be considered and hence the complexity of solving the overall LCS problem. However for the edit distance problem this property of forced paths no longer holds. Figures 2 and 3 show an example of this key difference between optimal paths under edit distance and LCS. In Figure 2 we can see that there is no optimal vertical (or horizontal) path through the mismatch block. By contrast, there is indeed an optimal vertical path for the LCS problem as illustrated by Figure 3.

In order to speed up edit distance computation on RLE strings we introduce a new data structure for input borders and output borders. This will allow us to derive the values of output borders from their respective input borders in amortised logarithmic time per border, rather than the previous linear time. The rest of the paper is devoted to this task.



		a	a	a	a	a	a	a	a	a
	0	1	2	3	4	5	6	7	8	9
a	1	0	1	2	3	4	5	6	7	8
a	2	1	0	1	2	3	4	5	6	7
a	3	2	1	0	1	2	3	4	5	6
b	4	3	2	1	1	2	3	4	5	6
b	5	4	3	2	2	2	3	4	5	6
b	6	5	4	3	3	3	3	4	5	6
b	7	6	5	4	4	4	4	4	5	6
b	8	7	6	5	5	5	5	5	5	6
b	9	8	7	6	6	6	6	6	6	6
a	10	9	8	7	6	6	6	6	6	6
a	11	10	9	8	7	6	6	6	6	6
a	12	11	10	9	8	7	6	6	6	6

Figure 2 Edit distance with forced turn in mismatch block.



		a	a	a	a	a	a	a	a	a
	0	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1	1
a	0	1	2	2	2	2	2	2	2	2
a	0	1	2	3	3	3	3	3	3	3
b	0	1	2	3	3	3	3	3	3	3
b	0	1	2	3	3	3	3	3	3	3
b	0	1	2	3	3	3	3	3	3	3
b	0	1	2	3	3	3	3	3	3	3
b	0	1	2	3	3	3	3	3	3	3
b	0	1	2	3	3	3	3	3	3	3
a	0	1	2	3	4	4	4	4	4	4
a	0	1	2	3	4	5	5	5	5	5
a	0	1	2	3	4	5	6	6	6	6

Figure 3 LCS with optimal vertical path through mismatch block.

3 Efficient Manipulation of Piecewise-Linear Functions

In this section, we describe the data structure we will use to represent input borders and output borders in the dynamic programming table. We will then show how the operations from Definition 3 can be implemented efficiently using this data structure.

Recall that *a piecewise linear function* is a real-valued function F whose domain $\mathrm{dom}(F)$ is a closed interval that can be represented as a union of closed intervals $\mathrm{dom}(F) = \bigcup_{j=1}^{k} I_j$ such that F restricted to I_j is an affine function ($g_j x + h_j$ for some coefficients g_j and h_j). The input and output borders as defined in Section 2 are by definition piecewise linear.

In this section, we impose a few further restrictions:

- For each integer $x \in \mathrm{dom}(F)$, the value $F(x)$ is also an integer.
- The gradient g_j of each $F|_{I_j}$ is -1, 0, or 1.
- The endpoints of $\mathrm{dom}(F)$ are integers or half-integers.

The graph of a piecewise linear function F is a simple polygonal curve, and thus it can be interpreted as a sequence of *turning points* connected by straight-line *segments*. Due to the restrictions imposed on F, each turning point has integer or half-integer coordinates. We represent such a function F as a sequence of segments stored in an annotated balanced binary search tree, where each segment explicitly keeps the coordinates of its endpoints.[2]

We first provide a simple implementation of curves supporting a few basic operations, and then we gradually augment it to handle more complicated operations. We conclude with an amortised running time analysis.

3.1 Basic Operations

Our first implementation just stores the corresponding segment for each node v:

(x_ℓ, y_ℓ): The coordinates of the left endpoint of the segment corresponding to v.
(x_r, y_r): The coordinates of the right endpoint of the segment corresponding to v.

Nevertheless, we are already able to implement some operations useful in Algorithms 1 and 2.

Create. The create operation produces a function F whose graph consists of just one segment S with given endpoints (x_ℓ, y_ℓ) and (x_r, y_r). This enables us to implement the whole of line 4 of Algorithm 1 in worst-case constant time.

Join. The join operation takes two functions, F_L and F_R with domains $\mathrm{dom}(F_L) = [x_L, x_M]$ and $\mathrm{dom}(F_R) = [x_M, x_R]$, respectively, and with a common endpoint $F_L(x_M) = F_R(x_M)$. It returns a function F with $\mathrm{dom}(F) = [x_L, x_R]$ such that $F_L = F|_{[x_L,x_M]}$ and $F_R = F|_{[x_M,x_R]}$. To implement this operation, we first join the two balanced binary search trees. If the rightmost segment of F_L has the same gradient as the leftmost segment of F_R, we also join these segments. The resulting tree represents F. The worst-case running time is logarithmic.

Split. The split operation takes a function F with $\mathrm{dom}(F) = [x_L, x_R]$ and a value $x_M \in \mathrm{dom}(F)$. It returns two functions $F_L = F|_{[x_L,x_M]}$ and $F_R = F|_{[x_M,x_R]}$. To implement it, we first descend the binary search tree to find a segment S with $x_M \in \mathrm{dom}(S)$. If x_M lies in the interior of $\mathrm{dom}(S)$, we split this segment into two. Next, we split the binary search tree to separate the segments to the left of x_M from the segments to the right of x_M. The resulting two trees represent F_L and F_R, respectively. The worst-case running time is logarithmic.

[2] Note that the coordinates of each internal turning point are stored with both incident segments.

Combine. The combine operation takes two functions F_1 and F_2 over the same domain $\mathrm{dom}(F_1) = \mathrm{dom}(F_2) = [x_L, x_R]$ and returns their pointwise minimum: a function F with $\mathrm{dom}(F) = [x_L, x_R]$ such that $F(x) = \min(F_1(x), F_2(x))$ for $x \in \mathrm{dom}(F)$. We assume that there exists $x_M \in [x_L, x_R]$ such that $F_1(x) > F_2(x)$ if $x < x_M$ and $F_1(x) \le F_2(x)$ if $x \ge x_M$.

If $F_1(x_L) \le F_2(x_L)$, then $x_M = x_L$. Hence, we return $F = F_1$ and discard F_2. Similarly, if $F_1(x_R) > F_2(x_R)$, then $x_M = x_R$. Hence, we return $F = F_2$ and discard F_1.

Otherwise, we are guaranteed that $F_1(x_M) = F_2(x_M)$. Our first task is to find x_M. For this, we locate segments S_1 of F_1 and S_2 of F_2 such that $x_M \in \mathrm{dom}(S_1) \cap \mathrm{dom}(S_2)$.

We observe that S_1 corresponds to the leftmost node v in the BST of F_1 such that $v.y_r = F_1(v.x_r) \le F_2(v.x_r)$. Hence, we perform a left-to-right in-order traversal of the BST to find S_1. For each visited node v, we evaluate $F_2(v.x_r)$ by descending the BST of F_2 to find a segment whose domain contains $v.x_r$. Symmetrically, S_2 corresponds to the rightmost node v in the BST of F_2 such that $F_1(v.x_\ell) > F_2(v.x_\ell) = v.y_\ell$, so we find S_2 by performing a right-to-left in-order traversal of the BST.

Next, we note that $(x_M, F_1(x_M)) = (x_M, F_2(x_M))$ is the leftmost common point of S_1 and S_2. Hence, we can now compute x_M easily (the restrictions on F_1 and F_2 guarantee that it is an integer or a half-integer). Finally, we split both F_1 and F_2 at x_M, discard $F_1|_{[x_L,x_M]}$ and $F_2|_{[x_M,x_R]}$, join $F_2|_{[x_L,x_M]}$ with $F_1|_{[x_M,x_R]}$, and return the resulting function as F.

As far as the running time is concerned, the cost is logarithmic for each discarded segment. We can now also implement the final combine step that produces our representation of the output border from the outputs of Algorithms 1 and 2 by finding the minimum at each index.

3.2 Shifts

Next, we extend our data structure to implement the shift operation which moves the whole function by a given vector. It is useful in Algorithms 1 and 2 for altering S^ℓ and S^r.

Formally, given a function F with $\mathrm{dom}(F) = [x_L, x_R]$ and a vector (Δ_x, Δ_y), we transform F into F' such that $F'(x) = F(x - \Delta_x) + \Delta_y$ for each $x \in \mathrm{dom}(F') = [x_L + \Delta_x, x_R + \Delta_x]$.

This update is performed using a technique known as *lazy propagation*. We augment each node v with the following extra field:

(δ_x, δ_y): A deferred shift to be propagated within the subtree of v.

This change is then lazily propagated as further operations are executed. Here, we rely on a key structural property of BST operations:

▶ **Observation 5.** *The execution of every BST operation can be extended (at the cost of an extra multiplicative constant in the running time) with a sequence of node* activations *and* deactivations *such that:*

- *a node v is accessed only when it is active and has no active descendant,*
- *when v is active, then all its ancestors are active,*
- *no node is active at the beginning and the end of the execution.*

The idea behind lazy propagation is that the deferred updates stored at a node v are propagated when v is activated. This way, every active node has no delayed updates pending. Hence, from the perspective of any other operation, the effect is the same as if we have meticulously modified every node for each update.

The shift propagation is very simple: when a node v receives a request for a shift by (Δ_x, Δ_y), then we just add (Δ_x, Δ_y) to the delayed shift $(v.\delta_x, v.\delta_y)$ stored at v. Upon activation of v, we propagate $(v.\delta_x, v.\delta_y)$ to the children of v, add $(v.\delta_x, v.\delta_y)$ to both (x_ℓ, y_ℓ)

and (x_r, y_r), and reset $(v.\delta_x, v.\delta_y) := (0, 0)$. To implement the *shift* operation, we just send a request for a shift by (Δ_x, Δ_y) to the root node r.

The worst-case running time of a shift is constant, and the extra cost of propagation does not increase the asymptotic running time of the remaining operations.

3.3 Gradient Changes

The gradient change operation takes a function F and a coefficient Δ_g, and it transforms F into F' such that $F'(x) = F(x) + \Delta_g \cdot x$ for each $x \in \operatorname{dom}(F') = \operatorname{dom}(F)$. This operation is needed in both Algorithms 1 and 2 to transform S^r and S^ℓ, respectively.

We first note that the constraints imposed on the gradients of functions F and F' yield that $\Delta_g = -1$, F is non-decreasing, and F' is non-increasing, or $\Delta_g = 1$, F is non-increasing, and F' is non-decreasing. However, these limitations only become relevant in Section 3.4.

To implement gradient change, we just add another field to each node v:

δ_g: A deferred gradient change to be propagated within the subtree of v.

We now have two types of lazily propagated updates: shift and gradient change. These two operations do *not* commute, so we need to decide how to interpret the two kinds of deferred updates stored at a node v. We shall assume that the gradient change by δ_g is to be performed *before* the shift by (δ_x, δ_y).

Thus, while shift propagation is implemented as in Section 3.2, adding Δ_g to $v.\delta_g$ is insufficient when a node v receives a request to change gradient by Δ_g: we also need to add $\Delta_g \cdot \delta_x$ to δ_y. This approach is correct since a shift by (Δ_x, Δ_y) followed by a gradient change by Δ_g is equivalent to a gradient change by Δ_g followed by a shift by $(\Delta_x, \Delta_y + \Delta_g \cdot \Delta_x)$.

Upon activation of v, we first apply the deferred gradient change: we propagate it to the children of v, increase $v.y_\ell$ by $v.\delta_g \cdot v.x_\ell$ and $v.y_r$ by $v.\delta_g \cdot v.x_r$, and reset $v.\delta_g = 0$. Then, we handle the deferred shift as in Section 3.2.

Finally, we note that to implement the *gradient change* operation, we just send a request for a gradient change by Δ_g to the root node r. The worst-case running time is constant.

3.4 Sliding Window Minima

We can finally show how to implement the SWM operation efficiently on our data structure. This is the most involved of the operations we will need. The SWM operation given a function F with $\operatorname{dom}(F) = [x_L, x_R]$ and a *window width* t, returns a function F' with $\operatorname{dom}(F') = [x_L, x_R + t]$ such that $F'(x) = \min\{F(x') : x' \in [x - t, x] \cap \operatorname{dom}(F)\}$.

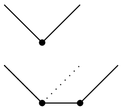
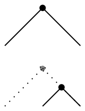
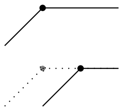
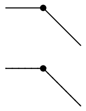
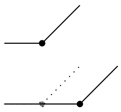
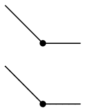
Combinatorial Properties

We begin by observing that the SWM operation is composable.

► **Observation 6.** *Every function F and positive window widths t, t' satisfy $\mathsf{SWM}(\mathsf{SWM}(F, t), t') = \mathsf{SWM}(F, t + t')$.*

Hence, instead of applying $\mathsf{SWM}(\cdot, t)$ for an integer width t, we may equivalently apply the $\mathsf{SWM}(\cdot, 1)$ operation t times. The key property of width 1 is that the changes to the transformed function are very local. The structure of these modifications can be described in terms of *types* of turning points. We classify internal turning points by the gradients (**I**ncreasing, **F**lat, or **D**ecreasing) of the incident segments; see Table 1, where we also analyse how a function changes in the vicinity of each turning point subject to $\mathsf{SWM}(\cdot, 1)$.

Table 1 Types of internal turning points and their behaviour subject to the SWM operation.

Type DI	Type ID	Type IF	Type FD	Type FI	Type DF
					

- A point of type FD or DF remains intact.
- A point of type FI or IF is shifted by $(1, 0)$.
- A point of type ID is shifted by $(0.5, -0.5)$.
- A point of type DI transformed into a point of type DF and a point of type FI, and the latter is shifted by $(1, 0)$.

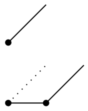
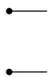
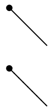
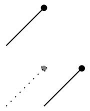
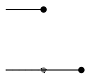
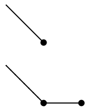
Note that the behaviour of DI points is unlike that of other types. However, this discrepancy disappears if we replace every DI point with two coinciding points of types DF and FI, respectively, with an artificial length-0 segment in between. Hence, whenever a new internal turning point is created (which happens only within the join operation), if the turning point would be of type DI, we pre-emptively replace it by two coinciding points of type DF and FI, respectively. Note that the resulting length-0 segment never changes its gradient since gradient change is allowed only on a monotone function. However, when an incident segment is modified, we may need to remove the length-0 segment. This process cannot cascade, though, causing another length zero segment to be removed.

Next, we analyse in Table 2 how the $\mathsf{SWM}(F, 1)$ operation affects the endpoints of the graph of F. In most cases, the left endpoint stays intact and the right endpoint is shifted by $(0, 1)$. The only exceptions are endpoints of type -I and D-, which exhibit similar behaviour to the internal turning points of type DI. Moreover, this discrepancy also disappears if we introduce artificial flat segments of length 0. Hence, we replace a point of type -I with two points of type -F and FI, respectively, and a point of type D- with two points of type DF and F-, respectively. However, this time the replacement is *not* pre-emptive: we perform it as the first step in the implementation of the SWM operation. This is possible because there are just two endpoints, while the number of internal turning points of type DI could be large. Our gain, on the other hand, is that we avoid length-0 segments changing their gradients.

With the artificial length-0 segments in place, it is now true that the effect of the SWM operation on each turning point can be described as a shift depending only on the type of the point. As a result of these shifts, some segments may disappear as their length reaches 0; in this case, we say that a segment *collapses*. Only segments of three kinds may collapse:

- a segment between a point of type ID and point of type DF;
- a segment between a point of type IF and point of type FD;
- a segment between a point of type FI and point of type ID;

Table 2 Types of endpoints and their behaviour subject to the SWM operation.

Type -I	Type -F	Type -D	Type I-	Type F-	Type D-
					

When a segment collapses, it is removed and the two incident turning points are merged.[3] Each segment of the three affected kinds has the *collapse time*, defined as the smallest t for which $\mathsf{SWM}(\cdot, t)$ makes it collapse (assuming no interaction from incident segments) equal to the Manhattan distance between its endpoints. Note that due to the restrictions on the piecewise linear functions considered in this section, the collapse time is always an integer.

Implementation

To implement the SWM operation, we augment each node v with the following fields:

$\mathbf{type}_\ell, \mathbf{type}_r$: The types of the turning points joined by the segment corresponding to v.
$\boldsymbol{\delta_t}$: The amount of a deferred SWM to be propagated within the subtree of v.
$\boldsymbol{t}_{\mathbf{min}}$: The minimum collapse time among the segments in the subtree of v.

Note that the type of each internal turning point is stored twice. Hence, whenever a node type changes, this fact needs to be reflected at both incident segments (and we need to reach the corresponding nodes by descending the BST; shortcuts would violate Observation 5).

The field $v.t_{\min}$ is of a kind we have not encountered yet: its value depends on the corresponding values for the children of v and on other fields at v. It is brought up to date whenever v is deactivated (so that it can be accessed only when v is inactive). We shall assume that its value already reflects the deferred updates stored at v. The procedure of recomputing $v.t_{\min}$ is simple: we determine the collapse time of the segment represented by v (which is infinite or equal to $|v.x_r - v.x_\ell| + |v.y_r - v.y_\ell|$ depending on the types of the incident turning points), and take the minimum of this value and $u.t_{\min}$ for every child u of v. Since v has no deferred changes when it is deactivated, the resulting minimum is $v.t_{\min}$.

Propagation. The main structural modification to the lazy propagation procedures is that we maintain an additional invariant that no deferred changes are stored on the leftmost and on the rightmost path of the BSTs representing every function F. To maintain this invariant, immediately after lazily updating of the whole F (sending a request to the root node r), we descend to the leftmost and to the rightmost segment F; this increases the cost of shift and gradient change to logarithmic. Note that the split operation must anyway visit the nodes representing the new boundary segments (to update the types of new endpoints). Moreover, if a path from the root to a given node v contains no deferred updates, then this is still true after any rebalancing of the BST (as only active nodes get rotated).

Concerning the lazy SWM propagation, we explicitly forbid requesting for SWM with window width exceeding $r.t_{\min}$, because collapsed segments need to be removed before we proceed further. Also, the window widths (and hence the values δ_t) are always non-negative.

We have three kinds of deferred updates now: SWM, gradient change, and shift. We fix the semantics of the fields δ_t, δ_g, and (δ_x, δ_y) so that an SWM of width δ_t is performed first, a gradient change by δ_g second, and a shift by (δ_x, δ_y) last. The requests for a shift and for a gradient change are still implemented as in Section 3.3; note that these updates do not affect the collapse times (the three segment kinds with finite collapse times cannot appear in monotone functions). On the other hand, the request for an SWM with a window width Δ_t requires more care. We clearly need to increase δ_t by Δ_t and decrease $t_{\min}$ by the same amount. The aforementioned steps suffice if $\delta_g = 0$. Otherwise, we note that the turning

[3] Two adjacent segments may collapse simultaneously. In that special case, three subsequent points, of type FI, ID, and DF, respectively, need to be deleted.

points in the subtree of v are all of types DF and FD or all of types IF and FI. (Observe that there are no deferred changes in the proper ancestors of v and that v is not on the leftmost or rightmost path in this case.) We can easily distinguish the two cases by analysing the endpoints of the segment corresponding to v. Moreover, the SWM operation is void in the first of these cases, and in the second one it reduces to a shift by $(\Delta_t, 0)$. Hence, we shall implement it this way rather than by modifying $v.\delta_t$.

The propagation itself is relatively easy: upon activation of a node v, we first propagate the SWM operation to the children of v, update the endpoints of the segment corresponding to v (according to Tables 1 and 2, with the shift multiplied by δ_t), and finally reset $\delta_t = 0$. Then, we propagate the gradient change and the shift. This is implemented as in Section 3.3 except that the gradient change now affects not only the coordinates but also the types of the segment's endpoints.

SWM Procedure. To implement the SWM procedure itself, we first check the endpoint types and perform appropriate substitutions for endpoints of type -I and D-. Next, we would like to lazily apply the SWM operation with window width t to the root r. However, this could result in negative collapse time $r.t_{\min}$, so instead we perform SWM gradually based on Observation 6. If $r.t_{\min} < t$, we make a request for SWM with window width just $r.t_{\min}$, leaving the remaining quantity $t - r.t_{\min}$ for later on. This already results in $r.t_{\min} = 0$, which indicates that there is a collapsed segment. We descend the tree to find such a collapsed segment (activating nodes on the way there and deactivating them on the way back), and take care of this segment appropriately (this may affect neighbouring segments as well). We repeat the process as long as $r.t_{\min} = 0$. Once this value is positive again, we are ready to proceed with further application of SWM.

As far as the running time is concerned, the cost of SWM consists of a logarithmic term for visiting the endpoints and further logarithmic terms for each collapsed segment.

3.5 Complexity Analysis

We complete this section by showing that the aforementioned operations run in *amortised* logarithmic time.

▶ **Lemma 7.** *A sequence of k operations on piecewise linear functions takes $\mathcal{O}(k \log k)$ time.*

Proof. Our potential is $\log k$ multiplied by the total number of turning points in all the stored functions. First, we observe that this potential grows by $\mathcal{O}(\log k)$: each operation creates a constant number of new turning points. In particular, the total number of turning points is $\mathcal{O}(k)$, so manipulating BSTs takes $\mathcal{O}(\log k)$ time. Next, we note that the worst-case running time of most operations is $\mathcal{O}(\log k)$, with extra $\mathcal{O}(\log k)$ time needed for each discarded or collapsed segment. However, every such segment decreases the potential by $\log k$. ◀

4 An $\mathcal{O}(mn \log(mn))$ time RLE Edit Distance Algorithm

As in the previous algorithm by Chen and Chao [8], we go through the dynamic programming table block by block. For every block, we transform the representation of its input border to the representation of its output border. As mentioned earlier, borders are piecewise linear with gradient ±1 or 0 so they can be maintained in the structure described in Section 3. We will assume that the left and the top part of the input border of every block are stored in separate structures. We start by generating the structures corresponding to the left and the top border of the whole dynamic programming table. These left and top borders are each a single decreasing and increasing sequence, respectively. As a result, we can generate the data structure for the parts corresponding to all blocks trivially in $\mathcal{O}(m+n)$ time using $m+n$ create operations. Now, we have to describe how to obtain the structure corresponding to the right and the bottom part of the output border of the current block given the structures corresponding to the left and the top part of its input border. We stress that any structure will be created and then used once as an input to a further operation, which is crucial for the amortisation argument within Lemma 7.

Recall that the semantics of split and join operating on arrays in Section 2 and of split and join operating on piecewise linear functions in Section 3 is slightly different: split now creates two functions that both contain the value of the original function at x_M; symmetrically, join takes two functions defined on $[x_L, x_M]$ and $[x_M, x_R]$ that share the same value at x_M. This is, however, not an issue because the cases in both (1) and (2) overlap at the boundaries.

For a match block, the value stored in an element (i, j) of the output border is a copy of the value stored in the corresponding element (i_d, j_d) of the input border. Recalling that (i_d, j_d) is the intersection of the input border with diagonal $j - i$, this can be readily implemented with a constant number of split and join operations.

For a mismatch block, we need to apply Algorithms 1 and 2, merge the returned solutions by taking the minimum at every position, and finally create separate structures corresponding to the right and the bottom part of the output border with a single split operation. Note that while we have already observed that both input border and output border are piecewise linear with gradient ±1 or 0, we need to make sure that the same is true for every function obtained inside Algorithms 1 and 2, and for $\mathrm{OUT}_{\mathrm{TOP}}$ and $\mathrm{OUT}_{\mathrm{LEFT}}$ in particular.

▶ **Lemma 8.** *Every function obtained in Algorithms 1 and 2 is piecewise linear with gradient* ± 1 *or* 0.

Proof. Consider Algorithm 1. It is easy to verify that S and hence also S^ℓ and S^r are indeed piecewise linear with gradient ±1 or 0. Additionally, $S^\ell[i]$ is equal to the minimum in $\mathrm{LEFT}[1, i]$ and so S^ℓ is non-increasing. Consequently, S_1, S_2, and S_3 are all piecewise linear with gradient ±1 or 0. We only need to verify that the same holds for their concatenation. This is true because each of these three parts corresponds to a case considered in (1), and these cases overlap at the boundaries.

Next, consider Algorithm 2. Similarly as above, it is easy to verify that S and so also S^ℓ and S^r are piecewise linear with gradient ±1 or 0. Furthermore, $S^r[i]$ is equal to the minimum in $\mathrm{TOP}[w-h+i+1, w]$ and so S^r is non-decreasing. Thus, S_1 and S_2 are piecewise linear with gradient ±1 or 0 and the same holds for their concatenation because the cases in (2) overlap at the boundaries. ◀

We now explain in detail how to implement Algorithm 1. We start with computing S^ℓ and S^r by first calling $\mathsf{SWM}(\mathrm{LEFT}, h-1)$ and then using split. Next, S_1 is obtained by applying gradient change and shift to S^ℓ, S_2 is obtained by calling create, and S_3 is obtained by applying shift to S^r. Finally, $\mathrm{OUT}_{\mathrm{LEFT}}$ is created with two calls to join.

Algorithm 2 is implemented by calling $\mathsf{SWM}(\mathrm{TOP}, w-1)$ and then using split. Next, S_1 is obtained by applying shift to S^ℓ, while S_2 is obtained by applying gradient change and shift to S^r. Finally, $\mathrm{OUT}_{\mathrm{TOP}}$ is created by a single call to join.

Having obtained a representation of $\mathrm{OUT}_{\mathrm{LEFT}}$ and $\mathrm{OUT}_{\mathrm{TOP}}$, we call combine to obtain a representation of the output border. Such a call is valid due to Lemma 4. The overall number of operations on structures is $\mathcal{O}(mn)$, making the final time complexity $\mathcal{O}(mn\log(mn))$ by Lemma 7.

References

1 Hsing-Yen Ann, Chang-Biau Yang, Chiou-Ting Tseng, and Chiou-Yi Hor. A fast and simple algorithm for computing the longest common subsequence of run-length encoded strings. *Information Processing Letters*, 108(6):360–364, 2008. doi:10.1016/j.ipl.2008.07.005.

2 Alberto Apostolico, Gad M. Landau, and Steven Skiena. Matching for Run-Length Encoded Strings. *Journal of Complexity*, 15(1):4–16, 1999. doi:10.1006/jcom.1998.0493.

3 Ora Arbell, Gad M. Landau, and Joseph S. B. Mitchell. Edit distance of run-length encoded strings. *Information Processing Letters*, 83(6):307–314, 2002. doi:10.1016/S0020-0190(02)00215-6.

4 Arturs Backurs and Piotr Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (Unless SETH is False). *SIAM Journal on Computing*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

5 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In Venkatesan Guruswami, editor, *56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.

6 Horst Bunke and János Csirik. An algorithm for matching run-length coded strings. *Computing*, 50(4):297–314, 1993. doi:10.1007/BF02243873.

7 Horst Bunke and János Csirik. An improved algorithm for computing the edit distance of run-length coded strings. *Information Processing Letters*, 54(2):93–96, 1995. doi:10.1016/0020-0190(95)00005-W.

8 Kuan-Yu Chen and Kun-Mao Chao. A fully compressed algorithm for computing the edit distance of run-length encoded strings. *Algorithmica*, 65(2):354–370, 2013. doi:10.1007/s00453-011-9592-4.

9 Stuart C. Hinds, James L. Fisher, and Donald P. D'Amato. A document skew detection method using run-length encoding and the Hough transform. In *10th International Conference on Pattern Recognition, ICDR 1990*, volume 1, pages 464–468. IEEE Computer Society, 1990. doi:10.1109/ICPR.1990.118147.

10 Jia Jie Liu, Guan-Shieng Huang, Yue-Li Wang, and Richard C. T. Lee. Edit distance for a run-length-encoded string and an uncompressed string. *Information Processing Letters*, 105(1):12–16, 2007. doi:10.1016/j.ipl.2007.07.006.

11 Veli Mäkinen and Gonzalo Navarro. Succinct suffix arrays based on run-length encoding. *Nordic Journal of Computing*, 12(1):40–66, 2005. URL: https://users.dcc.uchile.cl/~gnavarro/ps/njc05.pdf.

12 William J. Masek and Mike Paterson. A faster algorithm for computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1.

13 Yoshifumi Sakai. Computing the longest common subsequence of two run-length encoded strings. In Kun-Mao Chao, Tsan-sheng Hsu, and Der-Tsai Lee, editors, *23rd International Symposium on Algorithms and Computation, ISAAC 2012*, volume 7676 of *LNCS*, pages 197–206. Springer, 2012. doi:10.1007/978-3-642-35261-4_23.

14 Taras K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968. doi:10.1007/bf01074755.

15 Dong-Hui Xu, Arati S. Kurani, Jacob D. Furst, and Daniela S. Raicu. Run-length encoding for volumetric texture. In Juan J. Villanieva, editor, *4th IASTED International Conference on Visualization, Imaging, and Image Processing, VIIP 2004*. Acta Press, 2004. URL: http://facweb.cs.depaul.edu/research/vc/Publications/final_submission_paper_452_131_last.pdf.

Indexing Graph Search Trees and Applications

Sankardeep Chakraborty
RIKEN Center for Advanced Intelligence Project, Japan
sankar.chakraborty@riken.jp

Kunihiko Sadakane
The University of Tokyo, Japan
sada@mist.i.u-tokyo.ac.jp

Abstract

We consider the problem of compactly representing the Depth First Search (DFS) tree of a given undirected or directed graph having n vertices and m edges while supporting various DFS related queries efficiently in the RAM with logarithmic word size. We study this problem in two well-known models: *indexing* and *encoding* models. While most of these queries can be supported easily in constant time using $O(n \lg n)$ bits[1] of extra space, our goal here is, more specifically, to beat this trivial $O(n \lg n)$ bit space bound, yet not compromise too much on the running time of these queries. In the *indexing* model, the space bound of our solution involves the quantity m, hence, we obtain different bounds for sparse and dense graphs respectively. In the *encoding* model, we first give a space lower bound, followed by an almost optimal data structure with extremely fast query time. Central to our algorithm is a partitioning of the DFS tree into connected subtrees, and a compact way to store these connections. Finally, we also apply these techniques to compactly index the shortest path structure, biconnectivity structures among others.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Depth First Search Tree, Compact Data Structures, Encoding Schemes

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.67

Funding This work was partially supported by JST CREST Grant Number JPMJCR1402, Japan.

1 Introduction

Depth First Search (DFS) is a very well-known method for visiting the vertices and edges of a directed or undirected graph. DFS differs from other ways of traversing the graph such as Breadth First Search (BFS) by the following DFS protocol: Whenever two or more vertices were discovered by the search method and have unexplored incident (out)edges, an (out)edge incident on the most recently discovered such vertex is explored first. This DFS traversal produces a rooted spanning tree (forest), called DFS tree (forest) along with assigning an index to every vertex v i.e., the time vertex v is discovered for the first time during DFS. We call it depth-first-index (DFI(v)). Let $G = (V, E)$ be a graph on $n = |V|$ vertices and $m = |E|$ edges where $V = \{v_1, v_2, \cdots, v_n\}$. It takes $O(m + n)$ time to perform a DFS traversal of G and to generate its DFS tree (forest) with DFIs of all the vertices. The DFS rule confers a number of structural properties on the resulting graph traversal that cause DFS to have a large number of applications. These properties are captured in the DFS tree (forest), and can be used crucially to design efficient algorithms for many basic and fundamental algorithmic graph problems, namely, biconnectivity [22], 2-edge connectivity [23], strongly connected components [22], topological sorting [22], dominators [24], *st*-numbering [13] and planarity testing [17] among many others.

[1] We use lg to denote logarithm to the base 2.

There are two versions of DFS studied in the literature. In the lexicographically smallest DFS or lex-DFS problem, when DFS looks for an unvisited vertex to visit in an adjacency list, it picks the "first" unvisited vertex where the "first" is with respect to the appearance order in the adjacency list. The resulting DFS tree will be unique. In contrast to lex-DFS, an algorithm that outputs some DFS numbering of a given graph, treats an adjacency list as a set, ignoring the order of appearance of vertices in it, and outputs a vertex ordering Q such that there exists some adjacency ordering R such that Q is the DFS numbering with respect to R. We say that such a DFS algorithm performs general-DFS. In this work, we focus only on lex-DFS, thus, given a source vertex, the DFS tree is always unique. Given the lex-DFS tree, the non-tree edges of a given directed graph can be classified into four categories as follows. An edge directed from a vertex to its ancestor in the tree is called a back edge. Similarly, an edge directed from a vertex to its descendant in the tree is called a forward edge. Further, an edge directed from right to left in the DFS tree is called a cross edge. The remaining edges directed from left to right in the tree are called anti-cross edges. In the undirected graphs, there are no cross edges. Note that, we can store the complete DFS tree explicitly using $O(n \lg n)$ bits by storing pointers between nodes. In what follows, we formally define the problem which we call the DFS-Indexing problem.

DFS-Indexing problem
Input: A directed or undirected graph $G = (V, E)$ where $|V| = n$, $|E| = m$, and a source vertex v_s, preprocess G and answer the following queries with respect to the DFS tree T rooted at v_s:

1. Given any pair of vertices v_i and v_j,
   a. Who is visited first in the DFS traversal of G?
   b. Is v_i an ancestor of v_j in T?
2. Given v_i,
   a. Return the parent of v_i in T.
   b. Return the number of children (if any) of v_i in T.
   c. Enumerate all the children (if any) of v_i in T.
   d. Return the DFI of v_i.
3. Enumerate the order in which vertices of G are visited in the DFS.
4. Given $1 \leq i \leq n$, return the vertex with DFI i.

We study the DFS-Indexing problem in two well-known models: the *indexing* and *encoding* models [21]. In the indexing model, we wish to build an index *ind* after preprocessing the input graph G such that queries can be answered using both *ind* and G whereas in the encoding model, we seek to build a data structure *encod* after preprocessing the input graph G such that queries have to be answered using *encod* only. Typically the parameters of interest are (i) query time, (ii) space consumed (in bits) by *ind* and *encod* resp. and (iii) the preprocessing time and space. We address all these issues in our paper for the DFS-Indexing problem, assuming our computational model is a Random-Access-Machine with constant time operations on $O(\lg n)$-bit words. In both models, it is not hard to see that using $O(n \lg n)$ bits, we can answer all the queries of the DFS-Indexing problem in the optimal $O(1)$ time except the query of 3 which takes $O(n)$ time. Our main objective here is to beat this trivial $O(n \lg n)$ bit space bound without compromising too much on the query time.

The motivation for studying this question mainly stems from the rise of the "big data" phenomenon and its implications. To illustrate, the rate at which we store data is increasing even faster than the speed and capacity of computing hardware. Thus, if we want to use the stored data efficiently, we need to represent it in sophisticated ways. Many applications dealing with huge data structures can benefit from keeping them in compressed form. Compression has many advantages: it can allow a representation to fit in main memory rather than swapping out to disk, and it improves cache performance since it allows more data to fit into the cache. However, such a data structure is only handy if it allows the application to perform fast queries to the data, and this is the direction we want to explore for the DFS tree. More specifically, we are interested in representing the DFS tree of a given graph compactly while supporting all the queries mentioned above efficiently.

1.1 Representation of the Input Graph

We assume that the input graphs $G = (V, E)$ are represented using the *adjacency array* format, i.e., G is given by an array of length $|V|$ where the i-th entry stores a pointer to an array that stores all the neighbors of the i-th vertex. For the directed graphs, we assume that the input representation has both in/out adjacency array for all the vertices i.e., for directed graphs, every vertex v has access to two arrays, one array is for all the in-neighbors of v and the other array is for all the out-neighbors of v. This form of input graph representation has now become somewhat standard and was recently used in plenty of other works [2, 6, 7, 8, 9, 10]. Throughout this paper, we call a graph sparse when $m = O(n)$, and dense otherwise (i.e., $m = \omega(n)$).

1.2 Our Main Results and Organization of the Paper

We start by mentioning some preliminary results that will be used throughout the paper in Section 2. Section 3 contains the description of our main index for solving the DFS-Indexing problem in the indexing model. Our main results here can be summarized as follows,

▶ **Theorem 1.** *In the indexing model, given any sparse (dense resp.) undirected or directed graph G, there exists an $O(m + n)$ time and $O(n \lg n)$ bits preprocessing algorithm which outputs a data structure of size $O(n)$ ($O(n \lg(m/n))$ resp.) bits, using which the queries 1(a), 1(b), 2(d) and 4 can be reported in $O(\lg n)$ time, 2(a) and 2(b) in $O(1)$ time, 2(c) in time proportional to the number of solutions, and finally 3 can be solved in $O(n)$ time resp. for the* DFS-Indexing *problem.*

We want to emphasize that obtaining better results for sparse graphs is not only interesting from theoretical perspective but also from practical point of view as these graphs do appear very frequently in most of the realistic network scenario in real world applications, e.g., Road networks and the Internet.

In Section 4, we provide the detailed proof of our index in the encoding model. This contains a space lower bound for any index for the DFS-Indexing problem, followed by an index whose size asymptotically matches the lower bound and has efficient query time. We summarize our main results below.

▶ **Theorem 2.** *In the encoding model, the size of any data structure for the* DFS-Indexing *problem must be $\Omega(n \lg n)$ bits. On the other hand, given any (un)directed graph, there exists an $O(m + n)$ time and $O(n \lg n)$ bits preprocessing scheme that outputs an index of size $(1 + \epsilon)n \lg n + 2n + o(n)$ bits (for any constant $\epsilon > 0$), using which the queries 1(a), 1(b), 2(a), 2(b), 2(d) can be reported in $O(1)$ time, 2(c) in time proportional to the number of solutions, 3 in $O(n/\epsilon)$ time, and finally 4 in $O(1/\epsilon)$ time resp. for the* DFS-Indexing *problem in this setting.*

Building on all these aforementioned results, we also show a host of applications of our techniques in designing indices for other fundamental graph problems. We provide the details about these results in the full version of our paper. Finally, we conclude in Section 5 with some open problems and possible future directions to explore further.

Remark. At this point we want to emphasize that our results are more general, i.e., they can be extended to store any arbitrary labeled tree (arising from some underlying graph) along with the mechanism for fast querying. This method is very useful as many graph algorithms (like shortest path, minimum spanning tree, biconnectivity etc) induce a tree structure which is used subsequently during the execution of the algorithm. Hence, we can use our technique to store and query those trees compactly as well as efficiently. Thus, we also believe that our algorithm may find many other potential interesting applications. However, we chose to provide all the details in terms of DFS as DFS is very widely popular graph traversal technique and is used as the backbone for multiple fundamental algorithms, yet there is no explicit indexing scheme for storing DFS tree compactly. In the full version of our paper, we show how one can extend these techniques to design indexing schemes for a variety of other classical and fundamental graph problems.

1.3 Related Works

There already exists a large body of work concerning compactly representing various specific classes of graphs, for example planar, constant genus graphs etc [1, 5, 16, 18, 20, 21, 25]. All of these works are able to store an n-vertex unlabeled planar graph in $O(n)$ bits, and some of them even allow for $O(1)$-time neighbor queries. Generally what is meant by unlabeled is that the algorithm is free to choose an ordering on the vertices (integer labels from 1 to n). Our setting here is slightly different as we work with graphs whose vertices are labeled, and matches closely with [3]. Also we want to support more complex queries whereas the previous works only focused on adjacency queries mostly. Even though DFS being such a widely known method, and having many applications, to the best of our knowledge, we are not aware of any previous work focusing on compactly representing the DFS tree with efficient query support.

2 Preliminaries

Rank-Select. We make use of the following theorem:

▶ **Theorem 3** ([11]). *We can store a bitstring B of length n with additional $o(n)$ bits such that rank and select operations (defined below) can be supported in $O(1)$ time. Such a structure can also be constructed from the given bitstring in $O(n)$ time and space.*

For any $a \in \{0, 1\}$, the rank and select operations are defined as follows :

- $rank_a(B, i)$ = the number of occurrences of a in $B[1, i]$, for $1 \leq i \leq n$;
- $select_a(B, i)$ = the position in B of the i-th occurrence of a, for $1 \leq i \leq n$.

When the bitvector B is sparse, the space overhead of $o(n)$ bits can be avoided by using the following theorem, which will also be used later in our paper.

▶ **Theorem 4** ([21]). *We can store a bitstring B of length n with m 1s using $m \lg(n/m) + O(m)$ bits such that $select_1(B, 1)$ can be supported in $O(1)$ time, $select_0(B, 1)$ in $O(\lg m)$ time, and both the rank queries ($rank_1(B, i)$ and $rank_0(B, i)$) can be supported in $O(min(\lg m, \lg n/m))$ time. Such a structure can also be constructed from B in $O(n)$ time and space.*

Permutation. We also use the following theorem:

▶ **Theorem 5** ([19]). *A permutation π of length n can be represented using $(1+\epsilon)n \lg n$ bits so that $\pi(i)$ is answered in $O(1)$ time and π^{-1} in time $O(1/\epsilon)$ for any constant $\epsilon > 0$. Such a representation can be constructed using $O(n)$ time and space.*

Succinct Tree Representation. We need following result from [15].

▶ **Theorem 6** ([15]). *There exists a data structure to succinctly encode an ordered tree with n nodes using $2n + o(n)$ bits such that, given a node v, (a) child(v,i): i-th child of v, (b) degree(v): number of children of v, (c) depth(v): depth of v, (d) $select_{pre}(v)$: position of v in preorder, (e) $LA(v, i)$: ancestor of v at level i can be supported in $O(1)$ time among many others. Such a structure can also be constructed in $O(n)$ time and space.*

3 Algorithms in the Indexing Model

In this section, we provide the main algorithmic ideas needed for the solution of the DFS-Indexing problem in the indexing model. We start by describing the preprocessing procedure which is followed by the query algorithms.

3.1 Preprocessing Step

We first describe our algorithms for undirected graphs, and later mention the modifications required for the case of directed graphs. The preprocessing step of the algorithm is divided into two parts. In the first part, we perform a DFS of the input graph G along with storing some necessary data structures. In the second step, we perform a partition of the DFS tree of G using the well-known "tree covering technique" of the succinct data structures world [14], and also store some auxiliary data structures. Later, in the final step of our algorithm, we show how to use these data structures to answer the required queries. In what follows, we describe each step in detail.

Step 1: Creating Parent-Child Array using Unary Degree Sequence Array. The main idea of this step is to perform a DFS traversal of G and store in a *compact way* the parent-child relationship of the DFS tree T. The way we achieve this is by using three bitvectors of length $O(m + n)$ bits. Recall that, our input graphs $G = (V, E)$ are represented using the standard adjacency array. Central to our preprocessing algorithm is an encoding of the degrees of the vertices in unary. As usual, let $V = \{v_1, v_2, \cdots, v_n\}$ be the vertex set of G. The unary degree sequence encoding D of the undirected graph G has n 1s to represent the n vertices and each 1 is followed by a number of 0s equal to its degree. Moreover, if d is the degree of vertex v_i, then d 0s following the i-th 1 in the D array corresponds to d neighbors of v_i (or equivalently the edges from v_i to the d neighbors of v_i) in the same order as in the adjacency array of v_i. Clearly D uses $n + 2m$ bits and can be obtained from the neighbors of each vertex in $O(m + n)$ time. Now using *rank/select* queries of Theorem 3 in Section 2, the j-th outgoing edge of vertex v_i can be identified with the position $p = select_1(D, i) + j$ of D $(1 \leq j \leq degree(v_i)$ where $degree(v_i)$ denotes the degree of the vertex v_i). From a position p, we can obtain an endpoint of the corresponding edge by $i = rank_1(D, p)$, and the other endpoint is the j-th neighbor of v_i where $j = p - select_1(D, i)$.

We also use two bitvectors E, P of the same length where every bit is initialized to 0, and the bits in E, P are in one-to-one correspondence with bits in D. The bitvector E will be used to mark the tree edges of the DFS tree T, and the bitvector P to mark the unique parent of every vertex in T. The marking is carried out while performing a DFS of G in the preprocessing step. I.e., if (v_i, v_j) is an edge in the DFS tree where v_i is the parent of v_j, and suppose k is the index of the edge (v_i, v_j) in D, then the corresponding location in E is marked as 1 during DFS. At the same time, we scan the adjacency array of v_j to find the position of v_i (as G is undirected, there will be two entries for each edge in the adjacency array), and suppose t is the index of the edge (v_j, v_i) in D, then the corresponding location in P is marked as 1 during DFS. Thus, assuming G is a connected graph, once DFS finishes traversing G, the number of ones in E is exactly the number of tree edges (which is $n-1$) and the number of ones in P will be $n-1$ as root does not have any parent.

The parent of v_i in T is computed in $O(1)$ time as follows. Let v_r be the root of T. Then if $i > r$ (resp. $i < r$), the marked bit representing the parent of v_i is the $(i-1)$-st (resp. i-th) 1 in P. Let $p = select_1(P, i-1)$ (resp. $p = select_1(P, i)$) and $j = p - select_1(D, i)$. Then the parent of v_i is the j-th neighbor of v_i.

We use another bitvector D_T of length $2n$, which encodes the degree of each vertex in T by unary sequences. Then the degree of vertex v_i in T is $select_1(D_T, i+1) - select_1(D_T, i) - 1$, and j-th child of v_i in T is p-th neighbor of v_i in G where $p = select_1(E, select_1(D_T, i-1) + j) - select_1(D, i)$. These are computed in constant time.

Note that, the classical linear time implementation of DFS [12] uses a stack (which could grow to $O(n \lg n)$ bits) and a color array (of size $O(n)$ bits). Thus, the procedure takes $O(m+n)$ time and $O(n \lg n)$ bits overall. First, we argue that using the same linear time, we can also create bitvectors D, E and P and fill up them correctly. It's easy to see that creating D as well as initializing E and P to all zero takes $O(m+n)$ time. All it remains is to show, how one can fill up E and P while performing DFS. For this purpose, we build the data structures to support the constant time *rank/select* query (of Theorem 3) on D (and on E, P as well, the reason will be clear in the query step) and use the result of the select query to mark the tree edges on E (as they are in one-to-one correspondence). To illustrate, suppose, while traversing from v_i, DFS discovers the edge (v_i, v_j) as a tree edge in T where v_i is the parent of v_j, and suppose v_j is the c-th neighbor in v_i's adjacency array, then we find the index of the c-th zero after i-th one in D (using select query), and the corresponding index is marked as 1 in the E array. This takes $O(1)$ time for each tree edge marking. After this, we mark the index in P as 1 corresponding to the edge (v_j, v_i) to denote that v_i is the parent of v_j. Thus, marking parent takes $O(degree(v_j))$ time for the vertex v_j. Note that, all of this happens along with the classical stack-based DFS implementation. Thus overall it takes $O(m+n)$ time, and space required to store all these arrays is $O(m+n)$ bits. We refer to the bitvector D as the unary degree sequence array, E as the child array, and P the parent array. These three arrays are stored and used for the query step of our algorithm. Thus, we obtain the following lemma.

▶ **Lemma 7.** *Given an undirected graph G, there exists an $O(m+n)$ time and $O(n \lg n)$ bits preprocessing algorithm to construct the unary degree sequence array, parent and child arrays for G, each of which takes $O(m+n)$ bits of space.*

Step 2: Decomposing the DFS tree by the Tree Covering Technique. The main idea of this step is to perform a decomposition of the DFS tree, and along with storing some crucial informations which will be very useful for navigating the tree during the query step of our algorithm. For this purpose, we use the well-known tree covering technique in the context of succinct representation of rooted ordered trees. The high level idea is to decompose the tree into subtrees called *minitrees*, and further decompose the minitrees into yet smaller subtrees called *microtrees*. The microtrees are small enough to be stored in a compact table. The root

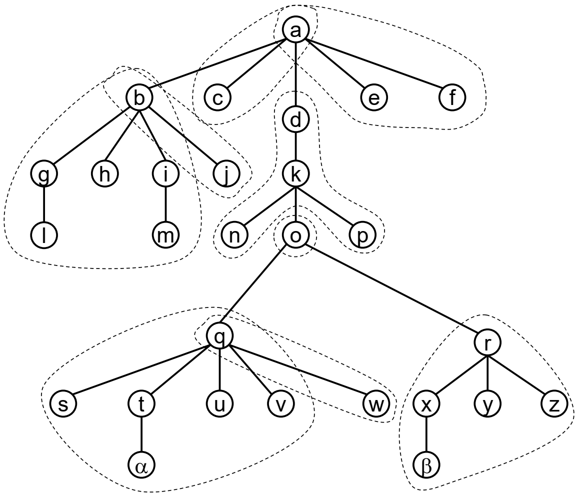


Figure 1 An example of Tree Covering technique with $L = 5$. Each closed region formed by the dotted lines represents a minitree. Here each minitree has at most one "child" minitree (other than the minitrees that share its root).

of a minitree can be shared by several other minitrees. To represent the tree, we only have to represent the connections and links between the subtrees. One such tree decomposition method was given by Farzan and Munro [14] where each minitree has at most one node, other than the root of the minitree, that is connected to the root of another minitree. This guarantees that in each minitree, there exists at most one non-root node which is connected to (the root of) another minitree. We use this decomposition in our algorithms, and the main result of Farzan et al. [14] is summarized in the following theorem:

▶ **Theorem 8** ([14]). *For any parameter $L \geq 1$, a rooted ordered tree with n nodes can be decomposed into $\Theta(n/L)$ minitrees of size at most $2L$ which are pairwise disjoint aside from the minitree roots. Furthermore, aside from edges stemming from the minitree root, there is at most one edge leaving a node of a minitree to its child in another minitree. The decomposition can be performed in linear time using linear words of space.*

See Figure 1 for an illustration. For the purpose of our algorithms, we apply Theorem 8 with $L = \lg n$ on the DFS tree T of G. For this parameter L, since the number of minitrees is only $O(n/\lg n)$, we can represent the structure of the minitrees within the original tree (i.e., how the minitrees are connected with each other) using $O(n)$ bits by simply storing both way pointers (so that we can traverse easily) between the roots of the minitrees. We refer to this as the *skeleton* S of the DFS tree T. See Figure 2 for a demonstration of Figure 1's skeleton. The decomposition algorithm of [14] also ensures that each minitree has at most one "child" minitree (other than the minitrees that share its root) in this structure. We use this property crucially later.

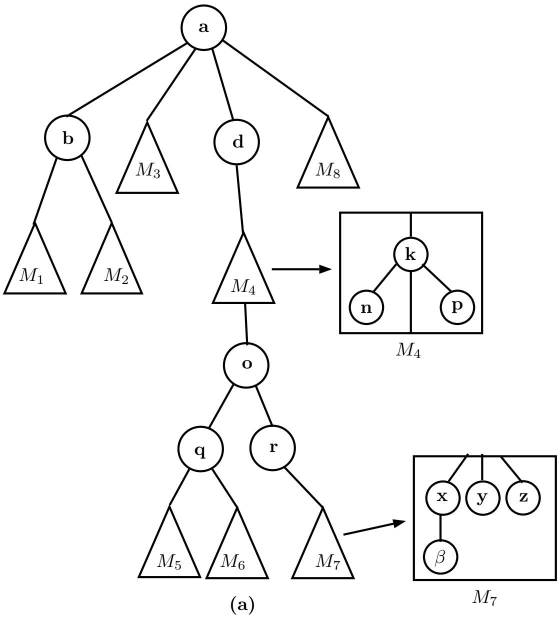


Figure 2 (a) A rough sketch of the skeleton of the tree decomposition shown in Figure 1. In this diagram, the triangles represent the minitrees along with the roots of the minitrees are marked inside the circle. For example, the minitrees M_1 and M_2 share the same root b. Also the node o is a minitree on its own. Strictly speaking, the skeleton will not have the traingles, rather it just contains the pointers between the roots of the minitrees (i.e., circles in this diagram). But we put this diagram for better visual description of the compact representation of the previous diagram.

In what follows, we explain how we compactly represent the minitree structure, and we refer to this compact representation obtained using this tree covering (TC) approach as the TC representation of the DFS tree. Towards this, first observe that every minitree root has unique first child and last child inside the minitree. In some cases, both are the same (see the minitree rooted at node d of Figure 1), and in some cases, both are absent (see the minitree rooted at node o of Figure 1). Thus, if we specify these two quantities, we can uniquely identify the root of the minitree (along with the exact portion of the nodes which are children of the root of this minitree and also belong to the same minitree as the first and last child of the root) even though the root is shared between multiple minitrees. We use this idea crucially in the design of the TC representation of the DFS tree.

We mark in a bitvector R of size n all the nodes which are the last child of a minitree root inside a minitree. Note that, there are $O(n/\lg n)$ such nodes which are marked as 1 in R. In the case of a minitree root not having any children, we mark the minitree root itself as 1 in R. We also build the data structure to support $O(1)$ time *rank/select* queries on R using Theorem 3. Next, we create an array C where each of the $O(n/\lg n)$ entries are $O(\lg n)$ bits long, thus overall it takes $O(n)$ bits. Basically, each entry of C stores some informations regarding the minitree for which the last child of the minitree root is marked 1 in R. More specifically, For a typical node, say v_i, which is the last child of some minitree, we have $R[i] = 1$, and $C[j]$ (where $j = rank_1(R, i)$) comprises of the following six informations (some of which could be empty), (i) label of the minitree root, say v_r, for which v_i is the last child inside the minitree, (ii) location of the first child, say v_j, of v_r inside the minitree in the adjacency array of v_r, (iii) DFI of v_j, (iv) the edge (v_c, v_d) (if any) that goes out of the minitree, (v) the size of the subtree rooted at v_c in the DFS tree, (vi) depth of v_r in the

DFS tree. The tree decomposition method ensures that a minitree has at most one edge (v_c, v_d), where v_c is a non-root node of minitree and v_d is a root of a different minitree, that goes out of the minitree. We also mark in a bitvector Z of size n bits all such vertices like v_c (also note, there could be $O(n/\lg n)$ such vertices). We mark in a bitvector L all the vertices which are the rightmost leaves of every minitree. Note that these vertices (there are, again, $O(n/\lg n)$ of them) have the highest DFI inside the minitree. In another bitvector A, we mark all the roots of the minitrees as 1, and build rank/select structure on top of A. Correspondingly, the F array will store the DFI of the roots so that we can retrieve them in constant time. More specifically, for a minitree root v_r, $A[r] = 1$ and $F[j]$ ($j = rank_1(A, r)$) will store the DFI of v_r. Next we build the $O(1)$ time level ancestor data sturcture, say LA, on the $O(n/\lg n)$ minitree roots (i.e., on the skeleton structure) using [4]. Thus, here, LA takes $O(n)$ bits and $O(n)$ time. As a root of the minitree is shared between multiple minitrees, from each node v_i of the skeleton S (where v_i is a root of a minitree), we store pointers to all the minitrees (in C array) which has v_i as their root. Overall these pointers also consume $O(n)$ bits. This completes the description of the TC representation. Note that, the creation of the skeleton and the TC representation for T can be done in $O(n)$ time using $O(n \lg n)$ bits (using Theorem 8) after the DFS (which takes $O(m + n)$ time and $O(n \lg n)$ bits). Hence, we obtain the following,

▶ **Lemma 9.** *Given an undirected graph G, there exists an $O(m + n)$ time and $O(n \lg n)$ bits preprocessing algorithm to construct the skeleton S and the TC representation of the DFS tree T of G, each of which occupies $O(n)$ bits.*

First observe that, the outputs of the previous step are the unary degree sequence array (D), parent array (E), child array (P), the D_T array, TC representation of T (this includes R, C, Z, A, F and L) along with the skeleton S with pointers to C, and finally the LA structure on S. The arrays D, E, and P take $O(m + n)$ bits, and the others take $O(n)$ bits. Now we show how to efficiently solve the DFS-Indexing problem using these structures.

Query Algorithms. Given v_i, to answer 2(a) in $O(1)$ time, we do the following. If v_i is the root of the DFS tree, we return null. Otherwise, we can compute the answer by using only $select_1$ queries on P and D, as described previously.

To answer 2(b) in $O(1)$ time, we use $select_1$ queries on the bitvector D_T.

To answer 2(c), we first compute the number of children of v_i in T using the query 2(b). Then j-th child is obtained in constant time as described above.

Note that, the queries 2(a), 2(b) and 2(c) can be answered using only D, E, P and D_T arrays. Before explaining the algorithms for the rest of the queries, we first prove the following very crucial lemma.

▶ **Lemma 10.** *Given any query node v_i which is not a root of a minitree, we can reconstruct the minitree M containing v_i in time proportional to the size of M along with the DFIs of all the nodes inside M. In the same amount of time, we can also retrieve the root node of M.*

Proof. First note that if a node v belongs to the minitree M, its children in T also belong to M, except for the following two cases. The first case is that v is the root v_r of M and the second case is that v is v_c of M. In the first case, as we have stored the location (in the adjacency array) of the first child, say v_j, of v_r inside M in the C array, we can enumerate all the children of v_r in M in constant time for each until we hit the rightmost child of v_r in M, which is stored in R. In the second case, we can also enumerate all the children of v_c

in T in constant time for each and discard v_d. For other vertices in M, we can enumerate children using constant time for each output. Note also that going to the parent can be performed in constant time.

The algorithm to achieve the claim can be broken down into three steps. In the first step, given v_i, we launch a DFS starting from v_i, and continue till we retrieve the rightmost leaf, say v_j, of the minitree M. In second step, we follow the path in T (by going to v_j's parent, then its parent and so on) till we reach the rightmost child, say v_k, of the root, say v_r, inside M by using the query algorithm to find the parent repeatedly. In the third and final step, we use the C array, by using v_k, to extract all the informations needed to reconstruct the full minitree by performing another round of DFS. We provide the details below.

To perform the first step, we only need to use the parent and child related queries, whose execution we already showed previously. Note that, as we have stored the information (in Z array) regarding the only edge that goes out of the minitree, we never incorrectly go out of M. Also we can verify if we have reached the unique node v_j which is the rightmost leaf of M from the L array. Once we reach v_j, it's easy to see that v_k has to be an ancestor of v_j (note that v_k and v_j could be same in some cases). Thus, we can reach v_k from v_j by repeatedly using the parent query algorithm, and this completes the second step. Finally, once we reach v_k, we use the informations in $C[j]$ (where $j = rank_1(R, k)$) to retrieve the root v_r of M and other informations. Then we carry out a DFS from v_r by first going to the first child of v_r inside M (retrieved from C), then its first child and so on till we fully reconstruct M. This step also requires repeated invoking of parent and child query only.

In order to retrieve the DFIs of the nodes inside M, observe that, if M doesn't have any child minitree (i.e., no edge is going out of M), then while doing the final DFS from v_r, we can easily compute the DFIs of all the nodes inside M. Otherwise, assume the edge (v_c, v_d) goes out of M where v_c belongs to M, then the DFI of next node inside M can be calculated by adding the size of the subtree rooted at v_c in T (which is stored in C) to the DFI of v_c. It is clear that all of these procedures can be performed in the time proportional to the size of M, which is $O(\lg n)$ here. This completes the description of the proof. ◀

As a corollary of the previous lemma, it is easy to see that the query of 2(d) can be reported in $O(\lg n)$ time for any node v_i which is not a root of the minitree. Otherwise, it can be done in $O(1)$ time by reporting the value stored in $F[j]$ where $j = rank_1(A, i)$.

To answer 1(a), first we invoke query algorithm of 2(d) for both v_i and v_j to retrieve their DFIs respectively, and then answer accordingly. Thus, this also takes $O(\lg n)$ time.

Answering 1(b) involves a few cases. In the first case, if both of them belong to the same minitree then we can figure out the answer by reconstructing the complete minitree. Secondly, suppose v_i and v_j are roots of the two separate minitrees, and their depths in T are x and y respectively (depth can be obtained from C array). Then using these values in LA data structure, we can figure out the required answer. Finally, if both of these nodes belong to two separate minitrees but are not the roots of the minitrees, then first we retrieve the roots of those minitrees using Lemma 10, then follow almost the same procedure as before to figure out the answer. Note that, in this case, it is enough to reconstruct the path from v_c (of the minitree located near to the root) to the root of that minitree (for the case when one of the minitree root is an ancestor of the other minitree root) to figure out the answer of the query. Thus, overall, it takes $O(\lg n)$ time.

To return the query for 3, we do a standard DFS traversal on the skeleton S and each time we visit a new node v_i in S, we follow the pointer from v_i in S to the part of the C array where the informations regarding the minitree rooted at v_i is stored. Note that, v_i might be shared between multiple minitrees, hence, we always start following these pointers from left to right. More specifically, if v_i is the root of p minitrees, we have p pointers emanating from v_i, and going to p different locations of C array. As these pointers are stored from left to right order, which is the same order in DFS of all the minitrees that share the root v_i. Thus we follow the first pointer, and reach the specific portion of C, use Lemma 10 to generate the complete minitree along with the DFIs of the nodes. Then if this minitree has any child minitree, we go on to explore that and so on (by following the (v_c, v_d) edge stored in that minitree). Once we finish all the descendant minitree of the first minitree rooted at v_i, we come back and start exploring the second minitree (by following the second pointer from v_i) and so on. Thus, we need to store these intermediate pointers, in stack, to know how much progress has been made in every node's (in skeleton) list. This procedure is continued until all the nodes of S are exhausted. It is clear that this procedure takes $O(n)$ time as there are $O(n/\lg n)$ nodes in S and for each node, we spend $O(\lg n)$ time. Also, we need $O(n)$ bits (as there could be $O(n/\lg n)$ pointers) of intermediate space for the execution of the DFS.

To answer 4, first note that, in any minitree M, if there is no egde going out (i.e., no (v_c, v_d) type edge), then the DFIs inside M are consecutive, i.e., in general, first child of root inside M has the smallest DFI and the rightmost leaf in M has the maximum DFI, and the numbers are consecutive. Otherwise, DFIs are consecutive from the root of M to the DFI of v_c, then there is a jump of DFI by the size of the subtree rooted at v_c in the DFS tree, then it's consecutive DFI again until the rightmost leaf (which has the largest DFI inside M) of M. Thus, the range of DFIs of the vertices inside any arbitrary minitree M can be broken into at most two disjoint consecutive intervals. We store these (at most $O(n/\lg n)$) intervals in an interval tree along with augmenting it with the last child of M inside M. Now, given i, we first find the interval where i belongs to from the tree and simultaneously retrieve the last child, say v_a, of the corresponding minitree, all using $O(\lg n)$ time. Then, we use the information from R and C array corresponding to v_a to invoke Lemma 10, and retrieve the desired vertex with DFI i using $O(\lg n)$ overall time. This completes the description of the query algorithms for undirected graphs.

We can handle directed graphs similarly except a few changes in the data structures. Recall that, for directed graphs, every vertex v_i has access to its in-neighbors array as well as out-neighbors array, and additionally we create two unary degree sequence arrays (each of size $O(m+n)$ bits), D_1 for the out-neighbors and D_2 for the in-neighbors. We also have two separate arrays, say E_1 (having one-to-one map with D_1), for marking child of every node and E_2 (having one-to-one map with D_2) where parents are marked. It is easy to see that almost in a similar fashion as in the undirected case, we can correctly mark, for any node v_i, the children of v_i in E_1 array and parent of v_i in E_2 array using both the D_1 and D_2 arrays while performing DFS of G. The second preprocessing step doesn't require any changes for the directed graphs. Now reporting queries also can be suitably modified to make use of these changes without affecting the asymptotic running time of the query algorithms. Basically the only change that takes place is as follows, whenever we need to find the parent of a node, now we need to use the in-neighbor array whereas finding children can be handled by consulting out-neighbor array along with the mapping with their respective unary degree sequence array. We omit the details. Thus, we obtain the following,

▶ **Theorem 11.** *Given any undirected or directed graph G, there exists an $O(m+n)$ time and $O(n \lg n)$ bits preprocessing algorithm which outputs a data structure of size $O(m+n)$ bits, using which the queries 1(a), 1(b), 2(d) and 4 can be reported in $O(\lg n)$ time, 2(a) and 2(b) in $O(1)$ time, 2(c) can be answered in time proportional to the number of solutions, and finally 3 can be solved in $O(n)$ time respectively for the* DFS-Indexing *problem.*

Note that, if the given input graph is sparse (i.e., $m = O(n)$), then both unary degree sequence array (D), and parent and child arrays (E, P) take $O(n)$ bits, and every other data structure anyway takes $O(n)$ in total, thus, we obtain the result mentioned in Theorem 1 for the case of sparse graphs. When the input graph is dense (i.e., $m = \omega(n)$), we compress the D, E, P arrays using Theorem 4. Note that we use only $select_1$ queries on compressed arrays and thus query time complexity on the arrays is still constant. Hence, we obtain the result of Theorem 1 for the dense graph case. It is worth mentioning that except the case for very dense graphs, our space bound *always* beats the space bound of the naive algorithm for every edge density in the full spectrum, albeit with super-constant query time. Thus, when the graph is sufficiently dense, it is better to use the standard solution which uses $O(n \lg n)$ bits with constant query time. This completes the description of our algorithms in the indexing model, and hence, the proof of Theorem 1.

4 Algorithms in the Encoding Model

Recall that in the encoding model, we seek to build a data structure *encod* after preprocessing the input graph G such that queries have to be answered using *encod* only, *without* accessing G. To this end, we first provide a lower bound for the space needed for *encod* to answer queries of the DFS-Indexing problem.

4.1 Space lower-bound

Observe that, in order to correctly answer the queries, the data structure *encod* must contain the information regarding the topology of the DFS tree T of the graph G along with the labels of the vertices of T as, unlike the indexing model, we don't have the access to G during the query time. It's easy to see that we need $\Omega(n \lg n)$ bits to store the vertex labels mappings. In what follows, we give a proof for the space needed to store the topology of the DFS tree by counting the number of such trees in any arbitrary graph G.

▶ **Lemma 12.** *For a graph with n vertices and m edges, the size of data structures for storing the topology of the DFS trees is $\Omega\left(n \lg \frac{m}{n}\right)$ bits.*

Proof. Let us consider the following graph G with n vertices and m edges ($m < n^2/2$). It has a vertex r, $k = m/n$ vertices $u_1, \ldots, u_k$, and $n-k-1$ vertices $v_1, \ldots, v_{n-k-1}$. The vertex r is connected to all u_i, and each v_j is also connected to all u_i. To construct a spanning tree of G, we choose one edge among all k edges connected to each v_j. Then the number of different spanning trees of G is at least k^{n-k-1}, and for all different spanning trees the set of DFI's are different. Therefore the size of data structure must be at least $\lg k^{n-k-1}$ bits, which is $\Omega\left(n \lg \frac{m}{n}\right)$. ◀

Thus, the space lower bound for *encod* is $\Omega(max\{n \lg \frac{m}{n}, n \lg n\})$ bits, which is $\Omega(n \lg n)$ bits as mentioned in Theorem 2. In what follows, we complement the above claim by providing a simple indexing structure which asymptotically matches this lower bound.

4.2 Upper-bound

Preprocessing Step. Our index for the DFS-Indexing problem consists of two components which we prepare during the preprocessing step. In the first component, we store, for every vertex v_i, DFI(v_i) as permutation using the structure of Theorem 5 of Section 2. Secondly, we encode the DFS tree succinctly using the structure of Theorem 6 of Section 2.

Query Algorithm. We answer the queries using the two above mentioned structures as follows. To answer 2(d), we just use $\pi(i)$. Similarly, 4 can be answered by invoking $\pi^{-1}(i)$. We report v_i (resp. v_j) as the answer for query 1(a) if $\pi(i) < \pi(j)$ (resp. $\pi(i) > \pi(j)$). We enumerate the vertex ordering as traversed in the DFS order by invoking $\pi^{-1}(1)$, then $\pi^{-1}(2)$, and so on till $\pi^{-1}(n)$. We answer 1(b) in affirmative by checking if $LA(v_j, depth(v_i))$ matches with v_i, otherwise no. To answer 2(a), we return $LA(v_i, depth(v_i) - 1)$. We return the answer of 2(b) by using the query $degree(v_i)$. Finally, we enumerate the children of a node v_i as requested in query 2(c) by using the query $child(v_i, 1)$ till $child(v_i, degree(v_i))$. Hence we obtain the results mentioned in Theorem 2.

5 Conclusion

In this paper, we provided procedures for compactly storing the DFS tree for any graph with efficiently supporting various queries in the indexing and encoding models, and showed how to extend these techniques to design indexing schemes for other fundamental and basic graph problems. With some work, our algorithm can be extended for indexing BFS tree (and other graph search tree also) as well while supporting similar types of queries. Also, as mentioned previously, our results are more general, and can be used in other situations as well.

This work opens up many possible future directions to explore. Can we further improve the query time while keeping the space bound same in the indexing model? Can we prove a space lower bound in the indexing model? Can we design compact data structures for indexing problems like maximum flow? Finally, we conclude by remarking that using [2, 9], we can improve the preprocessing space of our algorithms to $O(n)$ bits (from $O(n \lg n)$ bits) with marginal increment in the preprocessing time.

References

1 H. Acan, S. Chakraborty, S. Jo, and S. R. Satti. Succinct Data Structures for Families of Interval Graphs. In *WADS*, 2019.

2 N. Banerjee, S. Chakraborty, V. Raman, and S. R. Satti. Space Efficient Linear Time Algorithms for BFS, DFS and Applications. *Theory of Computing Systems*, 2018.

3 J. Barbay, L. C. Aleardi, M. He, and J. I. Munro. Succinct Representation of Labeled Graphs. In *18th ISAAC*, pages 316–328, 2007.

4 M. A. Bender and M. Farach-Colton. The Level Ancestor Problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. doi:10.1016/j.tcs.2003.05.002.

5 D. K. Blandford, G. E. Blelloch, and I. A. Kash. Compact representations of separable graphs. In *14th SODA*, pages 679–688, 2003.

6 S. Chakraborty. *Space Efficient Graph Algorithms.* PhD thesis. The Institute of Mathematical Sciences, HBNI, India, 2018.

7 S. Chakraborty, S. Jo, and S. R. Satti. Improved Space-efficient Linear Time Algorithms for Some Classical Graph Problems. *CoRR*, abs/1712.03349, 2017. arXiv:1712.03349.

8 S. Chakraborty, A. Mukherjee, V. Raman, and S. R. Satti. A Framework for In-place Graph Algorithms. In *26th ESA*, pages 13:1–13:16, 2018.

9 S. Chakraborty, V. Raman, and S. R. Satti. Biconnectivity, st-numbering and other applications of DFS using O(n) bits. *J. Comput. Syst. Sci.*, 90:63–79, 2017.

10 S. Chakraborty and S. R. Satti. Space-efficient algorithms for maximum cardinality search, its applications, and variants of BFS. *J. Comb. Optim.*, 37(2):465–481, 2019.

11 D. Clark. *Compact Pat Trees.* PhD thesis. University of Waterloo, Canada, 1996.

12 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.).* MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

13 S. Even and R. E. Tarjan. Computing an *st* -Numbering. *Theor. Comput. Sci.*, 2(3):339–344, 1976.

14 A. Farzan and J. I. Munro. Succinct representation of dynamic trees. *Theor. Comput. Sci.*, 412(24):2668–2678, 2011.

15 A. Farzan and J. I. Munro. A Uniform Paradigm to Succinctly Encode Various Families of Trees. *Algorithmica*, 68(1):16–40, 2014.

16 L. Ferres, J. F. Sepúlveda, T. Gagie, M. He, and G. Navarro. Fast and Compact Planar Embeddings. In *15th WADS*, pages 385–396, 2017.

17 J. E. Hopcroft and R. E. Tarjan. Efficient Planarity Testing. *J. ACM*, 21(4):549–568, 1974.

18 J. I. Munro and P. K. Nicholson. Compressed Representations of Graphs. In *Encyclopedia of Algorithms*, pages 382–386. Springer, 2016. doi:10.1007/978-1-4939-2864-4_646.

19 J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Succinct representations of permutations and functions. *Theor. Comput. Sci.*, 438:74–88, 2012.

20 J. I. Munro and V. Raman. Succinct Representation of Balanced Parentheses and Static Trees. *SIAM J. Comput.*, 31(3):762–776, 2001.

21 G. Navarro. *Compact Data Structures - A Practical Approach*. Cambridge University Press, 2016. URL: http://www.cambridge.org/de/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/compact-data-structures-practical-approach?format=HB.

22 R. E. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

23 R. E. Tarjan. A Note on Finding the Bridges of a Graph. *Information Processing Letters*, 2(6):160–161, 1974.

24 R. E. Tarjan. Finding Dominators in Directed Graphs. *SIAM J. Comput.*, 3(1):62–89, 1974.

25 K. Yamanaka and S. Nakano. A compact encoding of plane triangulations with efficient query supports. *Inf. Process. Lett.*, 110(18-19):803–809, 2010.

Additive Cellular Automata Over Finite Abelian Groups: Topological and Measure Theoretic Properties

Alberto Dennunzio[1]
Dipartimento di Informatica, Sistemistica e Comunicazione,
Università degli Studi di Milano-Bicocca, Viale Sarca 336/14, 20126 Milano, Italy
dennunzio@disco.unimib.it

Enrico Formenti[1]
Universite Côte d'Azur, CNRS, I3S, France
enrico.formenti@univ-cotedazur.fr

Darij Grinberg[1]
School of Mathematics, University of Minnesota, Minneapolis, USA
darijgrinberg@gmail.com

Luciano Margara[1]
Department of Computer Science and Engineering, University of Bologna, Campus of Cesena,
Via dell'Università 50, 47521 Cesena, Italy
luciano.margara@unibo.it

Abstract

We study the dynamical behavior of D-dimensional ($D \geq 1$) additive cellular automata where the alphabet is any finite abelian group. This class of discrete time dynamical systems is a generalization of the systems extensively studied by many authors among which one may list [36, 43, 44, 40, 12, 11]. Our main contribution is the proof that topologically transitive additive cellular automata are ergodic. This result represents a solid bridge between the world of measure theory and that of topology theory and greatly extends previous results obtained in [12, 43] for linear CA over $\mathbb{Z}_m$ i.e. additive CA in which the alphabet is the cyclic group $\mathbb{Z}_m$ and the local rules are linear combinations with coefficients in $\mathbb{Z}_m$. In our scenario, the alphabet is any finite abelian group and the global rule is any additive map. This class of CA strictly contains the class of linear CA over $\mathbb{Z}_m^n$, i.e., with the local rule defined by $n \times n$ matrices with elements in $\mathbb{Z}_m$ which, in turn, strictly contains the class of linear CA over $\mathbb{Z}_m$. In order to further emphasize that finite abelian groups are more expressive than $\mathbb{Z}_m$ we prove that, contrary to what happens in $\mathbb{Z}_m$, there exist additive CA over suitable finite abelian groups which are roots (with arbitrarily large indices) of the shift map.

As a consequence of our results, we have that, for additive CA, ergodic mixing, weak ergodic mixing, ergodicity, topological mixing, weak topological mixing, topological total transitivity and topological transitivity are all equivalent properties. As a corollary, we have that invertible transitive additive CA are isomorphic to Bernoulli shifts. Finally, we provide a first characterization of strong transitivity for additive CA which we suspect it might be true also for the general case.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Cellular Automata, Symbolic Dynamics, Complex Systems

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.68

[1] corresponding author

1 Introduction

Cellular automata (CA) are widely known formal models for studying and simulating complex systems [9]. They are used in many disciplines ranging from physics to biology, stepping through sociology and, of course, computer science (for recent results and an up-to date bibliography on CA, see [25, 28, 17, 16, 3], while for other models of natural computing see for instance [21, 19, 26, 18]). Their extensive use is due to the huge variety of dynamical behaviors. In computer science, applications can be found in many different contexts such as cryptography, pseudo-random number generation, image processing, data compression, *etc.*

More formally, a CA can be defined as an infinite set of finite automata arranged on the regular lattice $\mathbb{Z}^D$, where $D \in \mathbb{N}$ is the *dimension* of the CA. Each finite automaton has a *state* which is chosen from a finite set G, called the *set of states* or the *alphabet*. All finite automata update synchronously according to a *local rule* which takes into account the current state of the automaton and the states of a *neighborhood* $\mathcal{N}$ of neighboring automata. The local rule f induces a *global map* $F \colon G^{\mathbb{Z}^D} \to G^{\mathbb{Z}^D}$ which describes the overall evolution of the CA at each time tick.

Despite of their apparent simplicity, CA may exhibit extremely complex dynamical behaviors. Indeed, in most cases the problem of deciding if a given CA has a certain dynamical property or not is undecidable [5, 30, 37] and some Rice-like theorems have been proved [34, 39]. The complex dynamics of CA has been described through a great number of properties (see Section 2 for definitions) involving both the measure theoretical and the topological point of views. Figure 1 illustrates the relations between those that are studied in this paper.

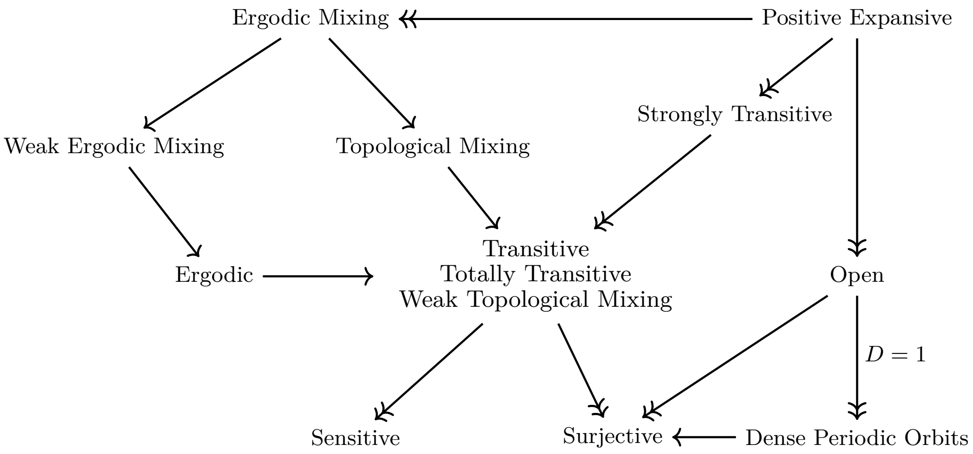


Figure 1 Known relations between dynamical properties of CA. An arrow with single tip indicates that the converse relation is unknown, an arrow with double tip means that the converse relation is false. Labels on arrows indicate that implications have been proved only for specific dimensions. Note that there are no expansive CA in dimension $D > 1$.

Imposing some additional constraints to the global update map allows a complete and efficient description of the dynamical behavior. These properties can take the form of a conservation law [33, 29, 32, 6, 49] or superposition principles induced by an algebraic structure imposed on the alphabet [36, 43, 44, 40, 12, 11] (in both cases the literature is really huge, only a small excerpt is cited here).

Similarly, in this paper, it is required that the alphabet G of the CA is a finite abelian group and its global update map is additive, i.e., an endomorphism of $G^{\mathbb{Z}^D}$. This is a pretty broad requirement which characterizes a class of CA generalizing those with linear local rule defined by $n \times n$ matrices (see the previous citations for the case $n = 1$ and [40, 8] for a generic n). Indeed, the local rule of an additive CA over a group $\langle G, + \rangle$ can be written as

$$f(x_1, \ldots, x_r) = \sum_{i=1}^{r} f_i(x_i)$$

where the functions f_i are endomorphisms of G and $\{x_1, \ldots, x_r\}$ is the neighborhood $\mathcal{N}$.

The fundamental theorem of finite abelian groups states that every finite abelian group is isomorphic to $\bigoplus_{i=1}^{h} \mathbb{Z}_{k_i}$ where the numbers $k_1, \ldots, k_h$ are powers of (not necessarily distinct) primes and $\oplus$ is the direct sum operation. Hence, the global rule F of an additive CA over G splits into the direct sum of a suitable number h' of additive CA over subgroups $G_1, \ldots, G_{h'}$ with $h' \leq h$ and such that $\gcd(|G_i|, |G_j|) = 1$ for each pair of distinct $i, j \in \{1, \ldots, h'\}$. Each of them can be studied separately and then the analysis of the dynamical behavior of F can be carried out by combining together the results obtained for each component.

In order to make things clearer, consider the following example. If F is an additive CA over $G \cong \mathbb{Z}_4 \times \mathbb{Z}_8 \times \mathbb{Z}_3 \times \mathbb{Z}_3 \times \mathbb{Z}_{25}$ then F splits into the direct sum of 3 additive CA over $\mathbb{Z}_4 \times \mathbb{Z}_8$, $\mathbb{Z}_3 \times \mathbb{Z}_3$ and $\mathbb{Z}_{25}$, respectively. Therefore, F will be topologically transitive iff each component is topologically transitive while F will be sensitive to initial conditions iff at least one component is sensitive to initial conditions (see Section 2.1 for the precise definitions of these properties).

The above considerations lead us to three distinct scenarios:

- $\boldsymbol{G \cong \mathbb{Z}_{p^k}}$. Then, G is cyclic and we can define each f_i simply assigning the value of f_i applied to the unique generator of G. Moreover, every pair f_i, f_j commutes, i.e. $f_i \circ f_j = f_j \circ f_i$, and this makes it possible a detailed analysis of the global behavior of F. For additive cellular automata over $\mathbb{Z}_{p^k}$ almost all dynamical properties are well understood and characterized [43].
- $\boldsymbol{G \cong \mathbb{Z}_{p^k}^n}$. In this case, G is not cyclic anymore and has n generators. We can define each f_i assigning the value of f_i for each generator of G. This gives rise to the class of linear CA over $\mathbb{Z}_{p^k}^n$ that have been studied in [20, 40, 8]. Now, f_i and f_j do not commute in general and this makes the analysis of the dynamical behavior much harder. As pointed out in [20], we also recall that linear CA over $\mathbb{Z}_{p^k}^n$ allow the investigation of some classes of non-uniform CA over $\mathbb{Z}_{p^k}$ (for these latter see [22, 10, 24, 23]).
- $\boldsymbol{G \cong \bigoplus_{i=1}^{n} \mathbb{Z}_{p^{k_i}}}$. In this case ($\mathbb{Z}_4 \times \mathbb{Z}_8$ in the example), G is again not cyclic and F turns out to be a subsystem (in the sense of topological dynamics) of a suitable linear CA. In this last case the analysis of the dynamical behavior of F is even more complex than in the previous case. We do not even know easy checkable characterizations of basic properties like surjectivity or injectivity.

Even if the superposition principle still allows us to prove deep and interesting results on the asymptotic behavior of additive CA over finite abelian groups, their dynamics is definitely more interesting and expressive than that of linear CA over $\mathbb{Z}_m$ and exhibits much more complex features. As a first example, consider the set A of graphs with n nodes represented by their adjacency matrices. A can be equipped with a binary operation "+" that makes it a finite (or finitely generated) abelian group G (isomorphic to the group of all $n \times n$ matrices over $\mathbb{Z}_2$ with the "+" operation). The group operation can be defined in many different ways,

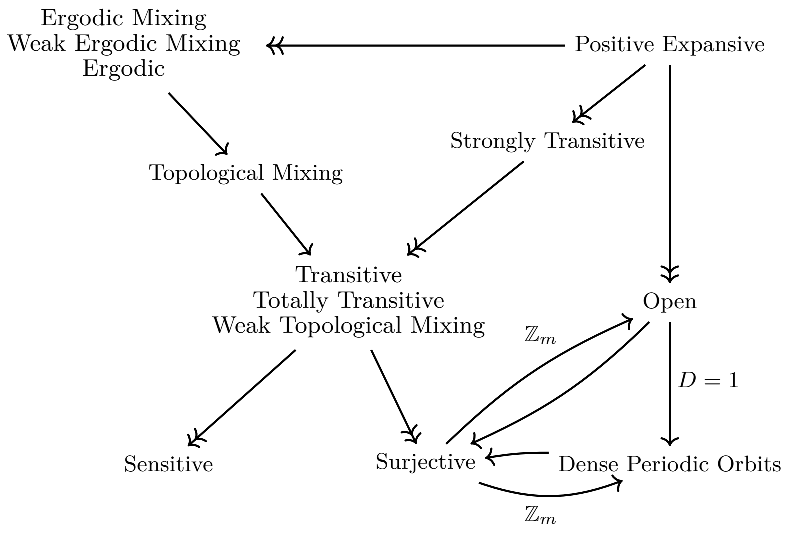


Figure 2 Known relations among dynamical properties of additive CA before the present paper. An arrow with single tip indicates that the converse relation is unknown, an arrow with double tip means that the converse relation is false. Labels on arrows indicate that implications have been proved only for specific alphabets or dimensions. Note that there are no expansive CA in dimension $D > 1$.

e.g. sum of matrices or, for undirected graphs, product of (symmetric) matrices. The local rule of the CA can be any map preserving the group structure. It is easy to verify that the dynamics of this kind of automata cannot be simulated by any linear CA over $\mathbb{Z}_m$.

This richness in terms of expressive power is further stressed by Theorem 9 which shows how the group structure of $\mathbb{Z}_{p^k}$ constraints the dynamics. However, there exist additive one-dimensional CA over suitable abelian groups that are arbitrarily "small" roots of the shift map, as illustrated in Example 10. This means that the group structure helps out in constructing CA which are able of transmitting the information (encoded in the initial configuration) at arbitrary slow speed. In particular, this allows the construction of ergodic maps with arbitrary low Lyapunov exponents. Theorem 8 tells that the same cannot be done by one-dimensional CA over alphabets of prime cardinality.

Figure 2 illustrates the known relations among dynamical properties of linear CA before the present paper. Figure 3 illustrates the impact of the results of the present paper. As a matter of fact, the overall picture have been greatly simplified and the dynamics much better understood.

The paper is structured as follows. Section 2 introduces all the necessary background and formal definitions. Section 3 states the main contributions of the paper and the next one provides all the proofs. In the last section we draw our conclusions and provide some perspectives.

2 Background

We begin by reviewing some general notions and introducing notations we will use throughout the paper.

Let $\mathbb{Z}$ and $\mathbb{N}$ be the set of integers and natural numbers, respectively. For any $v \in \mathbb{Z}^D$ we denote $||v|| = ||(v_1, \ldots, v_D)|| = \max\{|v_1|, \ldots, |v_D|\}$.

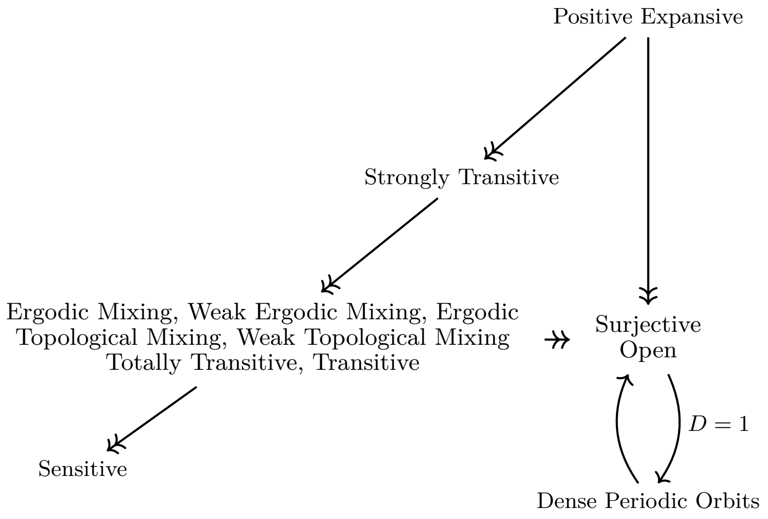

Positive Expansive
Strongly Transitive
Ergodic Mixing, Weak Ergodic Mixing, Ergodic Topological Mixing, Weak Topological Mixing Totally Transitive, Transitive
Surjective
Open
$D = 1$
Sensitive
Dense Periodic Orbits

Figure 3 Relations among dynamical properties of additive CA taking into account the results of the present paper. An arrow with single tip indicates that the converse relation is unknown, an arrow with double tip means that the converse relation is false. Labels on arrows indicate that implications have been proved only for specific dimensions. Note that there are no expansive CA in dimension $D > 1$.

Let G be a finite alphabet. A **configuration** over G is a map from $\mathbb{Z}^D$ to G. For any configuration $c \in G^{\mathbb{Z}^D}$ and any vector $v \in \mathbb{Z}^D$, $c(v)$ is the value of c in position v. The configuration space $G^{\mathbb{Z}^D}$ is equipped with the distance d defined as follows

$$\forall c, c' \in G^{\mathbb{Z}^D}, \quad d(c, c') = \begin{cases} 0, & \text{if} \quad c = c', \\ 2^{-\min\{||\boldsymbol{v}|| \,:\, c(v) \neq c'(v)\}}, & \text{otherwise} \end{cases} .$$

In this way, the set $G^{\mathbb{Z}^D}$ equipped with the topology induced by d turns out to be a compact, perfect, and totally disconnected topological space (i.e., a Cantor space). In the sequel, the configuration space $G^{\mathbb{Z}^D}$ will be sometimes denoted by X.

A **pattern** P is a function from $\{-\ell, \ldots, \ell\}^D$ to G, for some $\ell \in \mathbb{N}$. For any $\ell \in \mathbb{N}$, denote by $\mathcal{P}_\ell$ the set of all patterns with domain $\{-\ell, \ldots, \ell\}^D$. For any $P \in \mathcal{P}_\ell$, the **cylinder** individuated by the pattern P is the set $[P] = \{c \in G^{\mathbb{Z}^D} | \forall v \in \{-\ell, \ldots, \ell\}^D, c(v) = P(v)\}$. Cylinders are clopen sets and the set $\{[P] : \ell \in \mathbb{N}, P \in \mathcal{P}_\ell\}$ of all cylinders is a basis for the topology induced by d.

For some fixed integer $s \geq 1$, let f (named, s-sized *local rule*) and $\mathcal{N}$ (s-sized *neighborhood frame*) be any map from G^s to G and an ordered set of distinct vectors $u_1, \ldots, u_s$, respectively. A ***D*-dimensional CA** over G is a pair $(G^{\mathbb{Z}^D}, F)$, where $F \colon G^{\mathbb{Z}^D} \to G^{\mathbb{Z}^D}$ is the function (named, *global transition map*) defined on the basis of f and $\mathcal{N}$ as follows

$$\forall c \in G^{\mathbb{Z}^D}, \forall v \in \mathbb{Z}^D, \quad F(c)(v) = f\left(c(v + u_1), \ldots, c(v + u_s)\right) \quad . \tag{1}$$

Recall that F is a uniformly continuous map w.r.t. the metric d and any function $F : G^{\mathbb{Z}^D} \to G^{\mathbb{Z}^D}$ is the global transition map of a D-dimensional CA iff it is uniformly continuous and shift-commuting (Hedlund's theorem from [35]), i.e., $F \circ \sigma^u = \sigma^u \circ F$ for any $u \in \mathbb{Z}^D$, where $\sigma^u : G^{\mathbb{Z}^D} \to G^{\mathbb{Z}^D}$ is the D-dimensional **shift map** defined by $\forall c \in G^{\mathbb{Z}^D}, \forall v \in \mathbb{Z}^D, \sigma^u(c)(v) = c(v + u)$. From now on, for the sake of simplicity, we identify a CA with its global map. Moreover, we will denote σ^1 simply by σ.

In the sequel, the alphabet G will be an **abelian group**, with group operation $+$, neutral element 0, and inverse operation $-$. In this way, both the configuration space $G^{\mathbb{Z}^D}$ and the set $\mathcal{P}_\ell$ turn out to be abelian groups, too, where the group operation of $G^{\mathbb{Z}^D}$ and $\mathcal{P}_\ell$ is the component-wise extension of $+$ to $G^{\mathbb{Z}^D}$ and $\mathcal{P}_\ell$. With an abuse of notation, we denote by the same symbols $+$, 0, and $-$ the group operation, the neutral element, and the inverse operation, respectively, both of $\mathbb{Z}^D$, G, $G^{\mathbb{Z}^D}$, and $\mathcal{P}_\ell$. Observe that $+$ and $-$ are continuous functions in the topology induced by cylinders. Hence, $G^{\mathbb{Z}^D}$ is a compact topological group.

A configuration $c \in G^{\mathbb{Z}^D}$ is said to be finite if the number of positions $v \in \mathbb{Z}^D$ with $c(v) \neq 0$ is finite. A CA $(G^{\mathbb{Z}^D}, F)$ over G is **additive** if

$$\forall c, c' \in G^{\mathbb{Z}^D}, \quad F(c+c') = F(c) + F(c') \ .$$

In other words, additive D-dimensional CA over G are endomorphisms of $G^{\mathbb{Z}^D}$.

The **sum of two additive CA** F_1 and F_2 over G is naturally defined as the map on $G^{\mathbb{Z}^D}$ denoted by $F_1 + F_2$ and such that

$$\forall c \in G^{\mathbb{Z}^D}, \quad (F_1 + F_2)(c) = F_1(c) + F_2(c) \ .$$

2.1 Topological and measure theoretic properties

We now recall the definition of some topological and measure theoretical properties for general systems.

A **discrete time dynamical system (DTDS)** is a pair $(\mathcal{X}, \mathcal{F})$, where $\mathcal{X}$ is any set equipped with a distance d and $\mathcal{F}$ is a transformation on $\mathcal{X}$ which is continuous with respect to d. Clearly, CA are DTDS. A DTDS $(\mathcal{X}, \mathcal{F})$ is **surjective**, resp., **open**, if $\mathcal{F}$ is surjective, resp., open. Open CA are surjective (for a proof see [25], for instance). Moreover, any open one-dimensional CA F is characterized by the following property: there exists a natural $k > 0$ such that for every configuration $c \in G^{\mathbb{Z}^D}$ it holds that $|F^{-1}(c)| = k$. Two DTDS $(\mathcal{X}, \mathcal{F})$ and $(\mathcal{X}', \mathcal{F}')$ are isomorphic if there exists an homeomorphism $\varphi : \mathcal{X} \to \mathcal{X}'$ such that $\varphi \circ \mathcal{F} = \mathcal{F}' \circ \varphi$.

A DTDS $(\mathcal{X}, \mathcal{F})$ is **topologically transitive** (or, simply, transitive) if for all nonempty open subsets U and V of $\mathcal{X}$ there exists a natural number t such that $\mathcal{F}^t(U) \cap V \neq \emptyset$, while it is said to be **topologically mixing** if for all nonempty open subsets U and V of $\mathcal{X}$ there exists a natural number t_0 such that the previous intersection condition holds for every $t \geq t_0$. Intuitively, a topologically transitive system $(\mathcal{X}, \mathcal{F})$ has elements of $\mathcal{X}$ which eventually move under iteration of $\mathcal{F}$ from one arbitrarily small neighbourhood to any other. As a consequence, the dynamical system cannot be decomposed into two disjoint open sets which are invariant under the map $\mathcal{F}$. Clearly, topological mixing is a stronger condition than transitivity. Further, $(\mathcal{X}, \mathcal{F})$ is **topologically weakly mixing** if the DTDS $(\mathcal{X} \times \mathcal{X}, \mathcal{F} \times \mathcal{F})$ is topologically transitive, while it is **totally transitive** if $(\mathcal{X}, \mathcal{F}^t)$ is topologically transitive for all $t \in \mathbb{N}$. We now recall another condition stronger than transitivity. A DTDS is **strongly transitive** if for any nonempty open subset U of $\mathcal{X}$ it holds that $\bigcup_{t=1}^{\infty} \mathcal{F}^t(U) = \mathcal{X}$.

A DTDS $(\mathcal{X}, \mathcal{F})$ is **sensitive to initial conditions** if there exists $\epsilon > 0$ such that for any $\delta > 0$ and $x \in \mathcal{X}$, there are an element $y \neq x$ with $d(y, x) < \delta$ and a natural number t such that $d(\mathcal{F}^t(y), \mathcal{F}^t(x)) > \epsilon$. Roughly speaking, $(\mathcal{X}, \mathcal{F})$ is sensitive to initial conditions, or simply sensitive, if there exist elements arbitrarily close to x which eventually separate from x by at least ϵ under iteration of $\mathcal{F}$. If a DTDS is sensitive, then, for all practical purposes, the dynamics eventually defy numerical approximation. Small errors in computation which are introduced by round-off may become magnified upon iteration. The results of numerical computation of an orbit, no matter how accurate, may be completely different from the real orbit. In [13] it has been proven that, for CA, topological transitivity implies sensitivity. Thus, for CA, the notion of topological transitivity becomes central to chaos theory. A DTDS $(\mathcal{X}, \mathcal{F})$ is said to be **positively expansive** if there exists $\epsilon > 0$ such that for any pair of elements $x, y \in \mathcal{X}$ with $x \neq y$ there exists a natural number t such that $d(\mathcal{F}^t(y), \mathcal{F}^t(x)) > \epsilon$. If $\mathcal{X}$ has infinite cardinality then expansivity is a stronger condition than sensitivity (see [31] for a study concerning expansivity and sensitivity in CA). While there are no positively expansive D-dimensional CA when $D \geq 2$, in dimension 1, expansivity implies both mixing, strong transitivity, and openness [41].

An element $x \in \mathcal{X}$ is a periodic point for a DTDS $(\mathcal{X}, \mathcal{F})$ if $\mathcal{F}^t(x) = x$ for some integer $t > 0$. A DTDS $(\mathcal{X}, \mathcal{F})$ has **dense periodic orbits** (DPO) if the set of its periodic points is dense in $\mathcal{X}$. The popular book by Devaney [27] isolates three components as being the essential features of chaos for DTDS: topological transitivity, sensitivity to initial conditions and denseness of periodic orbits (see [27, Def. 8.5]).

Let $(\mathcal{X}, \mathcal{M}, \mu)$ be a probability space and let $(\mathcal{X}, \mathcal{F})$ be a DTDS where $\mathcal{F}$ is a measurable map which preserves μ, i.e., $\mu(E) = \mu(\mathcal{F}^{-1}(E))$ for every $E \in \mathcal{M}$. The DTDS $(\mathcal{X}, \mathcal{F})$, or, the map $\mathcal{F}$, is **ergodic** with respect to μ if for every $E \in \mathcal{M}$

$$\left(E = \mathcal{F}^{-1}(E)\right) \Rightarrow (\mu(E) = 0 \text{ or } \mu(E) = 1) .$$

It is well known that $\mathcal{F}$ is ergodic iff for any pair of sets $A, B \in \mathcal{M}$ it holds that

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \mu(\mathcal{F}^{-i}(A) \cap B) = \mu(A)\mu(B)$$

The DTDS $(\mathcal{X}, \mathcal{F})$ is **(ergodic) mixing**, if for any pair of sets $A, B \in \mathcal{M}$ it holds that

$$\lim_{n\to\infty} \mu(\mathcal{F}^{-n}(A) \cap B) = \mu(A)\mu(B) \quad ,$$

while it is **(ergodic) weak mixing**, if for any pair of sets $A, B \in \mathcal{M}$ it holds that

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} |\mu(\mathcal{F}^{-i}(A) \cap B) - \mu(A)\mu(B)| = 0$$

In order to apply ergodic theory to CA, we need to define the collection $\mathcal{M}$ of measurable subsets of $G^{\mathbb{Z}^D}$ and a probability measure $\mu : \mathcal{M} \to [0, 1]$. We will use the normalized *Haar* measure μ_H defined over the σ-algebra generated by the cylinders which is, to our knowledge, one of the most widely used probabilistic setting in CA theory. The measure μ_H is defined as the product measure induced by the uniform probability distribution over G. In this way, for any $\ell \in \mathbb{N}$ and any pattern $P \in \mathcal{P}_\ell$, it holds that $\mu_H([P]) = \frac{1}{|G|^{(2\ell+1)^D}}$. Since in the rest of this paper we will only use the Haar measure, then we will write μ instead of μ_H.

3 Statement of the main results

In this section, we state the main results of this paper. They allow us to simplify the relationships between the dynamical and measure theoretic properties of additive CA, as depicted in Figure 3.

The following is the main result of the paper. It builds a bridge between two pretty different ways of approaching the study of the dynamics of CA: measure-theoretic and topological. The arguments used in the proofs are closely crafted on the additivity property of the global rule and on the group structure of the alphabet, however, the overall impression is that this tight link between topology and measure theory shall be true in a much more general setting.

▶ **Theorem 1.** *Let F be an additive D-dimensional CA over a finite abelian group. If F is topologically transitive then F is ergodic.*

Theorem 2 provides a new facet of the ergodicity property. This time ergodicity is related to set theoretic properties like surjectivity and aperiodicity of finite configurations.

▶ **Theorem 2.** *Any additive D-dimensional CA F over a finite abelian group is ergodic if and only if F is surjective and no finite configuration except 0 is a periodic point for F.*

The following corollary collects all the known properties which are related to ergodicity in the context of additive CA over finite abelian groups.

▶ **Corollary 3.** *Let F be an additive D-dimensional CA over a finite abelian group. The following properties are equivalent*

1. *ergodic mixing;*
2. *weak ergodic mixing;*
3. *ergodicity;*
4. *topological mixing;*
5. *total transitivity;*
6. *weak topological mixing;*
7. *topological transitivity;*
8. *F is surjective and no finite configuration except 0 is a periodic point of F.*

▶ **Corollary 4.** *Let F be an additive D-dimensional CA over a finite abelian group. If F is invertible and transitive then F is isomorphic to a Bernoulli shift.*

Surjectivity has strong implications on the dynamics of general CA, the following proposition and its corollary prove that in the context of additive CA, those implications are even stronger.

▶ **Proposition 5.** *Let F be an additive D-dimensional CA over a finite abelian group. If F is surjective then it is open.*

▶ **Corollary 6.** *Surjectivity and openness are equivalent properties for additive D-dimensional CA over a finite abelian group. Furthermore, they are equivalent to DPO in dimension $D = 1$.*

The following theorem provides a first characterization of strong transitivity for additive CA over finite abelian groups. Roughly speaking, the theorem states that this property is conserved under translations and iterations. We wonder whether the same holds for general CA.

▶ **Theorem 7.** *Let F be and additive D-dimensional CA over a finite abelian group. The following conditions are equivalent:*

1. *F is strongly transitive;*
2. *for every $v \in \mathbb{Z}^D$, the CA $\sigma^v \circ F$ is strongly transitive;*
3. *for every $n \in \mathbb{N}$, the CA F^n is strongly transitive;*

In the context of 1-dimensional CA, the following result characterizes the *roots* of powers of the shift map. Recall that a CA F is a root of another CA F' if there exists an integer $n > 0$ such that $F^n = F'$.

▶ **Theorem 8.** *[35, Thm. 18.1] Let F be a 1-dimensional CA over an alphabet G of prime cardinality. If $F^n = \sigma^m$ for some naturals n, m with $n \geq 1$, then $n|m$.*

In the case of linear CA over an alphabet of cardinality which is a power of a prime, the following weaker form of Theorem 8 can be proved.

▶ **Theorem 9.** *Let $G = \mathbb{Z}_{p^k}$ with p prime and let F be a 1-dimensional CA over G defined by the neighborhood $\mathcal{N} = \{-r, \ldots, r\}$ and the local rule $f : \mathbb{Z}_{p^k}^{2r+1} \to \mathbb{Z}_{p^k}$ expressed by the linear combination with coefficients $a_{-r}, \ldots, a_r \in \mathbb{Z}_{p^k}$. If $F^n = \sigma^m$ for some naturals $n \geq 1$ and $m \geq 1$, then $m \geq n$.*

The following example shows that Theorem 9 is no longer true for additive CA over finite abelian groups.

▶ **Example 10.** Let F be the 1-dimensional CA over $\mathbb{Z}_m^n$ defined by the neighbourhood $\mathcal{N} = \{0, 1\}$ and the local rule $f : (\mathbb{Z}_m^n)^2 \to \mathbb{Z}_m^n$ such that

$$\forall (x_0, x_1) \in (\mathbb{Z}_m^n)^2, \quad f(x_0, x_1) = M_0 x_0 + M_1 x_1 \ ,$$

where

$$M_0 = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \quad \text{and} \quad M_1 = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix} .$$

It is easy to check that $F^n = \sigma$ contradicting the statement of Theorem 9.

Example 10 has also other important consequences. On the one hand, it provides examples of CA that are arbitrarily "small" roots of the shift map. On the other hand, it provides the basic building blocks for systems which are ergodic but have both arbitrarily low Lyapunov exponents and arbitrarily low topological entropies.

4 Proofs of the main results

4.1 Useful known results

Before going into the proofs of our main results let us recall some known facts which helped in shaping the situation depicted in Figure 2.

▶ **Theorem 11.** *[50, Thm. 1.28, pag. 50] Mixing, weak mixing and ergodicity are equivalent properties for endomorphisms of compact groups.*

▶ **Theorem 12.** *[4, Thm. 1] An ergodic automorphism of a compact metric abelian group is a Bernoulli shift.*

▶ **Proposition 13.** *[15, Prop. 6.7, pag. 32] Ergodic (resp., weak ergodic) mixing implies topological (resp., weak) mixing for endomorphisms of compact groups and measures with full support.*

The following will be fundamental for proving our main result.

▶ **Theorem 14.** *[48, Thm. 1] Let F be any endomorphism of a compact abelian group with normalized Haar measure. Then, F is ergodic if and only if F is surjective and $F^n - I$ is surjective for all $n \in \mathbb{N}$ (I is the identity map).*

4.2 Proofs of our results

In order to make the proof of the main result more readable, we cut it into several lemmata.

▶ **Lemma 15.** *Let F be an additive D-dimensional CA over a finite abelian group G. For all $n \geq 1$, it holds that*

$$F^n - I = (F - I) \circ (I + F + \cdots + F^{n-1}) = (I + F + \cdots + F^{n-1}) \circ (F - I) \tag{2}$$

Proof. It is an immediate consequence of the fact that the composition operation on the set of additive CA is distributive over the sum operation (defined on the same set). ◀

▶ **Lemma 16.** *Let F be an additive CA over a finite abelian group G. Let $\ell \in \mathbb{N}$ and $w \in \mathcal{P}_l$ such that $(F - I)(X) \cap [w] = \emptyset$. Then, it holds that*

$$\forall n \geq 1, \quad (F^n - I)(X) \cap [w] = \emptyset \ .$$

Proof. For every natural $n \geq 1$, by Lemma 15, we get

$$(F^n - I)(X) = (F - I)\left((I + F + \cdots + F^{n-1})(X)\right) \subseteq (F - I)(X) \ .$$

Since $(F - I)(X) \cap [w] = \emptyset$, it follows that $(F^n - I)(X) \cap [w] = \emptyset$. ◀

▶ **Lemma 17.** *Let F be an additive D-dimensional CA over a finite abelian group G. If there exist $\ell \in \mathbb{N}$ and $w \in \mathcal{P}_l$ such that*

$$\forall n \geq 1, \quad (F^n - I)(X) \cap [w] = \emptyset \ ,$$

then F is not topologically transitive.

Proof. For the sake of argument, assume that F is topologically transitive. Choose arbitrarily two patterns $w_1, w_2 \in \mathcal{P}_l$ such that $w_2 - w_1 = w$. By transitivity, there exist a configuration $c \in [w_1]$ and a natural $n \geq 1$ such that $F^n(c) \in [w_2]$. Thus, $F^n(c) - c \in [w]$, or, equivalently, $(F^n - I)(c) \in [w]$, which is a contradiction. ◀

At present all the necessary pieces have been built to go through the proof of Theorem 1.

Proof of Theorem 1. For the sake of argument, assume that F is transitive but not ergodic. Since transitive CA are surjective, then, by Theorem 14, there exists $n \geq 1$ such that $F^n - I$ is not surjective. Let $H = F^n$. Since $H - I$ is not surjective and $(H - I)(X)$ is closed, there exist a natural $\ell \in \mathbb{N}$ and a pattern $w \in \mathcal{P}_\ell$ such that $(H - I)(X) \cap [w] = \emptyset$. So, by Lemma 16, it follows that

$$\forall m \geq 1, \quad (H^m - I)(X) \cap [w] = \emptyset \ .$$

Therefore, by Lemma 17, H is not topologically transitive. Since topologically transitive CA are totally transitive (see [46], where the proof involving 1-dimensional CA can be extended to any dimension D), we conclude that neither is F, which is a contradiction. ◀

Proof of Theorem 2. By Theorem 14, F is ergodic if and only if it is surjective and for every natural $m \geq 1$ the CA $I - F^m$ is surjective. Fix an arbitrary natural $m \geq 1$ and define $H^{(m)} = I - F^m$. The Garden-of-Eden Theorem for CA [45, 47] guarantees that $H^{(m)}$ is surjective if and only if it is injective on finite configurations, i.e., $H^{(m)}(c) \neq H^{(m)}(c')$ for every pair of distinct finite configurations $c, c' \in G^{\mathbb{Z}^D}$. Set $d = c - c' \in G^{\mathbb{Z}^D}$. Clearly d is a finite configuration. Furthermore, by additivity, it holds that $H^{(m)}(c) \neq H^{(m)}(c')$ if and only if $H^{(m)}(d) \neq 0$. By definition of $H^{(m)}$, the condition $H^{(m)}(d) \neq 0$ is true for every $m \geq 1$ if and only if $F^m(d) \neq d$, i.e., d is not a periodic point of F. ◀

Proof of Proposition 5. We are going to prove that F is open at the configuration $0 \in G^{\mathbb{Z}^D}$, i.e., equivalently, for every $\ell \in \mathbb{N}$, the set $F([0])$ is open, where $0 \in \mathcal{P}_\ell$. Since F is an endomorphism of the topological group $G^{\mathbb{Z}^D}$, we conclude that F is open.

To proceed, consider any arbitrary $\ell \in \mathbb{N}$. Clearly, it holds that $G^{\mathbb{Z}^D} = \bigcup_{w \in \mathcal{P}_\ell}[w]$ and, since F is surjective, we get $G^{\mathbb{Z}^D} = \bigcup_{w \in \mathcal{P}_\ell} F([w])$. Thus, there exists $w' \in \mathcal{P}_\ell$ such that $F([w'])$ has non-empty interior. Let $c \in [w']$ be the configuration such that $c(v) = 0 \in G$ for every position $v \in \mathbb{Z}^D$ with $||v|| > \ell$. One gets $F([w']) = F([0] + c) = F([0]) + F(c)$ (where $[0] + c$ and $F([0]) + F(c)$ denote suitable cosets of $[0]$ and $F([0])$, respectively). Therefore, $F([0])$ must have non-empty interior, too. Since $F([0])$ is a subgroup of the topological group $G^{\mathbb{Z}^D}$, it follows that $F([0])$ is open. ◀

Proof of Corollary 6. It is well-known that any open D-dimensional CA is surjective. By Proposition 5, surjective additive D-dimensional CA over a finite abelian group are open. It is easy to see that DPO implies surjectivity in any dimension. While, by Proposition 5 and the fact that open 1-dimensional CA have DPO [7, Thm. 4.4], it follows that surjective additive 1-dimensional CA have DPO. ◀

Proof of Corollary 4. By Theorem 1, F is ergodic. The thesis follows from Theorem 12. ◀

▶ **Lemma 18.** *An additive CA F over a finite abelian group G is strongly transitive if and only if the following condition holds: for any natural $\ell \in \mathbb{N}$ and any pattern $P \in \mathcal{P}_\ell$ there exists $t \in \mathbb{N}$ such that $0 \in F^t([P])$.*

Proof. The "only if" part trivially follows from the definition of strong transitivity. Let us prove the "if" part. Assume that F is an additive CA over a finite abelian group G and satisfying the condition in the statement. Consider an arbitrary natural $\ell \in \mathbb{N}$. For every $P \in \mathcal{P}_\ell$ let $n_{P,\ell}$ be the smallest natural t such that $0 \in F^t([P])$. Define $n_\ell = \max\{n_{P,\ell} : P \in \mathcal{P}_\ell\}$. Since $F(0) = 0$ it holds that

$$\forall P \in \mathcal{P}_\ell, \quad 0 \in F^{n_\ell}([P]) \tag{3}$$

We now show that for any configuration $e \in G^{\mathbb{Z}^D}$, any natural ℓ, and any pattern $P \in \mathcal{P}_\ell$ there exists $t' = n_\ell$ such that $e \in F^{t'}([P])$, that is, F is strongly transitive. Choose arbitrarily a configuration $e \in G^{\mathbb{Z}^D}$, a natural ℓ, and pattern $P \in \mathcal{P}_\ell$. Let c be any configuration belonging to $F^{-n_\ell}(e)$. If $c \in [P]$ we are done. Otherwise, by (3), there exists $c' \in G^{\mathbb{Z}^D}$ such that $c + c' \in [P]$ and $F^{n_\ell}(c') = 0$. Thus, we get

$$F^{n_\ell}(c + c') = F^{n_\ell}(c) + F^{n_\ell}(c') = F^{n_\ell}(c) + 0 = F^{n_\ell}(c) = e \ ,$$

and this concludes the proof. ◀

Proof of Theorem 7. It is an immediate consequence of Lemma 18 and the fact that for every $v \in \mathbb{Z}^D$ and every $n \in \mathbb{N}$ both $\sigma^v(0) = 0$ and $F^n(0) = 0$ hold. ◀

Proof of Theorem 9. The CA F can be represented by the Laurent polynomial $\mathbb{p}(x, x^{-1}) = \sum_{i=-r}^{r} a_i x^i \in \mathbb{Z}_{p^k}[x, x^{-1}]$, while σ^1 can be represented by the Laurent polynomial x. Assume that $F^n = \sigma^m$ for some naturals n, m with $n \geq 1$ and $m \geq 1$. It is easy to verify that $F^n = \sigma^m$ if and only if $\left(\mathbb{p}(x, x^{-1})\right)^n = x^m$. We consider two cases. 1) If $p | a_i$ for each $i \in \mathcal{N}$ with $i \neq 0$ then, by [14, Lemma 5], putting $h = p^{k-1} \in \mathbb{N}$ we have that $\left(\mathbb{p}(x, x^{-1})\right)^h$ is a constant (i.e., it does not contain the formal variable x). Hence, for every $s \in \mathbb{N}$ it holds that $\left(\mathbb{p}(x, x^{-1})\right)^{sh} \neq x^{mh}$ and this contradicts that $F^n = \sigma^m$. 2) Otherwise, there exists $i \neq 0$ such that $\gcd(a_i, p) = 1$ (since $F^n = \sigma^m$ then F must be injective and so for every other coefficient a_j with $i \neq j$ we have $\gcd(a_j, p) = p$). First of all, we prove that there always exist n' and m' with $m' \geq n'$ such that $F^{n'} = \sigma^{m'}$. Indeed, by [14, Lemma 5], for $h = p^{k-1} \in \mathbb{N}$ we get $\left(\mathbb{p}(x, x^{-1})\right)^h = a_i^h x^{ih}$. Therefore, for $t = \varphi(p^k)$ (where φ is the Euler's totient function) we get $\left(\mathbb{p}(x, x^{-1})\right)^{n'} = x^{m'}$, i.e., $F^{n'} = \sigma^{m'}$, with $n' = ht$ and $m' = iht$. To conclude, assume by contradiction that $F^n = \sigma^m$ with $m < n$. Then, we get $F^{n'n} = \sigma^{m'n}$ and $F^{n'n} = \sigma^{n'm}$. So, it follows that $m'n = n'm$, but this is not possible since $m'n \geq n'n > n'm$. Thus, it holds that $m \geq n$ and this concludes the proof. ◀

5 Conclusions and perspectives

Comparing Figure 2 and 3, one can immediately appreciate the impact of the results in the paper. All single arrow tips (i.e. the known relations for which the opposite implication was unknown) disappeared and several properties coalesced in the group of transitivity, total transitivity and weak topological mixing. However, what is more important is that many measure theoretical and topological properties coincide. These facts legitimate the following

▶ **Conjecture 1.** *Transitive CA are ergodic with respect to Bernoulli measures.*

Solving the previous conjecture will probably clarify also the status of the properties of ergodic mixing, weakly ergodic mixing and topological mixing, much like it happened for the case of endomorphisms of compact abelian groups in this paper. Investigating, the following well-known conjecture due to Blanchard and Tisseur (see [2, Conjecture 1] and [1] for more details) will definitively complete the overall picture.

▶ **Conjecture 2.** *All surjective CA have a dense set of periodic orbits.*

Another interesting research direction consists in establishing the decidability of the dynamical properties. In the framework of general CA, recent results from Ville Lukkarila have shown that both topological mixing and transitivity are undecidable properties [42]. Undecidability of sensitivity, surjectivity (for dimensions larger than 1) and openness (for dimensions larger than 1) were already known for years [38, 30]. It is therefore natural to conjecture that the remaining ones are also undecidable.

▶ **Conjecture 3.** *Ergodicity, weak ergodic mixing, ergodic mixing and strong transitivity are undecidable for CA.*

We want to remark that we intentionally left out the expansivity property from Conjecture 3, since it is so peculiar that we believe it might be decidable.

Finally, it would be very interesting to extend Theorem 7 to all CA, and not only to additive CA. In some way, this would turn strong transitivity into the "strongest" translation invariant property, since it is well-known that expansivity is not translations invariant and that there are no expansive CA for dimensions greater than one.

References

1 Luigi Acerbi, Alberto Dennunzio, and Enrico Formenti. Shifting and Lifting of Cellular Automata. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *Computation and Logic in the Real World, Third Conference on Computability in Europe, CiE 2007, Siena, Italy, June 18-23, 2007, Proceedings*, volume 4497 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2007. doi:10.1007/978-3-540-73001-9.

2 Luigi Acerbi, Alberto Dennunzio, and Enrico Formenti. Conservation of some dynamical properties for operations on cellular automata. *Theor. Comput. Sci.*, 410(38-40):3685–3693, 2009.

3 Luigi Acerbi, Alberto Dennunzio, and Enrico Formenti. Surjective multidimensional cellular automata are non-wandering: A combinatorial proof. *Inf. Process. Lett.*, 113(5-6):156–159, 2013.

4 Nobuo Aoki. Ergodic Automorphisms of Compact Metric Groups are Isomorphic to Bernoulli Shifts. *Publications mathématiques et informatique de Rennes*, S4:1–10, 1975.

5 Vincent Bernardi, Bruno Durand, Enrico Formenti, and Jarkko Kari. A New Dimension Sensitive Property for Cellular Automata. In Jirí Fiala, Václav Koubek, and Jan Kratochvíl, editors, *Mathematical Foundations of Computer Science 2004, 29th International Symposium, MFCS 2004, Prague, Czech Republic, August 22-27, 2004, Proceedings*, volume 3153 of *Lecture Notes in Computer Science*, pages 416–426. Springer, 2004.

6 Nino Boccara and Henryk Fuks. Number-conserving cellular automaton rules. *Fundam. Inform.*, 52(1-3):1–13, 2002.

7 Mike Boyle and Bruce Kitchens. Periodic points for onto cellular automata. *Indagationes Mathematicae*, 10(4):483–493, 1999.

8 Lieven Le Bruyn and Michel Van den Bergh. Algebraic properties of linear cellular automata. *Linear algebra and its applications*, 157:217–234, 1991.

9 Gianpiero Cattaneo, Alberto Dennunzio, and Fabio Farina. A Full Cellular Automaton to Simulate Predator-Prey Systems. In Samira El Yacoubi, Bastien Chopard, and Stefania Bandini, editors, *Cellular Automata, 7th International Conference on Cellular Automata, for Research and Industry, ACRI 2006, Perpignan, France, September 20-23, 2006, Proceedings*, volume 4173 of *Lecture Notes in Computer Science*, pages 446–451. Springer, 2006.

10 Gianpiero Cattaneo, Alberto Dennunzio, Enrico Formenti, and Julien Provillard. Non-uniform Cellular Automata. In Adrian-Horia Dediu, Armand-Mihai Ionescu, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings*, volume 5457 of *Lecture Notes in Computer Science*, pages 302–313. Springer, 2009.

11 Gianpiero Cattaneo, Alberto Dennunzio, and Luciano Margara. Solution of some conjectures about topological properties of linear cellular automata. *Theoretical Computer Science*, 325(2):249–271, 2004.

12 Gianpiero Cattaneo, Enrico Formenti, Giovanni Manzini, and Luciano Margara. Ergodicity, transitivity, and regularity for linear cellular automata over $\mathbb{Z}_m$. *Theor. Comput. Sci.*, 233(1-2):147–164, 2000.

13 Bruno Codenotti and Luciano Margara. Transitive Cellular Automata are Sensitive. *The American Mathematical Monthly*, 103(1):58–62, 1996.

14 Michele d'Amico, Giovanni Manzini, and Luciano Margara. On computing the entropy of cellular automata. *Theoretical Computer Science*, 290(3):1629–1646, 2003.

15 Manfred Denker, Christian Grillenberger, and Karl Sigmund. *Ergodic Theory on Compact Spaces*, volume 527 of *Lecture Notes in Mathematics*. Springer-Verlag Berlin Heidelberg, 1976.

16 Alberto Dennunzio. From One-dimensional to Two-dimensional Cellular Automata. *Fundamenta Informaticae*, 115(1):87–105, 2012.

17 Alberto Dennunzio, Pietro Di Lena, Enrico Formenti, and Luciano Margara. Periodic Orbits and Dynamical Complexity in Cellular Automata. *Fundamenta Informaticae*, 126(2-3):183–199, 2013.

18 Alberto Dennunzio, Enrico Formenti, and Luca Manzoni. Computing Issues of Asynchronous CA. *Fundamenta Informaticae*, 120(2):165–180, 2012.

19 Alberto Dennunzio, Enrico Formenti, and Luca Manzoni. Reaction systems and extremal combinatorics properties. *Theoretical Computer Science*, 598:138–149, 2015.

20 Alberto Dennunzio, Enrico Formenti, Luca Manzoni, Luciano Margara, and Antonio E. Porreca. On the dynamical behaviour of linear higher-order cellular automata and its decidability. *Inf. Sci.*, 486:73–87, 2019.

21 Alberto Dennunzio, Enrico Formenti, Luca Manzoni, and Antonio E. Porreca. Ancestors, descendants, and gardens of Eden in reaction systems. *Theoretical Computer Science*, 608:16–26, 2015.

22 Alberto Dennunzio, Enrico Formenti, and Julien Provillard. Non-uniform cellular automata: Classes, dynamics, and decidability. *Information and Computation*, 215:32–46, 2012.

23 Alberto Dennunzio, Enrico Formenti, and Julien Provillard. Local rule distributions, language complexity and non-uniform cellular automata. *Theoretical Computer Science*, 504:38–51, 2013.

24 Alberto Dennunzio, Enrico Formenti, and Julien Provillard. Three research directions in non-uniform cellular automata. *Theoretical Computer Science*, 559:73–90, 2014.

25 Alberto Dennunzio, Enrico Formenti, and Michael Weiss. Multidimensional cellular automata: closing property, quasi-expansivity, and (un)decidability issues. *Theoretical Computer Science*, 516:40–59, 2014.

26 Alberto Dennunzio, Pierre Guillon, and Benoît Masson. Stable Dynamics of Sand Automata. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and C.-H. Luke Ong, editors, *Fifth IFIP International Conference On Theoretical Computer Science - TCS 2008, IFIP 20th World Computer Congress, TC 1, Foundations of Computer Science, September 7-10, 2008, Milano, Italy*, volume 273 of *IFIP*, pages 157–169. Springer, 2008.

27 Robert Luke Devaney. *An Introduction to Chaotic Dynamical Systems.* Addison-Wesley advanced book program. Addison-Wesley, 1989.

28 Pietro di Lena and Luciano Margara. Computational complexity of dynamical systems: The case of cellular automata. *Inf. Comput.*, 206(9-10):1104–1116, 2008.

29 Bruno Durand, Enrico Formenti, and Zsuzsanna Róka. Number-conserving cellular automata I: decidability. *Theor. Comput. Sci.*, 299(1-3):523–535, 2003.

30 Bruno Durand, Enrico Formenti, and Georges Varouchas. On undecidability of equicontinuity classification for cellular automata. In Michel Morvan and Eric Rémila, editors, *Discrete Models for Complex Systems, DMCS'03, Lyon, France, June 16-19, 2003*, volume AB of *Discrete Mathematics and Theoretical Computer Science Proceedings*, pages 117–128. DMTCS, 2003.

31 Michele Finelli, Giovanni Manzini, and Luciano Margara. Lyapunov Exponents versus Expansivity and Sensitivity in Cellular Automata. *J. Complexity*, 14(2):210–233, 1998.

32 Enrico Formenti and Aristide Grange. Number conserving cellular automata II: dynamics. *Theor. Comput. Sci.*, 304(1-3):269–290, 2003.

33 Enrico Formenti, Jarkko Kari, and Siamak Taati. On the hierarchy of conservation laws in a cellular automaton. *Natural Computing*, 10(4):1275–1294, 2011.

34 Pierre Guillon and Gaétan Richard. Revisiting the Rice Theorem of Cellular Automata. In Jean-Yves Marion and Thomas Schwentick, editors, *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, volume 5 of *LIPIcs*, pages 441–452. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

35 G. A. Hedlund. Endomorphisms and Automorphisms of the shift Dynamical System. *Mathematical Systems Theory*, 3:320–375, 1969.

36 Masanobu Ito, Nobuyasu Osato, and Masakazu Nasu. Linear cellular automata over $\mathbb{Z}_m$. *Journal of Computer and Systems Sciences*, 27:125–140, 1983.

37 Jarkko Kari. The Nilpotency Problem of One-Dimensional Cellular Automata. *SIAM J. Comput.*, 21(3):571–586, 1992.

38 Jarkko Kari. Reversibility and Surjectivity Problems of Cellular Automata. *J. Comput. Syst. Sci.*, 48(1):149–182, 1994.

39 Jarkko Kari. Rice's theorem for the limit sets of cellular automata. *Theor. Comput. Sci.*, 127(2):229–254, 1994.

40 Jarkko Kari. Linear Cellular Automata with Multiple State Variables. In Horst Reichel and Sophie Tison, editors, *STACS 2000*, volume 1770 of *LNCS*, pages 110–121. Springer-Verlag, 2000.

41 P. Kůrka. *Topological and Symbolic Dynamics.* Volume 11 of Cours Spécialisés. Société Mathématique de France, 2004.

42 Ville Lukkarila. Sensitivity and Topological Mixing are Undecidable for Reversible One-dimensional Cellular Automata. *J. Cellular Automata*, 5(3):241–272, 2010.

43 Giovanni Manzini and Luciano Margara. A Complete and Efficiently Computable Topological Classification of D-dimensional Linear Cellular Automata over $\mathbb{Z}_m$. *Theor. Comput. Sci.*, 221(1-2):157–177, 1999.

44 Giovanni Manzini and Luciano Margara. Attractors of Linear Cellular Automata. *J. Comput. Syst. Sci.*, 58(3):597–610, 1999.

45 E. F. Moore. Machine models of self-reproduction. *Proceedings of Symposia in Applied Mathematics*, 14:13–33, 1962.

46 T. K. Subrahmonian Moothathu. Homogenity of surjective cellular automata. *Discrete and continuous dynamical systems*, 13:195–202, 2005.

47 J. Myhill. The converse to Moore's garden-of-eden theorem. *Proceedings of the American Mathematical Society*, 14:685–686, 1963.

48 Mazi Shirvani and Thomas D. Rogers. Ergodic endomorphisms of compact abelian groups. *Communications in Mathematical Physics*, 118:401–410, 1988.

49 Shinji Takesue. Staggered invariants in cellular automata. *Complex Systems*, 9:149–168, 1995.

50 Peter Walters. *An introduction to ergodic theory*, volume 79 of *Grauate text in mathematics*. Springer-Verlag, 1982.

On Synthesis of Resynchronizers for Transducers

Sougata Bose
LaBRI, University of Bordeaux, France

Shankara Narayanan Krishna
Department of Computer Science & Engineering IIT Bombay, India

Anca Muscholl
LaBRI, University of Bordeaux, France

Vincent Penelle
LaBRI, University of Bordeaux, France

Gabriele Puppis
CNRS, LaBRI, University of Bordeaux, France

Abstract

We study two formalisms that allow to compare transducers over words under origin semantics: rational and regular resynchronizers, and show that the former are captured by the latter. We then consider some instances of the following synthesis problem: given transducers T_1, T_2, construct a rational (resp. regular) resynchronizer R, if it exists, such that T_1 is contained in $R(T_2)$ under the origin semantics. We show that synthesis of rational resynchronizers is decidable for functional, and even finite-valued, one-way transducers, and undecidable for relational one-way transducers. In the two-way setting, synthesis of regular resynchronizers is shown to be decidable for unambiguous two-way transducers. For larger classes of two-way transducers, the decidability status is open.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases String transducers, resynchronizers, synthesis

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.69

Related Version A full version of the paper is available at `https://arxiv.org/abs/1906.08688`.

Funding DeLTA project (ANR-16-CE40-0007)

1 Introduction

The notion of word transformation is pervasive in computer science, as computers typically process streams of data and transform them between different formats. The most basic form of word transformation is realized using finite memory. Such a model is called finite-state transducer and was studied from the early beginnings of automata theory. Differently from automata, the expressiveness of transducers is significantly affected by the presence of non-determinism (even when the associated transformation is a function), and by the capability of processing the input in both directions (one-way vs two-way transducers). Another difference is that many problems, notably, equivalence and containment, become undecidable when moving from automata to transducers [11, 14].

An alternative semantics for transducers, called *origin semantics*, was introduced in [4] in order to obtain canonical two-way word transducers. In the origin semantics, the output is tagged with positions of the input, called origins, that describe where each output element was produced. According to this semantics, two transducers may be non-equivalent even when they compute the same relation in the classical semantics. From a computational viewpoint, the origin semantics has the advantage that it allows to recover the decidability of equivalence and containment of non-deterministic (and even two-way) transducers [6].

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 69; pp. 69:1–69:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

It can be argued that comparing two transducers in the origin semantics is rather restrictive, because it requires that the same output is generated at precisely the same place. A natural approach to allow some 'distortion' of the origin information when comparing two transducers was proposed in [10]. *Rational* resynchronizers allow to compare *one-way* transducers (hence, the name 'rational') under origin distortions that are generated with finite control. A rational resynchronizer is simply a one-way transducer that processes an interleaved input-output string, producing another interleaved interleaved input-output string with the same input and output projection. For two-way transducers (or equivalently, streaming string transducers [1]) a different formalism is required to capture origin distortion, since the representation of the origin information through interleaved input-output pairs does not work anymore. To this purpose, *regular resynchronizers* were introduced in [6] as a logic-based transformation of origin graphs, in the spirit of Courcelle's monadic second-order logic definable graph transductions [8]. In [6] it was shown that containment of two-way transducers up to a (bounded) regular resynchronizer is decidable.

In this paper we first show that bounded regular resynchronizers capture the rational ones. This result is rather technical, because rational resynchronizers work on explicit origin graphs, encoded as input-output pairs, which is not the case for regular resynchronizers. Then we consider the following problem: given two transducers T_1, T_2, we ask whether some rational, or bounded regular, resynchronizer R exists such that T_1 is origin-contained in T_2 up to R. So here, the resynchronizer R is not part of the input, and we want to synthesize such a resynchronizer, if one exists.

Our main contributions can be summarized as follows:

1. synthesis of rational resynchronizers for functional (or even finite-valued) one-way transducers is decidable,
2. synthesis of rational resynchronizers for unrestricted one-way transducers is undecidable,
3. synthesis of bounded regular resynchronizers for unambiguous two-way transducers is decidable.

Somewhat surprisingly, for both decidable cases above the existence of a resynchronizer turns out to be equivalent to the classical inclusion of the two transducers.

Full proofs of the results presented in this paper can be found in the extended version `https://arxiv.org/abs/1906.08688`.

2 Preliminaries

One-way transducers

One of the simplest transducer model is the one-way non-deterministic finite-state transducer (hereafter, simply one-way transducer), capturing the class of so-called *rational relations*. This is basically an automaton in which every transition consumes one letter from the input and appends a word of any length to the output.

Formally, a *one-way transducer* is a tuple $T = (\Sigma, \Gamma, Q, I, E, F, L)$, where Σ, Γ are finite input and output alphabets, Q is a finite set of states, $I, F \subseteq Q$ are subsets of initial and final states, $E \subseteq Q \times \Sigma \times Q$ is a finite set of transition rules, and $L : E \uplus F \to 2^{\Gamma^*}$ is a function specifying a regular language of partial outputs for each transition rule and each final state. The relation defined by T contains pairs (u, v) of input and output words, where $u = a_1 \dots a_n$ and $v = v_1 \dots v_n \, v_{n+1}$, for which there is a run $q_0 \xrightarrow{a_1 \mid v_1} q_1 \xrightarrow{a_2 \mid v_2} \dots q_n \xrightarrow{\mid v_{n+1}}$ such that $q_0 \in I$, $q_n \in F$, $(q_{i-1}, a_i, q_i) \in E$, $v_i \in L(q_{i-1}, a_i, q_i)$, and $v_{n+1} \in L(q_n)$. The transducer is called *functional* if it associates at most one output with each input, namely, if it realizes a partial function. For example, Figure 1 shows two one-way transducers with input alphabet

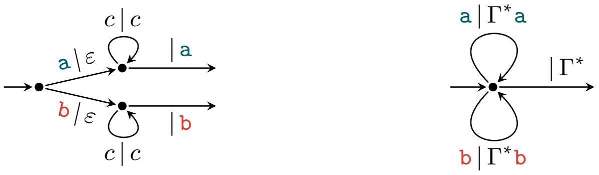


Figure 1 Examples of functional and relational one-way transducers.

$\Sigma = \{a, b\}$ and output alphabet $\Gamma \supseteq \Sigma$. The first transducer is functional, and realizes the cyclic rotation $f : cu \mapsto uc$, for any letter $c \in \{a, b\}$ and any word $u \in \{a, b\}^*$. The second transducer is not functional, and associates with an input $u \in \Sigma^*$ any possible word $v \in \Gamma^*$ as output such that u is a sub-sequence of v.

Two-way transducers

Allowing the input head to move in any direction, to the left or to the right, gives a more powerful model of transducer, which captures e.g. the relation $\{(u, u^n) : u \in \Sigma^*, n \in \mathbb{N}\}$. To define two-way transducers, we adopt the convention that, for any given input $u \in \Sigma^*$, $u(0) = \vdash$ and $u(|u| + 1) = \dashv$, where $\vdash, \dashv \notin \Sigma$ are special markers used as delimiters of the input. In this way, a transducer can detect when an endpoint of the input has been reached.

A *two-way transducer* is a tuple $T = (\Sigma, \Gamma, Q, I, E, F, L)$, whose components are defined just like those of a one-way transducer, except that the state set Q is partitioned into two subsets, $Q_\prec$ and $Q_\succ$, the set I of initial states is contained in $Q_\succ$, and the set E of transition rules is contained in $(Q \times \Sigma \times Q) \uplus (Q_\prec \times \{\vdash\} \times Q_\succ) \uplus (Q_\succ \times \{\dashv\} \times Q_\prec)$. The partitioning of the set of states is useful for specifying which letter is read from each state: states from $Q_\prec$ read the letter to the left, whereas states from $Q_\succ$ read the letter to the right. Given an input $u \in \Sigma^*$, a configuration of a two-way transducer is a pair (q, i), with $q \in Q$ and $i \in \{1, \ldots, |u| + 1\}$. Based on the types of source and target states in a transition rule, we can distinguish four types of transitions between configurations (the output v is always assumed to range over the language $L(q, a, q')$):

- $(q, i) \xrightarrow{a \mid v} (q', i + 1)$ if $(q, a, q') \in E$, $q, q' \in Q_\succ$, and $a = u(i)$,
- $(q, i) \xrightarrow{a \mid v} (q', i)$ if $(q, a, q') \in E$, $q \in Q_\succ$, $q' \in Q_\prec$, and $a = u(i)$,
- $(q, i) \xrightarrow{a \mid v} (q', i - 1)$ if $(q, a, q') \in E$, $q, q' \in Q_\prec$, and $a = u(i - 1)$,
- $(q, i) \xrightarrow{a \mid v} (q', i)$ if $(q, a, q') \in E$, $q \in Q_\prec$, $q' \in Q_\succ$, and $a = u(i - 1)$.

Note that, when reading a marker $\vdash$ or $\dashv$, the transducer is obliged to make a U-turn, either left-to-right or right-to-left. The notions of successful run, realized relation, and functional transducer are naturally generalized from the one-way to the two-way variant, (we refer to [6] for more details).

In [5], a slight extension of two-way transducers, called two-way transducers *with common guess*, was proposed. Before processing its input, such a transducer can non-deterministically guess some arbitrary annotation of the input over a fixed alphabet. Once an annotation is guessed, it remains the same during the computation. Transitions may then depend on the input letter and the guessed annotation at the current position. For example, this extension allows to define relations of the form $\{(u, vv) \mid u \in \Sigma^*, v \in \Gamma^*, |u| = |v|\}$. Note that the extension with common guess does not increase the expressiveness of one-way transducers, since these are naturally closed under input projections. Likewise, common guess does not affect the expressive power of functional two-way transducers, since one can guess a canonical annotation at runtime.

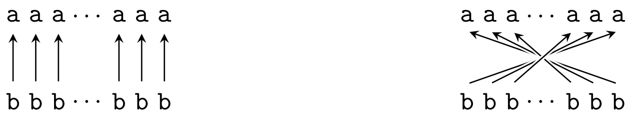


Figure 2 Input-output pairs annotated with origin information.

Classical vs origin semantics

In the previous definitions, we associated a classical semantics to transducers (one-way or two-way), which gives rise to relations or functions between input words over Σ and output words over Γ. In [4] an alternative semantics for transducers, called *origin semantics*, was introduced with the goal of getting canonical transducers for any given word function. Roughly speaking, in the origin semantics, every position of the output word is annotated with the position of the input where that particular output element was produced. This yields a bipartite graph, called *origin graph*, with two linearly ordered sets of nodes, representing respectively the input and the output elements, and edges directed from output nodes to input nodes, representing the so-called *origins*. Figure 2 depicts an input-output pair (a^n, b^n) annotated with two different origins: in the first graph, a position i in the output has its origin at the same position i in the input, while in the second graph it has origin at position $n - i$.

Formally, the origin semantics of a transducer is a relation $S_o \subseteq \Sigma^* \times (\Gamma \times \mathbb{N})^*$ consisting of pairs (u, ν), where $u = a_1 \dots a_n \in \Sigma^*$ is a possible input and $\nu = \nu_1 \dots \nu_{m+1} \in (\Gamma \times \mathbb{N})^*$ is the corresponding output tagged with input positions, as induced by a successful run of the form $(q_0, i_0) \xrightarrow{a_1 \mid \nu_1} (q_1, i_1) \xrightarrow{a_2 \mid \nu_2} \dots (q_m, i_m) \xrightarrow{\mid \nu_{m+1}}$, with each $\nu_j \in (\Gamma \times \{i_j\})^*$. We identify a pair (u, ν) with the origin graph obtained by arranging the input elements and the output elements along two lines (we omit the successor relation in the graph notation), and adding edges from every output element (a, i) to the i-th element of the input. Given an origin graph $G = (u, \nu)$, we denote by $\mathsf{in}(G)$, $\mathsf{out}(G)$, and $\mathsf{orig}(G)$ respectively the input word u, the output word obtained by projecting ν onto the finite alphabet Γ, and the sequence of input positions (origins) obtained by projecting ν onto $\mathbb{N}$.

For one-way transducers, there is a simpler presentation of origin graphs in the form of interleaved words. Assuming that the alphabets Σ and Γ are disjoint, we interleave the input and output word by appending after each input symbol the output word produced by reading that symbol. For example, if $\Sigma = \{a\}$ and $\Gamma = \{b\}$, then a word of the form $abb \dots abb$ represents an origin graph (a^n, ν), where $|\nu| = 2n$ and $\nu(2i - 1) = \nu(2i) = (b, i)$, for all $i = 1, \dots, n$. Words over $\Sigma \uplus \Gamma$ are called *synchronized words*. Just as every synchronized word represents an origin graph, a regular language over $\Sigma \uplus \Gamma$ represents a rational relation with origins, or equally the origin semantics of a one-way transducer.

In general, when comparing transducers, we can refer to one of the two possible semantics. Clearly, two transducers that are equivalent in the origin semantics are also equivalent in the classical semantics, but the converse is not true.

3 Resynchronizations

The central concept of this paper is that of resynchronization, which is a transformation of origin graphs that preserves the underlying input and output words. The concept was originally introduced in [10], and mostly studied in the setting of rational relations. Here we use the concept in the more general setting of relations definable by two-way transducers.

Formally, a *resynchronization* is any relation $R \subseteq (\Sigma^* \times (\Gamma \times \mathbb{N})^*)^2$ that contains only pairs (G, G') of origin graphs such that $\mathsf{in}(G) = \mathsf{in}(G')$ and $\mathsf{out}(G) = \mathsf{out}(G')$, namely, with the same projections onto the input and output alphabets.[1] A resynchronization R can be used to modify the origin information of a relation, while preserving the underlying input-output pairs. Formally, for every relation $S_o \subseteq \Sigma^* \times (\Gamma \times \mathbb{N})^*$ with origins, we define the *resynchronized relation* $R(S_o) = \{G' \in S_o \mid (G, G') \in R,\ G \in S_o\}$. Note that if the origin information is removed from both $R(S_o)$ and S_o, then $R(S_o) \subseteq S_o$. Moreover, $R(S_o) = S_o$ when R is the *universal resynchronization*, that is, when R contains all pairs (G, G'), with $G, G' \in \Sigma^* \times (\Gamma \times \mathbb{N})^*$, $\mathsf{in}(G) = \mathsf{in}(G')$, and $\mathsf{out}(G) = \mathsf{out}(G')$.

Definability of resynchronized relations

An important property that we need to guarantee in order to enable some effective reasoning on resynchronizations is the definability of the resynchronized relations. More precisely, given a class $\mathcal{C}$ of transducers, we say that a resynchronization R *preserves definability in* $\mathcal{C}$ if for every transducer $T \in \mathcal{C}$, the relation $R(T)$ is realized by some transducer $T' \in \mathcal{C}$, that can be effectively constructed from R and T. The class $\mathcal{C}$ will usually be the class of one-way transducers or the class of two-way transducers, and this will be clear from the context.

Below, we recall the definitions of two important classes of resynchronizations, called rational [10] and regular resynchronizers [6], that preserve definability by one-way transducers and by two-way transducers, respectively. We will then compare the expressive power of these two formalisms, showing that rational resynchronizers are strictly less expressive than regular resynchronizers.

Rational resynchronizers

A natural definition of resynchronizers for one-way transducers is obtained from rational relations over the disjoint union $\Sigma \uplus \Gamma$ of the input and output alphabets. Any such relation consists of pairs of synchronized words (w, w'), and thus represents a transformation of origin graphs. In addition, if the induced synchronized words w and w' have the same projections over the input and output alphabets, then the relation represents a resynchronization. We also recall that rational relations are captured by one-way transducers, so, by analogy, we call *rational resynchronizer* any one-way transducer over $\Sigma \uplus \Gamma$ that preserves the input and output projections.

It is routine to see that rational resynchronizers preserve definability of relations by one-way transducers. It is also worth noting that every rational resynchronizer is a length-preserving transducer. By a classical result of Elgot and Mezei [9] every rational resynchronizer can be assumed to be a *letter-to-letter* one-way transducer, namely, a transducer with transitions of the form $q \xrightarrow{a \mid b} q'$, with $a, b \in \Sigma \uplus \Gamma$.

▶ **Example 1.** Consider the functional one-way transducers T_1, T_2 in Figure 3. The domain of both transducers is $(aa)^*$. An origin graph of T_1 is a one-to-one mapping from the output to the input (each a produces one b). On the other hand, in an origin graph of T_2, every a at input position $2i+1$ is the origin of two b's at output positions $2i+1, 2i+2$. The transducer R depicted to the right of the figure transforms synchronized words while preserving their input and output projections. It is then a rational resynchronizer. In particular, R transforms origin graphs of T_1 to origin graphs of T_2.

[1] In [10], resynchronizers were further restricted to contain at least the pairs of identical origin graphs. Here we prefer to avoid this additional restriction and reason with a more general class of resynchronizations.

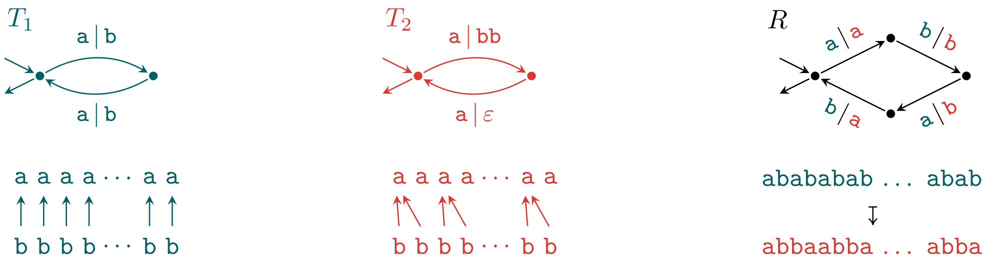


Figure 3 Two functional 1NFT T_1, T_2, their origin graphs, and a rational resynchronizer R.

Regular resynchronizers

While languages of synchronized words are a faithful representation of rational relations, this notation does not capture regular relations, so relations realized by two-way transducers. An alternative formalism for resynchronizations of relations defined by two-way transducers was proposed in [6] under the name of MSO resynchronizer (here we call it simply "resynchronizer"). The formalism describes pairs (G, G') of origin graphs by means of two relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$ $(\gamma, \gamma' \in \Gamma)$ in the spirit of MSO graph transductions. More precisely:

- move_γ describes how the origin y of an output position x labeled by γ is redirected to a new origin z (for short, we call y and z the *source* and *target* origins of x). Formally, move_γ is a relation contained in $\Sigma^* \times \mathbb{N} \times \mathbb{N}$ that induces resynchronization pairs (G, G') such that, for all output positions x, if $\mathsf{out}(G)(x) = \gamma$, $\mathsf{orig}(G)(x) = y$, and $\mathsf{orig}(G')(x) = z$, then $(\mathsf{in}(G), y, z) \in \mathsf{move}_\gamma$.
- $\mathsf{next}_{\gamma,\gamma'}$ constrains the target origins z and z' of any two consecutive output positions x and $x+1$ that are labelled by γ and γ', respectively. Formally, $\mathsf{next}_{\gamma,\gamma'}$ is a relation contained in $\Sigma^* \times \mathbb{N} \times \mathbb{N}$ that induces resynchronization pairs (G, G') such that, for all output positions x and $x+1$, if $\mathsf{out}(G)(x) = \gamma$, $\mathsf{out}(G)(x+1) = \gamma'$, $\mathsf{orig}(G')(x) = z$, and $\mathsf{orig}(G')(x+1) = z'$, then $(\mathsf{in}(G), z, z') \in \mathsf{next}_{\gamma,\gamma'}$.

A *resynchronizer* is a tuple $\big((\mathsf{move}_\gamma)_{\gamma\in\Gamma}, (\mathsf{next}_{\gamma,\gamma'})_{\gamma,\gamma'\in\Gamma}\big)$, and defines the resynchronization R with pairs (G, G') induced by the relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$, where $\gamma, \gamma' \in \Gamma$.

In order to obtain a well-behaved class of resynchronizations, that in particular preserves definability by two-way transducers, we need to enforce some restrictions. First, we require that the relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$ are described by regular languages (or equally, definable in monadic second-order logic). By this we mean that we encode the input positions y, z, z' with suitable annotations over the binary alphabet $\mathbb{B} = \{0, 1\}$, so that we can identify the relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$ with some *regular* languages over the expanded alphabet $\Sigma \times \mathbb{B}^2$. We call *regular resynchronizer* a resynchronizer where the relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$ are given by regular languages. In addition, we also require that regular resynchronizers are k*-bounded*, for some $k \in \mathbb{N}$, in the sense that for every input u, every output letter γ, and every target origin z, there are at most k positions y such that $(u, y, z) \in \mathsf{move}_\gamma$.

▶ **Example 2.** Consider the resynchronization R of Figure 4, containing the pairs (G, G'), where G (resp. G') is the origin graph that maps every output position to the first (resp. last) input position. R is "one-way", in the sense that it contains only origin graphs that are admissible outcomes of runs of one-way transducers. However, R is not definable by any rational resynchronizer, since, in terms of synchronized words, it should map $a\,v\,u$ to $a\,u\,v$, for every $a \in \Sigma$, $u \in \Sigma^*$, and $v \in \Gamma^*$, which is clearly not a rational relation. The resynchronization R can however be defined by a 1-bounded regular resynchronizer, e.g. $\big((\mathsf{move}_\gamma)_{\gamma\in\Gamma}, (\mathsf{next}_{\gamma,\gamma'})_{\gamma,\gamma'\in\Gamma}\big)$, with $\mathsf{move}_\gamma = \{(u, y, z) \mid u \in \Sigma^*, y = 1, z = |u|\}$ and

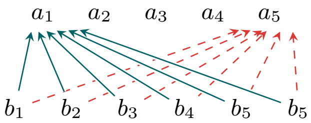


Figure 4 A 1-bounded, regular resynchronization that is not rational.

$\mathsf{next}_{\gamma,\gamma'} = \Sigma^* \times \mathbb{N} \times \mathbb{N}$.

One can observe that, in the previous example, next is not restricting the resynchronization further. For other examples that use next in a non-trivial way see for instance [6, Example 13].

The notion of resynchronizer can be slightly enhanced in order to allow some additional amount of non-determinism in the way origin graphs are transformed (this enhanced notion is indeed the one proposed in [6]). The principle is very similar to the idea of enhancing two-way transducers with common guess. More precisely, we allow additional monadic parameters that annotate the input and the output, thus obtaining words over expanded alphabets of the form $\Sigma \times \Sigma'$ and $\Gamma \times \Gamma'$. A resynchronizer *with parameters* is thus a tuple $\big(\mathsf{ipar}, \mathsf{opar}, (\mathsf{move}_\gamma)_\gamma, (\mathsf{next}_{\gamma,\gamma'})_{\gamma,\gamma'}\big)$, where $\mathsf{ipar} \subseteq (\Sigma \times \Sigma')^*$ describes the possible annotations of the input, $\mathsf{opar} \subseteq (\Gamma \times \Gamma')^*$ describes the possible annotations of the output, and, for every $\gamma, \gamma' \in \Gamma \times \Gamma'$, $\mathsf{move}_\gamma \subseteq (\Sigma \times \Sigma' \times \mathbb{B}^2)^*$ describes a transformation from source to target origins of γ-labelled output positions, and $\mathsf{next}_{\gamma,\gamma'} \subseteq (\Sigma \times \Sigma' \times \mathbb{B}^2)$ constraints the target origins of consecutive output positions labelled by γ and γ'. The resynchronization pairs (G, G') in this case are induced by $\big((\mathsf{move}_\gamma)_{\gamma \in \Gamma \times \Gamma'}, (\mathsf{next}_{\gamma,\gamma'})_{\gamma,\gamma' \in \Gamma \times \Gamma'}\big)$ and are obtained by projecting the input and output over the original alphabets Σ and Γ, under the assumption that the annotations satisfy ipar and opar. A resynchronizer with parameters is called *regular* if all its relations are regular. A regular resynchronizer is called *bounded* if is k-bounded for some k.

In [6] it was shown that, given a bounded regular resynchronizer R with parameters and a two-way transducer T with common guess, one can construct a two-way transducer T' with common guess such that $T' =_o R(T)$. The notation $T' =_o R(T)$ is used to represent the fact that T' and $R(T)$ define the same relation in the origin semantics.

Unless otherwise stated, hereafter we assume that two-way transducers are enhanced with common guess, and regular resynchronizers are enhanced with parameters.

Rational vs regular resynchronizers

Our first result shows that bounded, regular resynchronizers are more expressive than rational resynchronizers. Consider for instance Example 1: it can be captured by the regular resynchronizer with opar annotating even/odd positions. The resynchronizer shifts the origins of the even positions of the output by one to the left and keeps the origins of the odd positions unchanged. So here move_γ can be described by a regular language. On the other hand, Example 2 shows that there are bounded, regular resynchronizers that cannot be captured by rational resynchronizers.

▶ **Theorem 3.** *For every rational resynchronizer, there is an equivalent* 1*-bounded regular resynchronizer.*

The proof of the above result is rather technical and can be found in the extended version. Here we only provide a rough idea. Consider a rational resynchronizer R, that is, a one-way transducer that transforms synchronized words while preserving the input and output projections. For example, Figure 5 represents a possible pair of synchronized words,

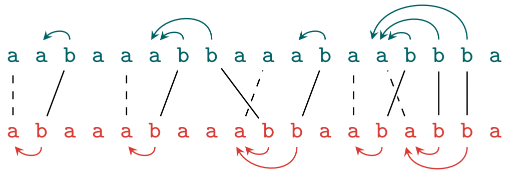

a a b a a a b b a a a b a a b b b a
a b a a a b a a a b b a a b a b b a

Figure 5 A 1-bounded, regular resynchronization that is not rational.

denoted w and w', shown in blue and in red, respectively, such that $(w, w') \in R$. We assume that $\Sigma = \{a\}$ and $\Gamma = \{b\}$.

From the given rational resynchronizer R we construct an equivalent 1-bounded, regular resynchronizer R'. The natural approach is to encode a successful run ρ of R over a synchronized word w. By measuring the differences between the partial inputs and the partial outputs that are consumed and produced along the run ρ, we obtain a partial bijection on the input letters that represents a mapping from source origins to target origins. This mapping determines the relation move_γ of R', and in fact depends on a suitable additional annotation γ of the underlying output position. The additional annotation is needed in order to distinguish output elements with the same origin in the source, but with different origins in the target.

For example, by referring again to the figure above, consider the first occurrence of b in w. Its origin in w is given by the closest input letter to the left (follow the blue arrow). To find the origin in w', one finds the same occurrence of b in w' (solid line), then moves to the closest input letter to the left (red arrow), and finally maps the latter input position in w' back to w (dashed line). The resulting position determines the new origin (w.r.t. w') of the considered output element.

The remaining components ipar, opar, and $\mathsf{next}_{\gamma,\gamma'}$ of R' are used to guarantee the correctness of the various annotations (notably, the correctness of the encoding of the run ρ and that of the output annotations).

4 Synthesis of Resynchronizers

Recall that containment between transducers depends on the adopted semantics. More precisely, according to the classical semantics, T_1 is contained in T_2 (denoted $T_1 \subseteq T_2$) if all input-output pairs realized by T_1 are also realized by T_2; according to the origin semantics, T_1 is contained in T_2 (denoted $T_1 \subseteq_o T_2$) if all origin graphs realized by T_1 are also realized by T_2. In this section, we study the following variant of the containment problem:

Resynchronizer synthesis problem.
Input: two transducers T_1, T_2.
Question: does there exist some resynchronization R such that $T_1 \subseteq_o R(T_2)$.

In fact, the above problem comes in several variants, depending on the model of transducers considered (one-way or two-way) and the class of admissible resynchronizations R (rational or bounded regular). Moreover, for the positive instances of the above problem, we usually ask to compute a witnessing resynchronization R from the given T_1 and T_2 (this is the reason for calling the problem a *synthesis problem*).

Clearly, the synthesis problem for unrestricted resynchronizers is equivalent to a classical containment, that is, $T_1 \subseteq T_2$ if and only if $T_1 \subseteq_o R(T_2)$ for some resynchronizer R. Therefore, the synthesis problem for unrestricted resynchronizers is undecidable. Thus we will consider the synthesis problem of rational (resp. bounded regular) resynchronizers for one-way (resp. two-way) transducers.

We also recall that rational resynchronizers preserve definability of relations by one-way transducers [10], while bounded regular resynchronizers (which, by Theorem 3, are strictly more expressive than rational resynchronizers) preserve definability by two-way transducers [6]. For the sake of presentation, we shall first consider the synthesis of rational resynchronizers in the functional one-way setting, that is, for instances given by functional one-way transducers. We show that in this setting the problem collapses again to the classical containment problem, which is however decidable now, that is: $T_1 \subseteq T_2$ if and only if $T_1 \subseteq_o R(T_2)$ for some rational resynchronizer R. The decidability result can be slightly extended to some non-functional transducers. More precisely, we will show that synthesis of rational resynchronizers for finite-valued one-way transducers is still decidable. When moving to the relational case, however, the problem becomes undecidable.

The decidability status in the one-way setting could be also contrasted with the two-way setting. In this respect, we observe that, in the functional case, the synthesis problem does not collapse anymore to classical containment, as there are functional two-way transducers T_1, T_2 such that $T_1 \subseteq T_2$, but for which no bounded regular resynchronizer R satisfies $T_1 \subseteq_o R(T_2)$ (an example can be found at the beginning of Section 4.3). We are able to prove decidability of synthesis of bounded, regular resynchronizers for unambiguous two-way transducers. The decidability status, however, remains open in the functional two-way case, as well as in the unrestricted (non-functional) two-way case.

4.1 Resynchronizing functional, one-way transducers

Recall that it can be decided in PSpace whether a transducer (be it one-way or two-way) is functional [2], and that the classical containment problem for functional (one-way/two-way) transducers is also in PSpace [3]. The following result shows that, for functional one-way transducers, classical containment and rational resynchronizer synthesis are inter-reducible.

▶ **Theorem 4.** *Let T_1, T_2 be two functional one-way transducers. The following conditions are equivalent, and decidable:*

1. $T_1 \subseteq T_2$,
2. $T_1 \subseteq_o R(T_2)$ *for some resynchronization R,*
3. $T_1 =_o R(T_2)$ *for some rational resynchronizer R.*

Proof sketch. The implications from 2. to 1. and 3. to 2. are trivial. The implication from 1. to 3. is proved by constructing a rational resynchronizer R as a product of T_1, T_2: at each step, R consumes a symbol $a \in \Sigma$ and a word $v_1 \in \Gamma^*$ from a transition of T_1 and produces the same symbol a and possibly a different word $v_2 \in \Gamma^*$ from a corresponding transition of T_2. The fact that R preserves the outputs relies on functionality of T_1 and T_2. ◀

A natural question arises: can a characterization similar to Theorem 4 be obtained for transducers that compute arbitrary relations, rather than just functions? The example below provides a negative answer to this question. Later in Section 4.2, we will see that synthesis of rational resynchronizers for unrestricted one-way transducers is an undecidable problem.

▶ **Example 5.** Consider a one-way transducer T_1 that checks that the input is from $(aa)^*$ and produces a single output letter b for each consumed input letter a, and another transducer T_2 that works in two phases: during the first phase, it produces two b's for each consumed a, and during the second phase consumes the remaining part of the input without producing any output. We have $T_1 \subseteq T_2$, but $T_1 \nsubseteq_o T_2$. The only resynchronization R that satisfies $T_1 \subseteq_o R(T_2)$ must map synchronized words from $(ab)^*$ to $(abb)^*(a)^*$, while preserving the number of a's and b's. Such a transformation cannot be defined by any rational resynchronizer, nor by a bounded regular resynchronizer.

There is however an intermediate case, between the functional and the full relational case, for which a generalization of Theorem 4 is possible. This is the case of *finite-valued* one-way transducers, that is, transducers that realize finite unions of partial functions. The generalization exploits a result from [10], stated just below, that concerns synthesis of bounded-delay resynchronizers. Formally, given two origin graphs G and G' with the same input and output projections, and given an input position y, we denote by $\mathsf{delay}_{G,G'}(y)$ the difference between the largest $x \in \mathsf{dom}(\mathsf{out}(G))$ such that $\mathsf{orig}(G)(x) = y$ and the largest $x' \in \mathsf{dom}(\mathsf{out}(G'))$ such that $\mathsf{orig}(G')(x') = y$. Given $d \in \mathbb{N}$, we define the *d-delay resynchronizer* as the resynchronization that contains all pairs (G, G') with the same input and output projections and such that $\mathsf{delay}_{G,G'}(y) \in [-d, +d]$ for all input positions y. It is easy to see that the d-delay resynchronizer is a special case of a rational resynchronizer.

▶ **Theorem 6** (Theorem 13 in [10]). *Let T_1, T_2 be one-way transducers, where T_2 is k-ambiguous.*[2] *One can compute a d-delay resynchronizer R_d, for some $d \in \mathbb{N}$, such that $T_1 \subseteq T_2$ implies $T_1 \subseteq_o R_d(T_2)$.*

As a corollary we can generalize Theorem 4 to k-valued one-way transducers, with the only difference that the witnessing rational resynchronizer now satisfies $T_1 \subseteq_o R(T_2)$ rather than $T_1 =_o R(T_2)$. We also recall that classical containment remains decidable for k-valued one-way transducers, thanks to the fact that these can be effectively transformed to finite unions of functional transducers [15]:

▶ **Corollary 7.** *Let T_1, T_2 be k-valued one-way transducers. The following conditions are equivalent, and decidable:*

1. $T_1 \subseteq T_2$,
2. $T_1 \subseteq_o R(T_2)$ *for some resynchronization R,*
3. $T_1 \subseteq_o R(T_2)$ *for some rational resynchronizer R.*

Proof. We prove the only interesting implication from 1. to 3. Suppose that T_1, T_2 are k-valued one-way transducers such that $T_1 \subseteq T_2$. Using the decomposition theorem from [15], we can construct a k-ambiguous one-way transducer T_2' that is classically equivalent to T_2 and such that $T_2' \subseteq_o T_2$. Since $T_1 \subseteq T_2'$, by Theorem 6 we can compute a d-delay (in particular, rational) resynchronizer R_d such that $T_1 \subseteq_o R_d(T_2')$. Finally, since $T_2' \subseteq_o T_2$, $T_1 \subseteq_o R_d(T_2')$, and $R_d(T_2') \subseteq_o R_d(T_2)$, we get $T_1 \subseteq_o R_d(T_2)$. ◀

4.2 Resynchronizing arbitrary one-way transducers

In the previous section we saw how to synthesize a rational resynchronizer for functional, or even finite-valued, one-way transducers. One may ask if finite-valuedness is necessary. We already know that classical containment $T_1 \subseteq T_2$ is undecidable [11, 12] for arbitrary one-way transducers, whereas origin-containment $T_1 \subseteq_o T_2$ is decidable [6]. Synthesis of a rational resynchronizer R such that $T_1 \subseteq_o R(T_2)$ is a question that lies between the two questions above. We show in this section that in the case of *real-time* transducers with unary output alphabet, the latter question is equivalent to language-boundedness of one-counter automata, a problem that we define below.

A transducer is said to be *real-time* if it produces bounded outputs for each consumed input symbol. A *one-counter automaton* (OCA) is a non-deterministic pushdown automaton

[2] A transducer is k-ambiguous if each input admits at most k successful runs.

with a single stack symbol, besides the bottom stack symbol. In the definition of the language-boundedness problem, we assume that the OCA recognizes a universal language; this assumption is used in the reduction to the synthesis problem.

Language-boundedness of OCA.
Input: An OCA A over alphabet Ω that recognizes the *universal* language $L(A) = \Omega^*$.
Question: Does there exist some bound k such that every word over Ω can be accepted by A with a run where the counter never exceeds k?

Our reductions between language-boundedness of OCA and synthesis of rational resynchronizers rely on the following result from [10], that implies that bounded-delay resynchronizers are enough for synthesizing resynchronizers of real-time transducers:

▶ **Theorem 8** (Theorem 11 in [10]). *Let T_1, T_2 be real-time, one-way transducers and R a rational resynchronizer such that $T_1 \subseteq_o R(T_2)$. One can compute a d-delay resynchronizer R_d such that $T_1 \subseteq_o R_d(T_2)$.*

▶ **Proposition 9.** *Synthesis of rational resynchronizers for real-time one-way transducers with unary output alphabet and language-boundedness of OCA are inter-reducible problems. Moreover, in the reductions, one can assume that the left hand-side transducer is functional.*

Proof sketch. Given some real-time transducers T_1, T_2, one constructs an OCA A that, when the input encodes a successful run of T_1, guesses and simulates an equivalent successful run of T_2. The OCA A keeps track in its counter, how ahead or behind is the partial output produced by the encoded run of T_1 compared to the partial output produced by the simulated run T_2, and accepts with an empty counter. Moreover, A accepts all inputs that do not encode successful runs of T_1: as soon as an error is detected, the counter is reset and frozen. Thus, badly-formed encodings do not affect language-boundedness. Then, using Theorem 8, one shows that A is language-bounded if and only if $T_1 \subseteq_o R(T_2)$ for some rational (and w.l.o.g. bounded-delay) resynchronizer R.

In the opposite reduction, one has to construct some real-time transducers T_1, T_2 from a given OCA A. Both transducers receive inputs over the same alphabet as A. T_1 is a simple functional transducer that outputs one symbol for each consumed input symbol. T_2, instead, guesses and simulates a run of A, and ouputs two symbols when the counter of the OCA increases, and no symbol when it decreases. As before, one argues using Theorem 8 that A is language-bounded if and only if $T_1 \subseteq_o R(T_2)$ for some rational resynchronizer R. ◀

The status of the problem of language-boundedness of OCA was open, to the best of our knowledge. Piotr Hofman communicated to us the following unpublished result, which can be obtained by a reduction from the undecidable boundedness problem for Minsky machines (the proof is in the extended version of this paper):

▶ **Theorem 10** ([13]). *The language-boundedness problem for OCA is undecidable.*

▶ **Corollary 11.** *Synthesis of rational resynchronizers for (real-time) one-way transducers is undecidable, and this holds even when the left hand-side transducer is functional.*

4.3 Resynchronizing unambiguous, two-way transducers

We now focus on the resynchronizer synthesis problem for two-way transducers. Here the appropriate class of resynchronizations is that of regular resynchronizers, since, differently from rational resynchronizer, they can handle origin graphs induced by two-way transducers.

The situation is more delicate, as the synthesis problem does not reduce anymore to classical containment. As an example, consider the transducer T_1 that consumes an input of the form a^* from left to right, while copying the letters to the output, and a two-way transducer T_2 that realizes the same function but while consuming the input in reverse. We have that $T_1 \subseteq T_2$, but there is no resynchronizer R that satisfies $T_1 \subseteq_o R(T_2)$ and that is bounded and regular at the same time. As we will see, extending Theorem 4 to two-way transducers is possible if we move beyond the class of regular resynchronizers and consider bounded resynchronizers defined by Parikh automata. The existence of bounded regular resynchronizers between functional two-way transducers can thus be seen as a strengthening of the classical containment relation. Unfortunately, we are only able to solve the synthesis problem of bounded regular resynchronizers for *unambiguous* two-way transducers, so the problem remains open for functional two-way transducers.

First we introduce resynchronizers definable by Parikh automata. Formally, a *Parikh automaton* is a finite automaton $A = (\Sigma, Q, I, E, F, Z, S)$ equipped with a function $Z : E \to \mathbb{Z}^k$ that associates vectors of integers to transitions and a semi-linear set $S \subseteq \mathbb{Z}^k$. A successful run of A is a run starting in I, ending in F and such as the sum of the weights of its transitions belongs to S. We say that A is *unambiguous* if the underlying finite automaton is. In this case, we can associate with each input u the vector $A(u) \in \mathbb{Z}^k$ associated with the unique accepting run of the underlying automaton of A on u, if this exists, otherwise $A(u)$ is undefined. By taking products, one can easily prove that unambiguous Parikh automata are closed under pointwise sum and difference, that is, given A_1 and A_2, there are A_+ and A_- such that $A_+(u) = A_1(u) + A_2(u)$ and $A_-(u) = A_1(u) - A_2(u)$ for all possible inputs u. Hereafter, we will only consider languages recognized by *unambiguous* Parikh automata with the trivial semilinear set $S = \{0^k\}$.

By a slight abuse of terminology, we call *Parikh resynchronizer* any resynchronizer with parameters whose relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$ are recognizable by unambiguous Parikh automata, and ipar and opar are regular. We naturally inherit from regular resynchronizers the notion of boundedness. Moreover, we introduce another technical notion, that will be helpful later. Given a resynchronizer R, we define its *target set* as the set of all pairs (u, z) where u is an input, z is a position in it, and $(w, y, z) \in \mathsf{move}_\gamma$ for some annotation w of u with input parameters, some input position y, and some output type γ. Similarly, we define the *target set* of a two-way transducer T as the set of all pairs (u, z), where $u = \mathsf{in}(G)$ and $z \in \mathsf{orig}(G)(x)$ for some $x \in \mathsf{dom}(\mathsf{out}(G))$ and some origin graph G realized by T.

▶ **Theorem 12.** *Let T_1, T_2 be two unambiguous two-way transducers. The following conditions are equivalent:*

1. *$T_1 \subseteq T_2$,*
2. *$T_1 \subseteq_o R(T_2)$ for some resynchronization R,*
3. *$T_1 =_o R(T_2)$ for some 1-bounded Parikh resynchronizer R whose target set coincides with that of T_1 and where, each relation $\mathsf{next}_{\gamma,\gamma'}$ is regular if move_γ and $\mathsf{move}_{\gamma'}$ are regular.*

Proof sketch. We focus on the implication from 1. to 3., as the other implications are trivial. Similarly to the one-way case, to synthesize a resynchronizer, we need to annotate the input with the (unique) successful runs of T_1 and T_2 (if these runs exist). Since T_1, T_2 are two-way, the natural way of doing it is to use *crossing sequences*. Thanks to the encoding of runs by means of crossing sequence, we can describe any output position x with a pair (y, i), where y is the origin of x (according to T_1 or T_2) i is the number of output positions before x with the same origin y. Note that i is bounded, as the transducers here are unambiguous, and hence every input position is visited at most a bounded number of times.

Given $T = T_1$ or $T = T_2$ and an index i, one can construct a unambiguous Parikh automaton $A_{T,i}$ that, when receiving as input a word u with a marked position y, produces the unique position x that is encoded by the pair (y, i), according to the transducer T. It follows that, for every output element correctly annotated with $\gamma = (a, i, j)$, the relation move_γ can be defined as $\{(y, z) \mid A_{T_2,i}(y) - A_{T_1,j}(z) = 0\}$, which is a unambiguous Parikh language. This almost completes the definition of the Parikh resynchronizer R. The remaining components of R consists of suitable relations $\mathsf{next}_{\gamma,\gamma'}$ that check correctness of the annotations. In particular, the relations $\mathsf{next}_{\gamma,\gamma'}$ are obtained by pairing a regular property with properties defined in terms of the prior relations move_γ and $\mathsf{move}_{\gamma'}$, and hence $\mathsf{next}_{\gamma,\gamma'}$ is regular whenever move_γ and $\mathsf{move}_{\gamma'}$ are. ◀

We now explain how to exploit the above characterization to decide bounded regular resynchronizer synthesis problem. We provide the following characterization, whose proof follows from the previous theorem:

▶ **Theorem 13.** *Let T_1, T_2 be two unambiguous two-way transducers such that $T_1 \subseteq T_2$, and let $\hat{R}$ be the bounded Parikh resynchronizer obtained from Theorem 12. The following conditions are equivalent:*

1. *$\hat{R}$ is a regular resynchronizer,*
2. *$T_1 \subseteq_o R(T_2)$ for some bounded regular resynchronizer R,*
3. *$T_1 \subseteq_o R(T_2)$ for some 1-bounded regular resynchronizer R,*
4. *$T_1 =_o R(T_2)$ for some 1-bounded regular resynchronizer R with the same target set as T_1.*

Theorems 12 and 13 together provide a characterization of those pairs of unambiguous two-way transducers T_1, T_2 for which there is a bounded regular resynchronizer R such that $T_1 \subseteq_o R(T_2)$. The effectiveness of this characterization stems from the decidability of regularity of languages recognized by unambiguous Parikh automata [7]. This result requires unambiguity and uses Presburger arithmetics to determine for each (simple) loop a threshold such that iterating the loop more than the threshold always satisfies the Parikh constraint. The language of the Parikh automaton is regular if and only if every (simple) loop has such a threshold. We thus conclude:

▶ **Corollary 14.** *Given two unambiguous two-way transducers T_1, T_2, one can decide whether there is a regular resynchronizer R such that $T_1 \subseteq_o R(T_2)$.*

5 Conclusions

We studied two notions of resynchronization for transducers with origin, called rational resynchronizer and regular resynchronizer. Rational resynchronizers are suited for transforming origin graphs of one-way transducers, while regular resynchronizers can be applied also to origin graphs of two-way transducers. We showed that the former are strictly included in the latter, even when restricting the origin graphs to be one-way. We then studied the following variant of containment problem for transducers: given two transducers T_1, T_2, decide whether $T_1 \subseteq_o R(T_2)$ for some (rational or regular) resynchronizer R. That is, if all origin graphs of T_1 can be seen as some origin graph of T_2 transformed according to R, then compute such a resynchronizer R. This problem can be seen as a synthesis problem of resynchronizers. It is shown that the synthesis problem is decidable when T_1, T_2 are finite-valued one-way transducers and the resynchronizer is constrained to be rational, as well as when T_1, T_2 are unambiguous two-way transducers and the resynchronizer is allowed to be regular (and bounded). In the one-way setting, the problem turns out to be undecidable already for unrestricted (non-functional) transducers and rational resynchronizers. In the two-way setting, the decidability status remains open already when the transducers are not unambiguous (be them functional or not). Concerning this last point, however, we recall that the synthesis problem becomes undecidable as soon as we consider regular resynchronizers that are unbounded, as in this case the problem is at least as hard as classical containment.

References

1 Rajeev Alur and Pavel Cerný. Expressiveness of streaming string transducer. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.

2 Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theor. Comput. Sci.*, 292:45–63, 2003.

3 Meera Blattner and Tom Head. Single-valued a-transducers. *J. Comput. and System Sci.*, 15:310–327, 1977.

4 Mikolaj Bojańczyk. Transducers with origin information. In *International Colloquium on Automata, Languages and Programming (ICALP'14)*, number 8572 in LNCS, pages 26–37. Springer, 2014.

5 Mikolaj Bojańczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle. Which Classes of Origin Graphs Are Generated by Transducers? In *International Colloquium on Automata, Languages and Programming (ICALP'17)*, volume 80 of *LIPIcs*, pages 114:1–114:13, 2017.

6 Sougata Bose, Anca Muscholl, Vincent Penelle, and Gabriele Puppis. Origin-Equivalence of Two-Way Word Transducers Is in PSPACE. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'18)*, volume 122 of *LIPIcs*, pages 1–18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

7 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Unambiguous constrained Automata. *Int. J. Found. Comput. Sci.*, 24(7):1099–1116, 2013. doi:10.1142/S0129054113400339.

8 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

9 Calvin C. Elgot and Jorge E. Mezei. On Relations Defined by Generalized Finite Automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965. doi:10.1147/rd.91.0047.

10 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On Equivalence and Uniformisation Problems for Finite Transducers. In *International Colloquium on Automata, Languages and Programming (ICALP'16)*, volume 55 of *LIPIcs*, pages 125:1–125:14, 2016.

11 Patrick C. Fischer and Arnold L. Rosenberg. Multi-tape one-way nonwriting automata. *J. Comput. and System Sci.*, 2:88–101, 1968.

12 T. V. Griffiths. The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. *J. ACM*, 15(3):409–413, 1968.

13 Piotr Hofman. Personal communication.

14 Oscar H. Ibarra. The unsolvability of the equivalence problem for e-free NGSM's with unary input (output) alphabet and applications. *SIAM J. of Comput.*, 7(4):524–532, 1978.

15 Andreas Weber. Decomposing a k-Valued Transducer into k Unambiguous Ones. *ITA*, 30(5):379–413, 1996.

Acceptance Ambiguity for Quantum Automata

Paul C. Bell
Department of Computer Science, Byrom Street, Liverpool John Moores University,
Liverpool, L3-3AF, UK
p.c.bell@ljmu.ac.uk

Mika Hirvensalo
Department of Mathematics and Statistics, University of Turku, FI-20014, Turku, Finland
mikhirve@utu.fi

Abstract

We consider notions of freeness and ambiguity for the acceptance probability of Moore-Crutchfield Measure Once Quantum Finite Automata (MO-QFA). We study the distribution of acceptance probabilities of such MO-QFA, which is partly motivated by similar freeness problems for matrix semigroups and other computational models. We show that determining if the acceptance probabilities of all possible input words are unique is undecidable for 32 state MO-QFA, even when all unitary matrices and the projection matrix are rational and the initial configuration is defined over real algebraic numbers. We utilize properties of the skew field of quaternions, free rotation groups, representations of tuples of rationals as a linear sum of radicals and a reduction of the mixed modification Post's correspondence problem.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Quantum finite automata, matrix freeness, undecidability, Post's correspondence problem, quaternions

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.70

Funding *Mika Hirvensalo*: Supported by Väisälä Foundation

1 Introduction

Measure-Once Quantum Finite Automata (MO-QFA) were introduced in [24] as a natural quantum variant of probabilistic finite automata. The model is defined formally in Section 3, but briefly a MO-QFA over an alphabet Σ is defined by a three tuple $\mathcal{Q} = (P, \{X_a | a \in \Sigma\}, u)$ where P is a projection matrix, X_a is a complex unitary matrix for each alphabet letter $a \in \Sigma$ and u is a unit length vector with respect to the Euclidean (ℓ^2) norm. Given an input word $w = w_1 \cdots w_k \in \Sigma^*$, then the acceptance probability $f_\mathcal{Q} : \Sigma^* \to \mathbb{R}$ of w under $\mathcal{Q}$ is given by

$$f_\mathcal{Q}(w) = ||PX_{w_k} \cdots X_{w_1} u||^2 .$$

The related model of Probabilistic Finite Automata (PFA) with n states over an alphabet Σ is defined as $\mathcal{P} = (\mathbf{x}, \{M_a | a \in \Sigma\}, \mathbf{y})$ where $\mathbf{y} \in \mathbb{R}^n$ is the initial probability distribution (unit length under ℓ^1 norm); $\mathbf{x} \in \{0,1\}^n$ is the final state vector and each $M_a \in \mathbb{R}^{n \times n}$ is a stochastic matrix. For a word $w = w_1 w_2 \cdots w_k \in \Sigma^*$, we define the acceptance probability $f_\mathcal{P} : \Sigma^* \to \mathbb{R}$ of $\mathcal{P}$ as:

$$f_\mathcal{P}(w) = \mathbf{x}^T M_{w_k} M_{w_{k-1}} \cdots M_{w_1} \mathbf{y}.$$

For any $\lambda \in [0,1]$ and automaton $\mathcal{A}$ (either PFA or QFA) over alphabet Σ, we define a cut-point language to be: $L_{\geq\lambda}(\mathcal{A}) = \{w \in \Sigma^* | f_\mathcal{A}(w) \geq \lambda\}$, and a strict cut-point language $L_{>\lambda}(\mathcal{A})$ by replacing $\geq$ with $>$. The (strict) emptiness problem for a cut-point language is to determine if $L_{\geq\lambda}(A) = \emptyset$ (resp. $L_{>\lambda}(A) = \emptyset$).

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 70; pp. 70:1–70:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The MO-QFA model is very restricted due to unitarity constraints and can recognize only group languages (those regular languages whose syntactic monoid is a group [10]). Whereas the emptiness question for a strict cut-point stochastic languages is undecidable, it surprisingly becomes decidable for MO-QFA [9]. The decidability is established via the compactness of the group generated by unitary matrices: a compact algebraic group has a finite polynomial basis, and the decision procedure is then based on Tarski's quantifier elimination theorem [9].

Another surprising undecidability result was already manifested in [9]: The emptiness problem for non-strict (allowing equality) cut-point languages is undecidable. The sizes of the automata exhibiting undecidability were subsequently improved in [17]. As the aforementioned examples illuminate, the border between decidability and undecidability may be crossed with a minor modification to the model or premises.

Underlying each linear automata model are matrices, which represent the dynamics of the computational model as input symbols are read. For deterministic/nondeterministic finite automata, the underlying matrices are binary matrices; for weighted automata, the matrices are integer; for probabilistic automata the matrices are stochastic (the set of columns of each matrix should be a probability distribution); and for quantum finite automata, the matrices are unitary (the set of columns of each matrix should be orthonormal).

Reachability problems for matrix semigroups have attracted a great deal of attention over the past few years. Typically in such problems we are given a finite set of generating matrices $\mathcal{G}$ forming a semigroup $\mathcal{S} = \langle \mathcal{G} \rangle$ and we ask some question about $\mathcal{S}$. As an example, it was shown even back in 1970 by M. Paterson that the mortality problem for integer matrix semigroups is undecidable in dimension three [25]. In this problem, $\mathcal{G} \subseteq \mathbb{Z}^{3\times 3}$ and we ask whether the zero matrix belongs to $\mathcal{S} = \langle \mathcal{G} \rangle$. It was later shown that the similar identity problem (does the identity matrix belong to the semigroup generated by a given set of generating matrices) is also undecidable for four-dimensional integral matrices [5].

A related problem is the freeness problem for integer matrices – given a set $\mathcal{G} \subseteq \mathbb{F}^{n\times n}$, where $\mathbb{F}$ is a semiring, determine if $\mathcal{G}$ is a code for the semigroup generated by $\mathcal{G}$ (i.e., if every element of $\langle \mathcal{G} \rangle$ has a unique factorization over elements of $\mathcal{G}$). It was proven by Klarner et al. that the freeness problem is undecidable over $\mathbb{N}^{3\times 3}$ in [20] and this result was improved by Cassaigne et al. to hold even for *upper-triangular* matrices over $\mathbb{N}^{3\times 3}$ in [11].

There are many open problems related to freeness in 2×2 matrices; see [12, 13, 14] for good surveys. The freeness problem over $\mathbb{H}^{2\times 2}$ is undecidable [4], where $\mathbb{H}$ is the skew-field of quaternions (in fact the result even holds when all entries of the quaternions are rationals). The freeness problem for two upper-triangular 2×2 rational matrices remains open, despite many partial results being known [13].

The freeness problem for matrix semigroups defined by a *bounded language* was recently studied. Given a finite set of matrices $\{M_1, \ldots, M_k\} \subseteq \mathbb{Q}^{n\times n}$, we define a bounded language of matrices to be of the form: $\{M_1^{j_1} \cdots M_k^{j_k} | j_i \geq 0 \text{ where } 1 \leq i \leq k\}$.

The freeness problem for such a bounded language of matrices asks if there exists a choice of these variables such that $j_1, \ldots, j_k, j'_1, \ldots, j'_k \geq 0$, where at least one $j_i \neq j'_i$ such that $M_1^{j_1} \cdots M_k^{j_k} = M_1^{j'_1} \cdots M_k^{j'_k}$ in which case the bounded language of matrices is not free. This problem was shown to be decidable when $n = 2$, but undecidable in general [13].

In a similar vein, we may study the vector freeness and ambiguity problems, where we are given a finite set of matrices $\mathcal{G} \subseteq \mathbb{F}^{n\times n}$ and a vector $u \in \mathbb{F}^n$. The *ambiguity problem* is to determine whether there exists two matrices $M_1, M_2 \in \mathcal{S} = \langle \mathcal{G} \rangle$ with $M_1 \neq M_2$ such that $M_1 u = M_2 u$ and the *freeness problem* is to determine the uniqueness of factorizations of $\{Mu | M \in \mathcal{S}\}$ i.e., does $M_{i_1} \cdots M_{i_k} u = M_{j_1} \cdots M_{j_{k'}} u$, where each $M_t \in \mathcal{G}$, imply that

$k = k'$ and $M_{i_r} = M_{j_r}$ for $1 \leq r \leq k$? It should be noted that these (related but distinct) problems are more difficult to solve than freeness for matrix semigroups, since by multiplying matrix M_1 and M_2 with u, some information may be lost. The motivation for such a problem is that many linear dynamic systems/computational models are defined in this way. The freeness question now asks whether starting from some initial point, we have two separate computational paths which coincide at some later point, or else whether every configuration starting from u is unique. Such vector ambiguity and freeness questions were studied in [3] and the problems were shown to be undecidable when $\mathcal{S} \subseteq \mathbb{Z}^{4\times4}$, or when $\mathcal{S} \subseteq \mathbb{Q}^{3\times3}$. The NP-completeness of the vector ambiguity and freeness problems for $\mathrm{SL}(2,\mathbb{Z})$ was recently shown in [21] (where $\mathrm{SL}(2,\mathbb{Z})$ is the special linear group of 2×2 matrices).

Whilst vector reachability questions are interesting from the point of view of dynamical systems, many computational models have the notion of a projection being taken at the end, in order to determine whether a computation path is successful or not. This usually takes the form of defining a partition of configurations into accepting or nonaccepting states. This leads to the notion of *scalar reachability* (also known as half-space reachability [15]), which may be defined in terms of *two* vectors, u and v, where we now study the set of scalars $\{u^T Mv | M \in \mathcal{S}\}$. The *scalar ambiguity* question then asks whether or not this set of scalars is unique i.e., does there exist two matrices $M_1, M_2 \in \mathcal{S}$ with $M_1 \neq M_2$ such that $u^T M_1 v = u^T M_2 v$? The difficulty with extending the undecidability result for vector reachability is that all information about each matrix M needs to be stored within a single scalar value $u^T Mv$ in a unique way.

In [1], the freeness problem (defined formally in Section 3.1) for 4-state weighted and 6-state probabilistic automata was shown to be undecidable together with results concerning the related ambiguity problem. The undecidability result was shown to hold even when the input words come from a bounded language, thus the matrices appear in some fixed order, and are taken to an arbitrary power. The problem can also be stated in terms of *formal power series*: given a formal power series r, determine if r has two equal coefficients. This problem was studied in [22] and Theorem 3.4 of [18]. As mentioned above, several reachability problems for PFA (such as emptiness of cut-point languages) are known to be *undecidable* [26], even in a fixed dimension [8, 17]. The reachability problem for PFA defined on a bounded language (i.e., where input words are from a bounded language which is given as part of the input) was shown to be undecidable in [2]. We may note that the scalar freeness and ambiguity problems are a similar concept to the *threshold isolation problem* which asks whether a given cutpoint may be approached arbitrarily closely and which is known to be undecidable [6, 8].

It is therefore natural to ask whether the *freeness and ambiguity problems* are undecidable for MO-QFA. This problem appears more difficult to prove than for weighted/probabilistic automata, since the acceptance probability of a MO-QFA $\mathcal{Q}$ has the form $f_{\mathcal{Q}}(w) = ||PX^R u||^2$ and it is thus difficult to encode sufficient information about the matrix X within $f_{\mathcal{Q}}(w)$ to guarantee uniqueness of matrices from $\mathcal{G}$. We show that freeness and ambiguity are undecidable for 32 (resp. 33) state MO-QFA by using an encoding of the mixed modification Post's Correspondence Problem and a result related to linear independence of rationals of a basis of squarefree radicals as well as techniques from linear algebra and properties of quaternions.

2 Notation

Let $\Sigma = \{x_1, x_2, \dots, x_k\}$ be a finite set of *letters* called an *alphabet*. A word w is a finite sequence of letters from Σ, the set of all words over Σ is denoted Σ^* and the set of nonempty words is denoted Σ^+. The *empty word* is denoted by ε. We use $|u|$ to denote the length of a word u and thus $|\varepsilon| = 0$. For two words $u = u_1u_2 \cdots u_i$ and $v = v_1v_2 \cdots v_j$, where $u, v \in \Sigma^*$, the concatenation of u and v is denoted by $u \cdot v$ (or by uv for brevity) such that $u \cdot v = u_1u_2 \cdots u_iv_1v_2 \cdots v_j$. Word $u^R = u_i \cdots u_2u_1$ denotes the mirror image or reverse of word u. A subset L of Σ^* is called a *language*. A language $L \subseteq \Sigma^*$ is called a *bounded language* if and only if there exist words $w_1, w_2, \dots, w_m \in A^+$ such that $L \subseteq w_1^*w_2^* \cdots w_m^*$. Given an alphabet Σ as above, we denote by Σ^{-1} the set $\{x_1^{-1}, \dots, x_k^{-1}\}$, where each x_i^{-1} is a new letter with the property that $x_ix_i^{-1} = x_i^{-1}x_i = \varepsilon$ are the only identities of the group $\langle \Sigma \cup \Sigma^{-1} \rangle$. A word $w = w_1w_2 \cdots w_i \in (\Sigma \cup \Sigma^{-1})^*$ is called *reduced* if there does not exist $1 \le j < i$ such that $w_{j+1} = w_j^{-1}$; i.e., no two consecutive letters are inverse.

Given any two rings R_1 and R_2 we use the notation $R_1 \hookrightarrow R_2$ to denote a *monomorphism* i.e., an injective homomorphism between R_1 and R_2. Given a finite set $\mathcal{G}$, we use the notation $\langle \mathcal{G} \rangle$ (resp. $\langle \mathcal{G} \rangle_{\text{gp}}$) to denote the *semigroup* (resp. group) generated by $\mathcal{G}$.

Semirings and quaternions

We denote by $\mathbb{N}$ the natural numbers, $\mathbb{Z}$ the integers, $\mathbb{Q}$ the rationals, $\mathbb{C}$ the complex numbers and $\mathbb{H}$ the quaternions. We denote by $\mathbb{C}(\mathbb{Q})$ the complex numbers with rational parts, by $\mathbb{H}(\mathbb{Q})$ the quaternions with rational parts and by $\mathbb{A}_{\mathbb{R}}$ the real algebraic numbers.

Given any semiring $\mathbb{F}$ we denote by $\mathbb{F}^{i \times j}$ the set of $i \times j$ matrices over $\mathbb{F}$. We denote by e_i the i'th basis vector of some dimension (which will be clear from the context).

In a similar style to complex numbers, a rational quaternion $\vartheta \in \mathbb{H}(\mathbb{Q})$ can be written $\vartheta = a+b\mathbf{i}+c\mathbf{j}+d\mathbf{k}$ where $a, b, c, d \in \mathbb{Q}$. To ease notation let us define the vector: $\mu = (1, \mathbf{i}, \mathbf{j}, \mathbf{k})$ and it is now clear that $\vartheta = (a, b, c, d) \cdot \mu$ where $\cdot$ denotes the inner or "dot" product.

Quaternion addition is simply the componentwise addition of elements. It is well known that quaternion multiplication is not commutative (hence they form a skew field). Multiplication is completely defined by the equations $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$, $\mathbf{ij} = \mathbf{k} = -\mathbf{ji}$, $\mathbf{jk} = \mathbf{i} = -\mathbf{kj}$ and $\mathbf{ki} = \mathbf{j} = -\mathbf{ki}$. Thus for two quaternions $\vartheta_1 = (a_1, b_1, c_1, d_1)\mu$ and $\vartheta_2 = (a_2, b_2, c_2, d_2)\mu$, we can define their product as $\vartheta_1\vartheta_2 = (a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2) + (a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)\mathbf{i} + (a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)\mathbf{j} + (a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)\mathbf{k}$.

In a similar way to complex numbers, we define the conjugate of $\vartheta = (a, b, c, d) \cdot \mu$ by $\overline{\vartheta} = (a, -b, -c, -d) \cdot \mu$. We can now define a norm on the quaternions by $||\vartheta|| = \sqrt{\vartheta\overline{\vartheta}} = \sqrt{a^2 + b^2 + c^2 + d^2}$. Any non zero quaternion has a multiplicative (and obviously an additive) inverse [23]. The other properties of being a skew field can be easily checked.

A *unit* quaternion (norm 1) corresponds to a rotation in three dimensional space [23].

Linear Algebra

Given $A = (a_{ij}) \in \mathbb{F}^{m \times m}$ and $B \in \mathbb{F}^{n \times n}$, we define the direct sum $A \oplus B$ and Kronecker product $A \otimes B$ of A and B by:

$$A \oplus B = \left[\begin{array}{c|c} A & \mathbf{0}_{m,n} \\ \hline \mathbf{0}_{n,m} & B \end{array}\right], \quad A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1m}B \\ a_{21}B & a_{22}B & \cdots & a_{2m}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mm}B \end{bmatrix},$$

where $\mathbf{0}_{i,j}$ denotes the zero matrix of dimension $i \times j$. Note that neither $\oplus$ nor $\otimes$ are commutative in general. Given a finite set of matrices $\mathcal{G} = \{G_1, G_2, \ldots, G_m\} \subseteq \mathbb{F}^{n\times n}$, $\langle \mathcal{G} \rangle$ is the semigroup generated by $\mathcal{G}$. We will use the following notations:

$$\bigoplus_{j=1}^{m} G_j = G_1 \oplus G_2 \oplus \cdots \oplus G_m, \qquad \bigotimes_{j=1}^{m} G_j = G_1 \otimes G_2 \otimes \cdots \otimes G_m.$$

Given a single matrix $G \in \mathbb{F}^{n\times n}$, we inductively define $G^{\otimes k} = G \otimes G^{\otimes (k-1)} \in \mathbb{F}^{n^k \times n^k}$ with $G^{\otimes 1} = G$ as the k-fold Kronecker power of G. Similarly, $G^{\oplus k} = G \oplus G^{\oplus (k-1)} \in \mathbb{F}^{n^k \times n^k}$ with $G^{\oplus 1} = G$. The following properties of $\oplus$ and $\otimes$ are well known; see [19] for proofs.

▶ **Lemma 1.** *Let $A, B, C, D \in \mathbb{F}^{n\times n}$. We note that:*

- *$(A \otimes B) \otimes C = A \otimes (B \otimes C)$ and $(A \oplus B) \oplus C = A \oplus (B \oplus C)$, thus $A \otimes B \otimes C$ and $A \oplus B \oplus C$ are unambiguous.*
- *Mixed product properties: $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$ and $(A \oplus B)(C \oplus D) = (AC) \oplus (BD)$.*
- *If A and B are unitary matrices, then so are $A \oplus B$ and $A \otimes B$.*

Given two vectors $u \in \mathbb{F}^{n_1}$ and $v \in \mathbb{F}^{n_2}$, we define $u \oplus v \in \mathbb{F}^{n_1+n_2}$ as $u \oplus v = (u_1, \ldots, u_{n_1}, v_1, \ldots, v_{n_2})$.

3 Quantum Finite Automata and Undecidability

▶ **Definition 2.** *A measure-once n-state quantum automaton (MO-QFA) over a k-letter alphabet Σ is a triplet $(P, \{X_a \mid a \in \Sigma\}, u)$, where $P \in \mathbb{C}^{n\times n}$ is a projection, each $X_a \in \mathbb{C}^{n\times n}$ is a unitary matrix (where rows form an orthonormal set), and $u \in \mathbb{C}^n$ is a unit-length vector.*

A morphism $\Sigma^* \to \langle X_a \rangle$ is defined as $w = a_{i_1} \ldots a_{i_t} \mapsto X_w \overset{def}{=} X_{i_1} \ldots X_{i_t}$ and the acceptance probability *of a MO-QFA $\mathcal{Q}$ is defined as $f_{\mathcal{Q}}(w) = ||PX_{w^R}u||^2$. We use the reverse of the word w, denoted w^R, so that w_1 is applied first, then w_2 etc.*

3.1 Ambiguity and Freeness for QFA

Consider a finite set of unitary matrices $\mathcal{G} = \{X_1, X_2, \ldots, X_k\} \subset \mathbb{C}^{n\times n}$, a projection matrix $P \in \mathbb{Z}^{n\times n}$ and a unit column vector $u \in \mathbb{C}^n$. Let $\mathcal{Q} = (P, \mathcal{G}, u)$ be a QFA and define $\Lambda(\mathcal{Q})$ be the set of scalars such that $\Lambda(\mathcal{Q}) = \{||PXu||^2 ; X \in \langle \mathcal{G} \rangle\}$. If for $\lambda \in \Lambda(\mathcal{Q})$ there exists a unique matrix $X \in \langle \mathcal{G} \rangle$ such that $\lambda = ||PXu||^2$, then we say that λ is *unambiguous* with respect to $\mathcal{Q}$. We call $\Lambda(\mathcal{Q})$ unambiguous if every $\lambda \in \Lambda(\mathcal{Q})$ is unambiguous.

An acceptance probability $\lambda \in \Lambda(\mathcal{Q})$ is called *free* with respect to $\mathcal{Q}$ if

$$\lambda = ||PX_{i_1}X_{i_2}\cdots X_{i_m}u||^2 = \left|\left|PX_{j_1}X_{j_2}\cdots X_{j_{m'}}u\right|\right|^2,$$

where each $X_{i_k}, X_{j_{k'}} \in \mathcal{G}$ for $1 \leq k \leq m$ and $1 \leq k' \leq m'$ implies that $m = m'$ and each $i_k = j_k$ for $1 \leq k \leq m$. We call $\Lambda(\mathcal{Q})$ free if every $\lambda \in \Lambda(\mathcal{Q})$ is free.

▶ **Problem 3** (QFA Scalar Ambiguity). *Given a Quantum Finite Automaton $\mathcal{Q}$, is $\Lambda(\mathcal{Q})$ unambiguous?*

▶ **Problem 4** (QFA Scalar Freeness [1]). *Given a Quantum Finite Automaton $\mathcal{Q}$, is $\Lambda(\mathcal{Q})$ free?*

[1] We may also call this the *injectivity problem* for QFA; does there exist two distinct words $w_1, w_2 \in \Sigma^*$ such that $f_{\mathcal{Q}}(w_1) = f_{\mathcal{Q}}(w_2)$?

▶ **Example 5.** Let $A = \begin{pmatrix} \frac{3}{5} & \frac{4}{5} \\ -\frac{4}{5} & \frac{3}{5} \end{pmatrix}$, $P = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ and $u = (1,0)^T$. We thus see that $\mathcal{Q} = (P, \{A\}, u)$ is a unary 2-state QFA. Note that A represents rotations of the Euclidean plane of angle $\arccos(3/5)$, and thus we see that $f_{\mathcal{Q}}(a^k) = ||PA^k u||^2$ is dense in $[0,1]$ for $k \in \mathbb{N}$. Since the angle of rotation of A is an irrational multiple of π, then every acceptance probability of $\mathcal{Q}$ is unique, and thus $\mathcal{Q}$ is both free and unambiguous.

We show that freeness and ambiguity are undecidable for MO-QFA in Section 5. The reduction is from the Mixed Modification Post's Correspondence Problem, now defined.

▶ **Problem 6** (Mixed Modification PCP (MMPCP)). *Given set of letters $\Sigma = \{s_1, \ldots, s_{|\Sigma|}\}$, binary alphabet Σ_2, and pair of homomorphisms $h, g : \Sigma^* \to \Sigma_2^*$, the MMPCP asks to decide whether there exists a word $w = x_1 \cdots x_k \in \Sigma^+, x_i \in \Sigma$ such that*

$$h_1(x_1)h_2(x_2)\cdots h_k(x_k) = g_1(x_1)g_2(x_2)\cdots g_k(x_k),$$

where $h_i, g_i \in \{h, g\}$, and there exists at least one j such that $h_j \neq g_j$.

▶ **Theorem 7.** *[11] - The Mixed Modification PCP is undecidable for $|\Sigma| \geq 9$.*

▶ **Definition 8.** *We call an instance of the (MM)PCP a* Claus instance *if the minimal solution words are of the form $w = s_1 x_2 x_3 \cdots x_{k-1} s_{|\Sigma|}$, where $x_2, \ldots, x_{k-1} \in \Sigma - \{s_1, s_{|\Sigma|}\}$, i.e., the minimal solution words must start with letter s_1, end with letter $s_{|\Sigma|}$, and all other letters are not equal to s_1 or $s_{|\Sigma|}$.*

In fact most proofs of the undecidability of (MM)PCP have this property [16]. Claus instances can be useful for decreasing the resources required for showing certain undecidability results, and we use this property later.

▶ **Theorem 9.** *[16] - Mixed Modification PCP is undecidable for Claus instances, when $|\Sigma| \geq 9$.*[2]

4 A mapping from arbitrary words to rational unitary matrices

Let $\Sigma_n = \{x_1, x_2, \ldots, x_n\}$ be an n-letter alphabet for some $n > 0$. We begin by deriving a monomorphism $\gamma : \Sigma_n^* \hookrightarrow \mathbb{Q}^{4\times 4}$ such that $\gamma(w)$ is a unitary matrix for any $w \in \Sigma_n^*$. The mapping γ will be a composition of several monomorphisms.

Given alphabet $\Sigma_n = \{x_1, x_2, \ldots, x_n\}$, we now show that there exists a monomorphism $\gamma : \Sigma_n^* \hookrightarrow \mathbb{Q}^{4\times 4}$ where $\gamma(w)$ is unitary for all $w \in \Sigma_n^*$.

We first describe a monomorphism γ_1 from an arbitrary sized alphabet to a binary alphabet. We then show monomorphism γ_2 from a binary alphabet to unit quaternions, and conclude by injectively mapping such quaternions to unitary matrices.

γ_1: Let $\Sigma_2 = \{a, b\}$ be a binary alphabet. We define $\gamma_1 : \Sigma_n^* \hookrightarrow \Sigma_2^*$ by $\gamma_1(x_k) = a^k b$ for $1 \leq k \leq n$. It is immediate that γ_1 is injective.

γ_2: Define mapping $\gamma_2 : \Sigma_2^* \hookrightarrow \mathbb{H}(\mathbb{Q})$ by $\gamma_2(a) = \left(\frac{3}{5}, \frac{4}{5}, 0, 0\right) \cdot \mu$ and $\gamma_2(b) = \left(\frac{3}{5}, 0, \frac{4}{5}, 0\right) \cdot \mu$. It is known that γ_2 is an injective homomorphism [4] since such quaternions represent rotations about perpendicular axes by a rational angle (not equal to $0, \pm\frac{1}{2}, \pm 1$), thus $\gamma_2 : \Sigma_2^* \hookrightarrow \mathbb{H}(\mathbb{Q})$ and $\gamma_2(w_1) = \gamma_2(w_2)$ for $w_1, w_2 \in \Sigma_2^*$ implies that $w_1 = w_2$ [27].

[2] The result in [16] states undecidability for $|\Sigma| \geq 7$ since they fix the first/last letters of a potential solution.

γ_3: Define $\gamma_3 : \mathbb{H}(\mathbb{Q}) \hookrightarrow \mathbb{Q}^{4\times 4}$ by:

$$\gamma_3((r,x,y,z)\cdot\mu) = \begin{pmatrix} r & x & y & z \\ -x & r & z & -y \\ -y & -z & r & x \\ -z & y & -x & r \end{pmatrix}. \tag{1}$$

It is well known that γ_3 is a monomorphism in this case. Injectivity is clear, and using the rules of quaternion multiplication shows that γ_3 is a homomorphism.

We finally define $\gamma = \gamma_3 \circ \gamma_2 \circ \gamma_1$ and thus by the above reasoning $\gamma : \Sigma_n^* \hookrightarrow \mathbb{Q}^{4\times 4}$ is an injective homomorphism. Note that the matrix $\gamma(w)$ for a word $w \in \Sigma_n^*$ contains quite a lot of redundancy, and in fact can be uniquely described by just four elements (the top row) as is shown by the matrix in Eqn. (1). Of course, these four elements simply correspond to the four elements of the quaternion used in the construction of γ. Note also that $\gamma(w)$ is a unitary matrix since γ_2 generates a *unit quaternion* (of norm 1) in each case.

Using γ, we can now find matrices $A, B \in \mathbb{Q}^{4\times 4}$, such that $\gamma(w) \in \langle\{A,B\}\rangle_{\mathrm{gp}}$ for all $w \in \Sigma^*$; i.e., the value of $\gamma(w)$ lies within the semigroup generated by $\{A, B\}$. This will prove useful later since we may reason about the structure of this freely presented semigroup.

▶ **Definition 10.** *Given $\Sigma_2 = \{a, b\}$, then let:*

$$A = \gamma_3(\gamma_2(a)) = \begin{pmatrix} \frac{3}{5} & \frac{4}{5} & 0 & 0 \\ -\frac{4}{5} & \frac{3}{5} & 0 & 0 \\ 0 & 0 & \frac{3}{5} & \frac{4}{5} \\ 0 & 0 & -\frac{4}{5} & \frac{3}{5} \end{pmatrix}, \; B = \gamma_3(\gamma_2(b)) = \begin{pmatrix} \frac{3}{5} & 0 & \frac{4}{5} & 0 \\ 0 & \frac{3}{5} & 0 & -\frac{4}{5} \\ -\frac{4}{5} & 0 & \frac{3}{5} & 0 \\ 0 & \frac{4}{5} & 0 & \frac{3}{5} \end{pmatrix},$$

and define $\Gamma' = \langle\{A,B\}\rangle \subset \mathbb{Q}^{4\times 4}$, which is a free semigroup (freely generated by $\{A, B\}$). All elements in the range of γ thus belong to Γ'. We define $\Gamma \subset \Gamma'$ by $\Gamma = \{\gamma(w) | w \in \Sigma_n^*\}$.

5 Freeness and ambiguity for QFA with radicals

In order to prove that the ambiguity and freeness problems are undecidable for QFA defined over rationals (with real algebraic initial vector), we require the following (folklore) theorem. This will essentially allow us to uniquely represent a tuple of rationals as a linear sum of radicals. For completeness, we will show a simple proof of this theorem using the theory of field extensions.

▶ **Theorem 11** ([7]). *The (finite) set*

$$\mathcal{S} = \{\sqrt{m_1}, \ldots, \sqrt{m_n} : \; m_i \textit{ are coprime square-free numbers}\}$$

is linearly independent over $\mathbb{Q}$.

Proof. Define $E_k = \mathbb{Q}(\sqrt{m_1}, \ldots, \sqrt{m_k})$, so $E_0 = \mathbb{Q}$ and $E_1 = \mathbb{Q}(\sqrt{m_1})$. Clearly $[E_0 : \mathbb{Q}] = 1 = 2^0$, and $[E_1 : \mathbb{Q}] = 2^1$. As each element $\sqrt{m_i}$ satisfies a quadratic equation over $\mathbb{Q}$, the field extension degree $[E_n : \mathbb{Q}]$ is at most 2^n. The theorem is proven if we can show that $[E_n : \mathbb{Q}] = 2^n$.

Assume the induction hypothesis true for values less than k. We will prove it true for $k + 1$, as well, i.e., $[E_{k+1} : E_k] = 2$. For this aim, we must demonstrate that $\sqrt{m_{k+1}} \notin E_k$, so let us assume the contrary, that

$$\sqrt{m_{k+1}} \in E_k = E_{k-1}(\sqrt{m_k}),$$

hence $\sqrt{m_{k+1}} = a + b\sqrt{m_k}$, where $a, b \in E_{k-1}$ Then

$$m_{k+1} = a^2 + m_k b^2 + 2ab\sqrt{m_k}.$$

If $ab \neq 0$, then $\sqrt{m_k} \in E_{k-1}$, which implies that $[E_k : E_{k-1}] = 1$, a contradiction.

If $a = 0$, then $\sqrt{m_{k+1}} = b\sqrt{m_k}$, and hence $\sqrt{m_k}\sqrt{m_{k+1}} = bm_k \in E_{k-1}$. By the induction hypothesis we then have

$$[\mathbb{Q}(\sqrt{m_1}, \dots, \sqrt{m_{k-1}}, \sqrt{m_k m_{k+1}}) : \mathbb{Q}] = 2^k,$$

but since the last extending element belongs to E_{k-1}, the extension degree cannot be more than 2^{k-1}, a contradiction. Here we actually need the assumption that the numbers are coprime, since otherwise $m_k m_{k+1}$ would not necessarily be squarefree.

If $b = 0$, then $\sqrt{m_{k+1}} \in \mathbb{E}_{k-1}$, and as above, the induction hypothesis gives

$$[\mathbb{Q}(\sqrt{m_1}, \dots, \sqrt{m_{k-1}}, \sqrt{m_{k+1}}) : \mathbb{Q}] = 2^k,$$

but as the last extending element belongs to E_{k-1}, the extension degree cannot be more than 2^{k-1}, a contradiction. ◀

For example, given $p_1, p_2, q_1, q_2 \in \mathbb{Q}$, then the equality $p_1\sqrt{2} + q_1\sqrt{3} = p_2\sqrt{2} + q_2\sqrt{3}$ is true iff $p_1 = p_2$ and $q_1 = q_2$.

The following technical lemma concerns the free group $\mathcal{S}$ generated by $\mathcal{G} = \{\gamma_2(a), \gamma_2(b)\}$ and will crucially allow us to characterise elements of $\mathcal{S}$ which differ only in the signs of one or more of their imaginary components. To define this lemma we require a nonstandard inversion function defined on elements of $\mathcal{S} = \langle \mathcal{G} \rangle_{gr}$. Since $\mathcal{S}$ is free, any reduced (i.e., not containing consecutive inverses) $q_w \in \mathcal{S}$ can be uniquely written in the form

$$q_w = \gamma_2(a)^{k_0}\gamma_2(b)^{k_1}\gamma_2(a)^{k_2} \cdots \gamma_2(a)^{k_{n-2}}\gamma_2(b)^{k_{n-1}}\gamma_2(a)^{k_n},$$

where $k_0, k_n \in \mathbb{Z}$ and $k_1, \dots, k_{n-1} \in \mathbb{Z} - \{0\}$, i.e., an alternating product of either positive or negative powers of $\gamma_2(a)$ and $\gamma_2(b)$ which may start and end with either element. We define the following three functions:

i) $\lambda_a(q_w) = \gamma_2(a)^{-k_0}\gamma_2(b)^{k_1}\gamma_2(a)^{-k_2} \cdots \gamma_2(a)^{-k_{n-2}}\gamma_2(b)^{k_{n-1}}\gamma_2(a)^{-k_n}$;

ii) $\lambda_b(q_w) = \gamma_2(a)^{k_0}\gamma_2(b)^{-k_1}\gamma_2(a)^{k_2} \cdots \gamma_2(a)^{k_{n-2}}\gamma_2(b)^{-k_{n-1}}\gamma_2(a)^{k_n}$;

iii) $\lambda_{a,b}(q_w) = \gamma_2(a)^{-k_0}\gamma_2(b)^{-k_1}\gamma_2(a)^{-k_2} \cdots \gamma_2(a)^{-k_{n-2}}\gamma_2(b)^{-k_{n-1}}\gamma_2(a)^{-k_n}$.

These three functions thus invert all $\gamma_2(a)$ elements in a product for λ_a, all $\gamma_2(b)$ elements in a product for λ_b and both $\gamma_2(a)$ and $\gamma_2(b)$ elements in a product for $\lambda_{a,b}$. As an example, if $q_w = \gamma_2(a)^3\gamma_2(b)^2\gamma_2(a)^{-4}\gamma_2(b)$, then $\lambda_a(q_w) = \gamma_2(a)^{-3}\gamma_2(b)^2\gamma_2(a)^4\gamma_2(b)$, $\lambda_b(q_w) = \gamma_2(a)^3\gamma_2(b)^{-2}\gamma_2(a)^{-4}\gamma_2(b)^{-1}$ and $\lambda_{a,b}(q_w) = \gamma_2(a)^{-3}\gamma_2(b)^{-2}\gamma_2(a)^4\gamma_2(b)^{-1}$. Bizzare as such a definition may appear, it allows us to exactly characterize those elements of $\mathcal{S}$ which differ only in the sign of one or more of their imaginary components, as we now show.

▶ **Lemma 12.** *Given a quaternion $q_w = \gamma_2(w) = (r, x, y, z) \cdot \mu \in \langle \gamma_2(a), \gamma_2(b) \rangle_{gr}$ with $w = w_1 w_2 \cdots w_{|w|}$, each $w_i \in (\Sigma_2 \cup \Sigma_2^{-1})$ and $\Sigma_2 = \{a, b\}$, then:*

i) $q_{w^R} = \gamma_2(w^R) = (r, x, y, -z) \cdot \mu$;

ii) $\lambda_a(q_w) = (r, -x, y, -z) \cdot \mu$;

iii) $\lambda_b(q_w) = (r, x, -y, -z) \cdot \mu$;

iv) $\lambda_{a,b}(q_w) = (r, -x, -y, z) \cdot \mu$.

Proof. We proceed via induction. For the base case, when $w = \varepsilon$, then $q_w = (1,0,0,0) \cdot \mu$ and $q_{w^R} = \lambda_a(q_w) = \lambda_b(q_w) = \lambda_{a,b}(q_w) = (1,0,0,0) \cdot \mu$ and so the properties (trivially) hold. For the induction hypothesis, assume $i) - iv)$ are true for q_w. We handle each property individually.

i) By assumption, $q_{w^R} = (r, x, y, -z) \cdot \mu$. Since $\gamma_2(a) = \left(\frac{3}{5}, \frac{4}{5}, 0, 0\right) \cdot \mu$ and $\gamma_2(b) = \left(\frac{3}{5}, 0, \frac{4}{5}, 0\right) \cdot \mu$, by the rules of quaternion multiplication, we see that:

$$\begin{aligned}
\gamma_2(a) \cdot q_w &= \frac{1}{5}(3r - 4x, 3x + 4r, 3y - 4z, 3z + 4y) \cdot \mu, \\
q_{w^R} \cdot \gamma_2(a) &= \frac{1}{5}(3r - 4x, 3x + 4r, 3y - 4z, -3z - 4y) \cdot \mu
\end{aligned}$$

Note that the fourth component is negated as expected. In a similar way, we also see that:

$$\begin{aligned}
\gamma_2(b) \cdot q_w &= \frac{1}{5}(3r - 4y, 3x + 4z, 3y + 4r, 3z - 4x) \cdot \mu, \\
q_{w^R} \cdot \gamma_2(b) &= \frac{1}{5}(3r - 4y, 3x + 4z, 3y + 4r, -3z + 4x) \cdot \mu
\end{aligned}$$

with negated fourth element. Since $\gamma_2(a^{-1}) = \left(\frac{3}{5}, -\frac{4}{5}, 0, 0\right) \cdot \mu$ and $\gamma_2(b^{-1}) = \left(\frac{3}{5}, 0, -\frac{4}{5}, 0\right) \cdot \mu$, then the property of the fourth element being negated is also clearly true for $\gamma_2(c^{-1}) \cdot q_w$ and $q_{w^R} \cdot \gamma_2(c^{-1})$ for $c \in \{a, b\}$. The other properties are similar, we give a brief proof of each.

ii) By the induction hypothesis, $\lambda_a(q_w) = (r, -x, y, -z) \cdot \mu$ and thus:

$$\begin{aligned}
q_w \cdot \gamma_2(a) &= \frac{1}{5}(3r - 4x, 3x + 4r, 3y + 4z, 3z - 4y) \cdot \mu, \\
\lambda_a(q_w) \cdot \gamma_2(a)^{-1} &= \frac{1}{5}(3r - 4x, -3x - 4r, 3y + 4z, -3z + 4y) \cdot \mu,
\end{aligned}$$

with the second and fourth components negated as required. Also,

$$\begin{aligned}
q_w \cdot \gamma_2(a)^{-1} &= \frac{1}{5}(3r + 4x, 3x - 4r, 3y - 4z, 3z + 4y) \cdot \mu, \\
\lambda_a(q_w) \cdot \gamma_2(a) &= \frac{1}{5}(3r + 4x, -3x + 4r, 3y - 4z, -3z - 4y) \cdot \mu,
\end{aligned}$$

as expected. Right multiplication of q_w and $\lambda_a(q_w)$ by either $\gamma_2(b)$ or $\gamma_2(b)^{-1}$ retains the given structure, as is not difficult to calculate.

iii) By the induction hypothesis, $\lambda_b(q_w) = (r, x, -y, -z) \cdot \mu$ and thus:

$$\begin{aligned}
q_w \cdot \gamma_2(b) &= \frac{1}{5}(3r - 4y, 3x - 4z, 3y + 4r, 3z + 4x) \cdot \mu, \\
\lambda_b(q_w) \cdot \gamma_2(b)^{-1} &= \frac{1}{5}(3r - 4y, 3x - 4z, -3y - 4r, -3z - 4x) \cdot \mu,
\end{aligned}$$

with the third and fourth components negated as required. Also,

$$\begin{aligned}
q_w \cdot \gamma_2(b)^{-1} &= \frac{1}{5}(3r + 4y, 3x + 4z, 3y - 4r, 3z - 4x) \cdot \mu, \\
\lambda_b(q_w) \cdot \gamma_2(b) &= \frac{1}{5}(3r + 4y, 3x + 4z, -3y + 4r, -3z + 4x) \cdot \mu,
\end{aligned}$$

as expected. Right multiplication of q_w and $\lambda_b(q_w)$ by either $\gamma_2(a)$ or $\gamma_2(a)^{-1}$ retains the given structure, as is not difficult to calculate.

iv) By the induction hypothesis, $\lambda_{a,b}(q_w) = (r, -x, -y, z) \cdot \mu$ and thus:

$$\begin{aligned}
\lambda_{a,b}(q_w) \cdot \gamma_2(a) &= \frac{1}{5}(3r + 4x, -3x + 4r, -3y + 4z, 3z + 4y) \cdot \mu, \\
\lambda_{a,b}(q_w) \cdot \gamma_2(b) &= \frac{1}{5}(3r + 4y, -3x - 4z, -3y + 4r, 3z - 4x) \cdot \mu, \\
\lambda_{a,b}(q_w) \cdot \gamma_2(a)^{-1} &= \frac{1}{5}(3r - 4x, -3x - 4r, -3y - 4z, 3z - 4y) \cdot \mu, \\
\lambda_{a,b}(q_w) \cdot \gamma_2(b)^{-1} &= \frac{1}{5}(3r - 4y, -3x + 4z, -3y - 4r, 3z + 4x) \cdot \mu,
\end{aligned}$$

with the second and third components of each product negated with relation to $q_w \cdot \gamma_2(a)^{-1}$, $q_w \cdot \gamma_2(b)^{-1}$, $q_w \cdot \gamma_2(a)$ and $q_w \cdot \gamma_2(b)$ (resp.) as required. ◀

The following lemma allows us to represent a quaternion (and its corresponding rotation matrix) by using only absolute values and will be crucial later.

▶ **Lemma 13.** *Given a word $w \in \Sigma_k^*$, then $\gamma_2(\gamma_1(w)) = (r, x, y, z) \cdot \mu$ is uniquely determined by $(|r|, |x|, |y|, |z|)$. All matrices $\gamma(w) \in \Gamma$ are similarly uniquely determined by*

$$(|\gamma(w)_{1,1}|, |\gamma(w)_{1,2}|, |\gamma(w)_{1,3}|, |\gamma(w)_{1,4}|),$$

i.e., by the absolute values of each element of the top row of the matrix.

Proof. Another way to state this Lemma is that if we have $u = u_1 u_2 \cdots u_t$ and $v = v_1 v_2 \cdots v_{t'}$ with each $u_i, v_i \in \Sigma_k^*$, such that $\gamma_2(\gamma_1(u)) = (a_1, b_1, c_1, d_1) \cdot \mu$, $\gamma_2(\gamma_1(v)) = (a_2, b_2, c_2, d_2) \cdot \mu$ and $(|a_1|, |b_1|, |c_1|, |d_1|) = (|a_2|, |b_2|, |c_2|, |d_2|)$, then $t = t'$ and $u_i = v_i$ for all $1 \leq i \leq t$. A similar property holds for the top row of the unitary matrices when applying γ_3 to these elements. We shall now prove this.

By definition, $\gamma_2 : \Sigma_2^* \hookrightarrow \mathbb{H}(\mathbb{Q})$ maps to a free monoid $\mathcal{S}$ of $\mathbb{H}(\mathbb{Q})$ generated by $\mathcal{G} = \{\gamma_2(a), \gamma_2(b)\}$ with $\gamma_2(a) = \left(\frac{3}{5}, \frac{4}{5}, 0, 0\right) \cdot \mu$ and $\gamma_2(b) = \left(\frac{3}{5}, 0, \frac{4}{5}, 0\right) \cdot \mu$. As shown in Section 4, $\gamma_2 \circ \gamma_1 : \Sigma_n^* \hookrightarrow \mathbb{H}(\mathbb{Q})$; i.e., $\gamma_2 \circ \gamma_1$ is an injective homomorphism. Let $\Gamma' = \{\gamma_2(\gamma_1(w')) | w' \in \Sigma_n^*\} \subseteq \mathbb{H}(\mathbb{Q})$. Clearly then, Γ' is freely generated by $\{\gamma_2(\gamma_1(w')) | w' \in \Sigma_n\}$ by the injectivity of $\gamma_2 \circ \gamma_1$.

Let $q_w = \gamma_2(\gamma_1(w)) = (r, x, y, z) \cdot \mu \in \Gamma' \subseteq \mathcal{S}$ and define $Q_w = \{(\pm r, \pm x, \pm y, \pm z) \cdot \mu\}$, thus $|Q_w| = 16$. We will now show that for all $q' \in Q_w - \{q_w\}$ then $q' \notin \Gamma'$ which proves the lemma.

Since (unit) quaternion inversion simply involves negating all imaginary components, then using the identities of Lemma 12, we can derive that $q_w^{-1} = (r, -x, -y, -z)$, $\lambda_a(q_w)^{-1} = (r, x, -y, z)$ and $\lambda_b(q_w)^{-1} = (r, -x, y, z)$ which we summarize in the following table.

q_w	$(r, x, y, z)\mu$	q_w^{-1}	$(r, -x, -y, -z)\mu$
$\lambda_a(q_w)$	$(r, -x, y, -z)\mu$	$\lambda_a(q_w)^{-1}$	$(r, x, -y, z)\mu$
$\lambda_b(q_w)$	$(r, x, -y, -z)\mu$	$\lambda_b(q_w)^{-1}$	$(r, -x, y, z)\mu$
$\lambda_{a,b}(q_w)$	$(r, -x, -y, z)\mu$	q_{w^R}	$(r, x, y, -z)\mu$

We might also notice other identites, such as $q_{w^R} = \lambda_{a,b}(q_w)^{-1}$ which is clear from the definition of $\lambda_{a,b}$. Note that this table covers 8 elements of Q_w.

Note q_w belongs (by definition) to $\Gamma' = (\gamma_2(a)^+ \gamma_2(b))^+ = \{\gamma_2(\gamma_1(w')) | w' \in \Sigma_n\} \subseteq \mathcal{S}$. Since $\langle \gamma_2(a), \gamma_2(b) \rangle_{gr}$ generates a free group, this means that no reduced element of $\mathcal{S}$ is equal to a product with a nontrivial[3] factor $\gamma_2(a)^{-1}$ or $\gamma_2(b)^{-1}$. Each element in the above

[3] Reduced meaning the element contains no consecutive inverse elements and nontrivial meaning we ignoring any such element adjacent to its multiplicative inverse.

table contains at least one nonreducible factor $\gamma_2(a)^{-1}$ or $\gamma_2(b)^{-1}$, excluding q_w and q_{w^R}. Note however that q_{w^R} trivially does not belong to $\Gamma' = (\gamma_2(a)^+\gamma_2(b))^+$ since it necessarily begins with nonreducible factor $\gamma_2(b)$.

Finally, to cover the remaining 8 elements of Q_w, we consider the *free group* $\mathcal{S}_{\rm gr} = \langle\{\gamma_2(a), \gamma_2(b)\}|\emptyset\rangle_{gr}$. For any $q'_w \in \mathcal{S}_{\rm gr}$ then $-q'_w \notin \mathcal{S}_{\rm gr}$ since $\mathcal{S}_{\rm gr}$ is free. This holds since if $-q'_w \in \mathcal{S}$, then $-1 \in \mathcal{S}$ (because $(q'_w)^{-1} \in \mathcal{S}$), which gives a nontrivial identity $-1^2 = 1$ in $\mathcal{S}_{\rm gr}$ (a contradiction).

This covers all sixteen possible elements of Q_w and shows that q_w is the only member of Q_w belonging to Γ'. By the definition of $\gamma_3 : \mathbb{H}(\mathbb{Q}) \hookrightarrow \mathbb{Q}^{4\times4}$, then also all matrices $\gamma(w) \in \Gamma$ are uniquely determined by $(|\gamma(w)_{1,1}|, |\gamma(w)_{1,2}|, |\gamma(w)_{1,3}|, |\gamma(w)_{1,4}|)$ as required. ◀

▶ **Theorem 14.** *The freeness problem for measure-once quantum finite automata is undecidable for* 32 *states over an alphabet of size* 17*.*

Proof. We will encode an instance (h, g) of the mixed modification Post's Correspondence Problem into a finite set of matrices so that if there exists a solution to the instance then there exists some scalar which is nonfree, otherwise every scalar is free.

Let $\Sigma = \{x_1, x_2, \ldots, x_{n-2}\}$ and $\Delta = \{x_{n-1}, x_n\}$ be distinct alphabets and $h, g : \Sigma^* \to \Delta^*$ be an instance of the mixed modification PCP and let $\Sigma_n = \Sigma \cup \Delta$. The naming convention will become apparent below, but intuitively we will be applying γ, from Section 4 to both the input and output alphabets.

Recall that we showed the injectivity of γ in Section 4, and thus have a monomorphism $\gamma : \Sigma_n^* \hookrightarrow \mathbb{Q}^{4\times4}$. We define a function $\varphi : \Sigma_n^* \times \Sigma_n^* \hookrightarrow \mathbb{Q}^{32\times32}$ by

$$\varphi(w_1, w_2) = \bigoplus_{j=1}^{4} \gamma(w_1) \oplus \bigoplus_{j=1}^{4} \gamma(w_2).$$

We may note that $\varphi(w_1, w_2)$ remains a unitary matrix since $\gamma(w_i)$ is unitary and the direct sum of unitary matrices is unitary. Define $\mathcal{G} = \{\varphi(x_i, h(x_i)), \varphi(x_i, g(x_i))|x_i \in \Sigma\} \subset \mathbb{Q}^{32\times32}$.

Let p_i denote the i'th prime number and define $u_i = \sqrt[4]{p_i} \cdot e_i \in {\mathbb{A}_\mathbb{R}}^4$, $v_i = \sqrt[4]{p_{4+i}} \cdot e_i \in {\mathbb{A}_\mathbb{R}}^4$ for $1 \leq i \leq 4$ (${\mathbb{A}_\mathbb{R}}^4$ denotes a 4-tuple of elements from $\mathbb{A}_\mathbb{R}$) and $u' = \bigoplus_{j=1}^4 u_j \oplus \bigoplus_{j=1}^4 v_j \in {\mathbb{A}_\mathbb{R}}^{32}$. Now, we normalise this vector so that $u = \frac{u'}{\sqrt{\sum_{i=1}^8 \sqrt{p_i}}} \in {\mathbb{A}_\mathbb{R}}^{32}$, with u a unit vector. Note that each element of u is a real algebraic number. Let $P_1 = 1 \oplus \mathbf{0}_3$ where $\mathbf{0}_3$ is the 3×3 zero matrix, thus P_1 has a 1 in the upper left element and zero elsewhere. Then define $P = P_1^{\oplus 8} \in \mathbb{Q}^{32\times32}$. Note that $P^2 = P$ and P is a projection matrix.

We are now ready to define our QFA $\mathcal{Q}$ by the triple $\mathcal{Q} = (P, \mathcal{G}, u)$ and prove the claim of the theorem.

Let $X = X_{i_1} \cdots X_{i_p} = \varphi(x_{i_1}, f_{i_1}(x_{i_1})) \cdots \varphi(x_{i_p}, f_{i_p}(x_{i_p}))$, with $f_{i_k} \in \{g, h\}$ for $1 \leq k \leq p$ be one factorization of a matrix $X \in \mathcal{G}$. Define $x = x_{i_1} \cdots x_{i_p}$ and $f(x) = f_{i_1}(x_{i_1}) \cdots f_{i_p}(x_{i_p})$. Then we see that:

$$
\begin{aligned}
||PXu||^2 &= \left\| \frac{\bigoplus_{j=1}^4 (P_1\gamma(x)u_j) \oplus \bigoplus_{j=1}^4 (P_1\gamma(f(x))v_j)}{\sqrt{\sum_{i=1}^8 \sqrt{p_i}}} \right\|^2 && (2)\\
&= \left\| \frac{\bigoplus_{j=1}^4 (P_1\gamma(x)\sqrt[4]{p_j}\cdot e_j) \oplus \bigoplus_{j=1}^4 (P_1\gamma(f(x))\sqrt[4]{p_{4+j}}\cdot e_j)}{\sqrt{\sum_{i=1}^8 \sqrt{p_i}}} \right\|^2 && (3)\\
&= \left(\sqrt{\frac{\sum_{j=1}^4 (\gamma(x)_{1,j}\sqrt[4]{p_j})^2 + \sum_{j=1}^4 (\gamma(f(x))_{1,j}\sqrt[4]{p_{4+j}})^2}{\sqrt{\sum_{i=1}^8 \sqrt{p_i}}}} \right)^2 && (4)\\
&= \frac{\sum_{j=1}^4 \gamma(x)_{1,j}^2\sqrt{p_j} + \sum_{j=1}^4 \gamma(f(x))_{1,j}^2\sqrt{p_{4+j}}}{\sqrt{\sum_{i=1}^8 \sqrt{p_i}}}. && (5)
\end{aligned}
$$

Assume that matrix X has two distinct factorizations $X = X_{i_1}\cdots X_{i_p} = X_{j_1}\cdots X_{j_q} \in \mathcal{G}^+$ and $p \neq q$ or $X_{i_k} \neq X_{j_k}$ for some $1 \leq k \leq p$, such that

$$||PXu||^2 = \left|\left|PX_{i_1}\cdots X_{i_p}u\right|\right|^2 = \left|\left|PX_{j_1}\cdots X_{j_q}u\right|\right|^2,$$

and thus $\Lambda(\mathcal{Q})$ is not free. Let $X = X_{j_1}\cdots X_{j_q} = \varphi(x_{j_1}, f'_{j_1}(x_{j_1}))\cdots\varphi(x_{j_q}, f'_{j_q}(x_{j_q}))$, with $f'_{j_k} \in \{g,h\}$ for $1 \leq k \leq q$ and define $x' = x_{j_1}\cdots x_{j_q}$ and $f'(x') = f'_{j_1}(x_{j_1})\cdots f'_{j_q}(x_{j_p})$ with each $f'_{j_k} \in \{g,h\}$. Note that in Eqn. (5) the denominator is constant and thus when determining equality $\left|\left|PX_{i_1}\cdots X_{i_p}u\right|\right|^2 = \left|\left|PX_{j_1}\cdots X_{j_q}u\right|\right|^2$ we may ignore it. By Lemma 13, each $\gamma(w)$ is uniquely determined by the absolute value of the top four elements of the matrix (e.g. $|\gamma(w)_{1,j}|$ for $1 \leq j \leq 4$). Since each p_j is squarefree, for $1 \leq j \leq 8$, then by Theorem 11, the following equation is satisfied if and only if $|\gamma(x)| = |\gamma(x')|$ and $|\gamma(f(x))| = |\gamma(f'(x'))|$:

$$
\begin{aligned}
&\sum_{j=1}^4 \gamma(x)_{1,j}^2\sqrt{p_j} + \sum_{j=1}^4 \gamma(f(x))_{1,j}^2\sqrt{p_{4+j}}\\
= \;&\sum_{j=1}^4 \gamma(x')_{1,j}^2\sqrt{p_j} + \sum_{j=1}^4 \gamma(f'(x'))_{1,j}^2\sqrt{p_{4+j}}.
\end{aligned}
$$

Finally, note that $\gamma(x) = \gamma(x')$ if and only if $x = x'$. As before, let $x = x_{i_1}\cdots x_{i_p}$, then $\gamma(f(x)) = f_{i_1}(x_{i_1})\cdots f_{i_p}(x_{i_p}) = f_{j_1}(x_{i_1})\cdots f_{j_p}(x_{i_p}) = \gamma(f'(x'))$ with some $f_{i_k} \neq f_{j_k}$ for $1 \leq k \leq p$ if and only if the instance of the MMPCP has a solution.

If the MMPCP is undecidable for Claus instances with an alphabet of size n' (see Theorem 9), then the undecidability of the current theorem holds for $|\mathcal{G}| \geq 2n'$. We now prove that the result holds for $|\mathcal{G}| \geq 2n' - 1$. Let $\Sigma = \{x_1, \ldots, x_{n'}\}$. Since h, g is a Claus instance, any solution word w is of the form $w = x_1 w' x_{n'}$, with $w' \in (\Sigma - \{x_1, x_{n'}\})^*$. By symmetry, we may assume that $h_1 = h$ and by the proof in [16], $g_i = g$ and $h_i = h$ for all $1 \leq i \leq t$. Clearly then, one of $h(x_{n'})$ and $g(x_{n'})$ is a proper suffix of the other (assume that $g(x_{n'})$ is a suffix of $h(x_{n'})$; the opposite case is similar). Now, redefine $u' = \gamma(x_{n'}, g(x_{n'}))u$, remove the matrix corresponding to $g(x_{n'})$ from $\mathcal{G}$ and redefine the matrix corresponding to $h(x_{n'})$ by $h'(x_{n'}) = \gamma(x_{n'}, h(x_{n'})g(x_{n'})^{-1})$. Since $g(x_{n'})$ is a proper suffix of $h(x_{n'})$, then $h(x'_n)g(x'_n)^{-1}$ is the prefix of $h(x_{n'})$ after removing the common suffix with $g(x_{n'})$. This means that an ambiguous scalar only exists if there exists a solution to the instance of MMPCP and we had reduced the alphabet size by 1. MMPCP is undecidable for instances of size 9 (Theorem 9), thus the undecidability holds for MO-QFA with 32 states and an alphabet size of 17. ◀

▶ **Corollary 15.** *The ambiguity problem for measure-once quantum finite automata is undecidable for* 33 *states over an alphabet of size* 17*.*

Proof. The corollary follows from the proof of Theorem 14. We notice that if there exists a solution to the encoded instance of the MMPCP, then some matrix X has two distinct factorizations over $\mathcal{G}$ and therefore there exists two distinct matrix products giving the same scalar. Our technique in this corollary is to make these two factorizations produce distinct matrices X_1 and X_2, such that they still lead to the same scalar. This is simple to accomplish by redefining the projection matrix P as $P' = P \oplus 0$, redefining the initial vector u as $u' = u \oplus 0$ and for each matrix $M \in \mathcal{G} - \{\varphi(x_1, h(x_1))\}$, we redefine M as $M' = M \oplus 1$ and let $\varphi(x_1, h(x_1))$ be redefined as $\varphi(x_1, h(x_1)) \oplus -1$. In this case, any matrix product containing $\varphi(x_1, h(x_1)) \oplus -1$ will have -1 in the bottom right element, otherwise the bottom right element is 1. Since we encode a Claus instance of MMPCP, one factorization has -1 in this case, and the other has 1, and thus we always have distinct matrices. If no solution exists, then each matrix leads to a unique scalar anyway.

Note that we increased the number of states of the MO-QFA by 1 and also note that the acceptance probability is unaffected by the above modifications since the projection matrix was increased by a zero row/column. ◀

6 Conclusion

An interesting question is whether Theorem 14 can be shown to hold when the initial vector is rational, rather than real algebraic. We can prove this result if a certain open problem related to rational packing functions holds (does there exist a polynomial which maps n-tuples of rationals to a single rational injectively). Such a function is well known for integer values (the Cantor polynomial), but not for rational n-tuples. This seems a difficult problem to approach however, and thus we leave the following open problem.

▶ **Open Problem 16.** *Can undecidability of the ambiguity and freeness problems for MO-QFA be shown when the initial vector, projection matrix and all unitary matrices are over rationals?*

We also note that in [1] the ambiguity and freeness problems for weighted finite automata and probabilistic finite automata were shown to be undecidable even when the input words were restricted to come from a given *letter monotonic language*, which is a restriction of bounded languages of the form $x_1^* x_2^* \cdots x_k^*$ where each x_i is a single letter of the input alphabet. The undecidability result of [1] used an encoding of Hilbert's tenth problem, which seems difficult to encode into unitary matrices and thus we pose the following open problem.

▶ **Open Problem 17.** *Can the undecidability of the ambiguity and freeness problems for MO-QFA be shown when the input word is necessarily from a given letter monotonic language?*

References

1 P. C. Bell, S. Chen, and L. M. Jackson. Scalar ambiguity and freeness in matrix semigroups over bounded languages. In *Language and Automata Theory and Applications*, volume LNCS 9618, pages 493–505, 2016.

2 P. C. Bell, V. Halava, and M. Hirvensalo. Decision Problems for Probabilistic Finite Automata on Bounded Languages. *Fundamenta Informaticae*, 123(1):1–14, 2012.

3 P. C. Bell and I. Potapov. Periodic and infinite traces in matrix semigroups. *Current Trends in Theory and Practice of Computer Science (SOFSEM)*, LNCS 4910:148–161, 2008.

4 P. C. Bell and I. Potapov. Reachability problems in quaternion matrix and rotation semigroups. *Information and Computation*, 206(11):1353–1361, 2008.

5 P. C. Bell and I. Potapov. On the undecidability of the identity correspondence problem and its applications for word and matrix semigroups. *International Journal of Foundations of Computer Science*, 21(6):963–978, 2010.

6 A. Bertoni, G. Mauri, and M. Torelli. Some recursively unsolvable problems relating to isolated cutpoints in probabilistic automata. In *Automata, Languages and Programming*, volume 52 of *LNCS*, pages 87–94, 1977.

7 A. S. Besicovitch. On the linear independence of fractional powers of integers. *J. London Math. Soc.*, 15:3–6, 1940.

8 V. Blondel and V. Canterini. Undecidable problems for probabilistic automata of fixed dimension. *Theory of Computing Systems*, 36:231–245, 2003.

9 V. Blondel, E. Jeandel, P. Koiran, and N. Portier. Decidable and undecidable problems about quantum automata. *SIAM Journal on Computing*, 34:6:1464–1473, 2005.

10 A. Brodsky and N. Pippenger. Characterizations of 1-way quantum finite automata. *SIAM Journal on Computing*, 31:1456–1478, 2002.

11 J. Cassaigne, T. Harju, and J. Karhumäki. On the undecidability of freeness of matrix semigroups. *International Journal of Algebra and Computation*, 9(3-4):295–305, 1999.

12 J. Cassaigne and F. Nicolas. On the decidability of semigroup freeness. *RAIRO - Theoretical Informatics and Applications*, 46(3):355–399, 2012.

13 É. Charlier and J. Honkala. The freeness problem over matrix semigroups and bounded languages. *Information and Computation*, 237:243–256, 2014.

14 C. Choffrut and J. Karhumäki. Some decision problems on integer matrices. *Informatics and Applications*, 39:125–131, 2005.

15 T. Colcombet, J. Ouaknine, P. Semukhin, and J. Worrell. On reachability problems for low dimensional matrix semigroups. In *ArXiV Manuscript (to appear ICALP'19)*, volume arXiv:1902.09597, pages 1–15, 2019.

16 V. Halava, T. Harju, and M. Hirvensalo. Undecidability bounds for integer matrices using Claus instances. *International Journal of Foundations of Computer Science (IJFCS)*, 18,5:931–948, 2007.

17 M. Hirvensalo. Improved undecidability results on the emptiness problem of probabilistic and quantum cut-point languages. *SOFSEM 2007: Theory and Practice of Computer Science, Lecture Notes in Computer Science*, 4362:309–319, 2007.

18 J. Honkala. Decision problems concerning thinness and slenderness of formal languages. In *Acta Informatica*, volume 35, pages 625–636, 1998.

19 R. A. Horn and C. R. Johnson. *Topics in matrix analysis.* Cambridge University Press, 1991.

20 D. A. Klarner, J.-C. Birget, and W. Satterfield. On the undecidability of the freeness of integer matrix semigroups. *International Journal of Algebra and Computation*, 1 (2):223–226, 1991.

21 S.-K. Ko and I. Potapov. Vector ambiguity and freeness problems in SL(2, $\mathbb{Z}$). *Fandumenta Informaticae*, 162(2-3):161–182, 2018.

22 W. Kuich and A. Salomaa. *Semirings, Automata, Languages*, volume 5. Springer, 1986.

23 E. Lengyel. *Mathematics for 3D Game Programming & Computer Graphics.* Charles River Media, 2004.

24 C. Moore and J. P. Crutchfield. Quantum automata and quantum grammars. *Theoretical Computer Science*, 237(1-2):275–306, 2000.

25 M. S. Paterson. Unsolvability in 3×3 matrices. *Studies in Applied Mathematics*, 49(1):105–107, 1970.

26 A. Paz. *Introduction to Probabilistic Automata.* Academic Press, 1971.

27 S. Swierczkowski. A class of free rotation groups. *Indag. Math.*, 5(2):221–226, 1994.

From Regular Expression Matching to Parsing

Philip Bille
Technical University of Denmark, DTU Compute, Denmark
phbi@dtu.dk

Inge Li Gørtz
Technical University of Denmark, DTU Compute, Denmark
inge@dtu.dk

Abstract

Given a regular expression R and a string Q, the regular expression parsing problem is to determine if Q matches R and if so, determine how it matches, e.g., by a mapping of the characters of Q to the characters in R. Regular expression parsing makes finding matches of a regular expression even more useful by allowing us to directly extract subpatterns of the match, e.g., for extracting IP-addresses from internet traffic analysis or extracting subparts of genomes from genetic data bases. We present a new general techniques for efficiently converting a large class of algorithms that determine if a string Q matches regular expression R into algorithms that can construct a corresponding mapping. As a consequence, we obtain the first efficient linear space solutions for regular expression parsing.

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

Keywords and phrases regular expressions, finite automata, regular expression parsing, algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.71

Related Version A full version of the paper is available at `https://arxiv.org/abs/1804.02906`

Funding Supported by the Danish Research Council (DFF – 4005-00267, DFF – 1323-00178)

1 Introduction

A regular expression specifies a set of strings formed by characters combined with concatenation, union (`|`), and Kleene star (`*`) operators. For instance, `(a|(ba))*)` describes the set of strings of `a`s and `b`s, where every `b` is followed by an `a`. Regular expressions are a fundamental concept in formal language theory and a basic tool in computer science for specifying search patterns. Regular expression search appears in diverse areas such as internet traffic analysis [14, 27, 17], data mining [11], data bases [19, 21], computational biology [23], and human computer interaction [16].

Given a regular expression R and a string Q, the *regular expression parsing problem* [15, 8, 10, 24, 25, 18] is to determine if Q matches a string in the set of strings described by R and if so, determine how it matches by computing the corresponding sequence of positions of characters in R, i.e., the mapping of each character in Q to a character in R corresponding to the match. For instance, if $R =$ `(a|(ba))*)` and $Q =$ `aaba`, then Q matches R and $1, 1, 2, 3$ is a corresponding parse specifying that $Q[1]$ and $Q[2]$ match the first `a` in R, $Q[3]$ match the `b` in R, and $Q[4]$ match the last `a` in R[1]. Regular expression parsing makes finding matches of a regular expression even more useful by allowing us to directly extract subpatterns of the match, e.g., for extracting IP-addresses from internet traffic analysis or extracting subparts of genomes from genetic data bases.

[1] Another typical definition of parsing is to compute a parse tree (or a variant thereof) of the derivation of Q on R. Our definition simplifies our presentation and it is straightforward to derive a parse tree from our parses.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 71; pp. 71:1–71:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

To state the existing bounds, let n and m be the length of the string and the regular expression, respectively. As a starting point consider the simpler *regular expression matching problem*, that is, to determine if Q matches a string in the set of strings described by R (without necessarily providing a mapping from characters in Q to characters in R). A classic textbook algorithm to matching, due to Thompson [26], constructs and simulates a non-deterministic finite automaton (NFA) in $O(nm)$ time and $O(m)$ space. An immediate approach to solve the parsing problem is to combine Thompson's algorithm with backtracking. To do so, we store all state-sets produced during the NFA simulation and then process these in reverse order to recover an accepting path in the NFA matching Q. From the path we then immediately obtain the corresponding parse of Q since each transition labeled by a character uniquely corresponds to a character in R. This algorithm uses $O(nm)$ time and space. Hence, we achieve the same time bound as matching but increase the space by an $\Omega(n)$ factor. We can improve the time by polylogarithmic factors using faster algorithms for matching [22, 3, 4, 6, 7], but by a recent conditional lower bound [2] we cannot hope to achieve $\Omega((nm)^{1-\varepsilon})$ time assuming the strong exponential time hypothesis. Other direct approaches to regular expression parsing [15, 8, 10, 24, 25, 18] similarly achieve $\Theta(nm)$ time and space (ignoring polylogarithmic factors), leaving a substantial gap between linear space for matching and $\Theta(nm)$ space for parsing. The goal of this paper is to address this gap.

1.1 Results

We present a new technique to efficiently extend the classic state-set transition algorithms for matching to also solve parsing in the same time complexity while only using linear space. Specifically, we obtain the following main result based on Thompson's algorithm:

▶ **Theorem 1.** *Given a regular expression of length m and a string of length n, we can solve the regular expression parsing problem in $O(nm)$ time and $O(n+m)$ space.*

This is the first bound to significantly improve upon the combination of $\Theta(nm)$ time and space. The result holds on a comparison-based, pointer machine model of computation. Our techniques are sufficiently general to also handle the more recent faster state-set transition algorithms [22, 4, 3] and we also obtain a similar space improvement for these.

1.2 Techniques

Our overall approach is similar in spirit to the classic divide and conquer algorithm by Hirschberg [13] for computing a longest common subsequence of two strings in linear space. Let A be the *Thompson NFA* (TNFA) for R built according to Thompson's rules [26] (see also Figure 1) with m states, and let Q be the string of length n.

We first decompose A using standard techniques into a pair of nested subTNFAs called the *inner subTNFA* and the *outer subTNFA*. Each have roughly at most $2/3$ of the states of A and overlap in at most 2 boundary states. We then show how to carefully simulate A to decompose Q into substrings corresponding to subparts of an accepting path in each of the subTNFAs. The key challenge here is to efficiently handle cyclic dependencies between the subTNFAs. From this we construct a sequence of subproblems for each of the substrings corresponding to the inner subTNFAs and a single subproblem for the outer subTNFA. We recursively solve these to construct a complete accepting path in A. This strategy leads to an $O(nm)$ time and $O(n \log m + m)$ space solution. We show how to tune and organize the recursion to avoid storing intermediate substrings leading to the linear space solution in Theorem 1. Finally, we show how to extend our solution to obtain linear space parsing solutions for other state-set transition algorithms.

2 Preliminaries

Strings. A string Q of length $n = |Q|$ is a sequence $Q[1] \dots Q[n]$ of n characters drawn from an alphabet Σ. The string $Q[i] \dots Q[j]$ denoted $Q[i, j]$ is called a substring of Q. The substrings $Q[1, i]$ and $Q[j, n]$ are the ith prefix and the jth suffix of Q, respectively. The string ϵ is the unique empty string of length zero.

Regular Expressions. First we briefly review the classical concepts used in the paper. For more details see, e.g., Aho et al. [1]. We consider the set of non-empty regular expressions over an alphabet Σ, defined recursively as follows. If $\alpha \in \Sigma \cup \{\epsilon\}$ then α is a regular expression, and if S and T are regular expressions then so is the *concatenation*, $(S) \cdot (T)$, the *union*, $(S)|(T)$, and the *star*, $(S)^*$. The *language* $L(R)$ generated by R is defined as follows. If $\alpha \in \Sigma \cup \{\epsilon\}$, then $L(\alpha)$ is the set containing the single string α. If S and T are regular expressions, then $L(S \cdot T) = L(S) \cdot L(T)$, that is, any string formed by the concatenation of a string in $L(S)$ with a string in $L(T)$, $L(S)|L(T) = L(S) \cup L(T)$, and $L(S^*) = \bigcup_{i \geq 0} L(S)^i$, where $L(S)^0 = \{\epsilon\}$ and $L(S)^i = L(S)^{i-1} \cdot L(S)$, for $i > 0$. The *parse tree* $\mathcal{T}^P(R)$ of R (not to be confused with the parse of Q wrt. to R) is the rooted, binary tree representing the hierarchical structure of R. The leaves of $\mathcal{T}^P(R)$ are labeled by a character from Σ or ϵ and internal nodes are labeled by either $\cdot$, $|$, or $*$.

Finite Automata. A *finite automaton* is a tuple $A = (V, E, \Sigma, \theta, \phi)$, where V is a set of nodes called *states*, E is a set of directed edges between states called *transitions* either labeled ϵ (called ϵ-transitions) or labeled by a character from Σ (called character-transitions), $\theta \in V$ is a *start state*, and $\phi \in V$ is an *accepting state*[2]. In short, A is an edge-labeled directed graph with a special start and accepting node. A is a *deterministic finite automaton* (DFA) if A does not contain any ϵ-transitions, and all outgoing transitions of any state have different labels. Otherwise, A is a *non-deterministic automaton* (NFA). When we deal with multiple automatons, we use a subscript $_A$ to indicate information associated with automaton A, e.g., θ_A is the start state of automaton A.

Given a string Q and a path P in A we say that Q and P *match* if the concatenation of the labels on the transitions in P is Q. Given a state s in A we define the *state-set transition* $\delta_A(s, Q)$ to be the set of states reachable from s through paths matching Q. For a set of states S we define $\delta_A(S, Q) = \bigcup_{s \in S} \delta_A(s, Q)$. We say that A *accepts* the string Q if $\phi_A \in \delta_A(\theta_A, Q)$. Otherwise A *rejects* q. For an accepting path P in A, we define the *parse* of P for A to be the sequence of character transitions in A on P. Given a string Q accepted by A, a parse of Q is a parse for A of any accepting path matching Q.

We can use a sequence of state-set transitions to test acceptance of a string Q of length n by computing a sequence of state-sets $S_0, \dots, S_n$, given by $S_0 = \delta_A(\theta_A, \epsilon)$ and $S_i = \delta_A(S_{i-1}, Q[i])$, $i = 1, \dots, n$. We have that $\phi_A \in S_n$ iff A accepts Q. We can extend the algorithm to also compute the parse of Q for A by processing the state-sets in reverse order to recover an accepting path and output the character transitions. Note that for matching we only need to store the latest two state-sets at any point to compute the final state-set S_n, whereas for parsing we store the full sequence of state-sets.

Thompson NFA. Given a regular expression R, we can construct an NFA accepting precisely the strings in $L(R)$ by several classic methods [20, 12, 26]. In particular, Thompson [26] gave the simple well-known construction shown in Figure 1. We will call an NFA constructed

[2] Sometimes NFAs are allowed a *set* of accepting states, but this is not necessary for our purposes.

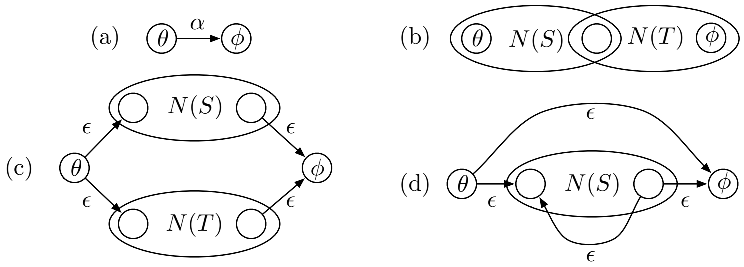


Figure 1 Thompson's recursive NFA construction. The regular expression $\alpha \in \Sigma \cup \{\epsilon\}$ corresponds to NFA (a). If S and T are regular expressions then $N(ST)$, $N(S|T)$, and $N(S^*)$ correspond to NFAs (b), (c), and (d), respectively. In each of these figures, the leftmost state θ and rightmost state ϕ are the start and the accept nodes, respectively. For the top recursive calls, these are the start and accept states of the overall automaton. In the recursions indicated, e.g., for $N(ST)$ in (b), we take the start state of the subautomaton $N(S)$ and identify with the state immediately to the left of $N(S)$ in (b). Similarly the accept state of $N(S)$ is identified with the state immediately to the right of $N(S)$ in (b).

with these rules a *Thompson NFA* (TNFA). A TNFA $N(R)$ for R has at most $2m$ states, at most $4m$ transitions, and can be computed in $O(m)$ time. Note that each character in R corresponds to a unique character transition in $N(R)$ and hence a parse of a string Q for $N(R)$ directly corresponds to a parse of Q for R. The parse tree of a TNFA $N(R)$ is the parse tree of R. With a breadth-first search of A we can compute a state-set transition for a single character in $O(m)$ time. By our above discussion, it follows that we can solve regular expression matching in $O(nm)$ time and $O(m)$ space, and regular expression parsing in $O(nm)$ time and $O(nm)$ space.

TNFA Decomposition. We need the following decomposition result for TFNAs (see Figure 2). Similar decompositions are used in [22, 3]. Given a TNFA A with $m > 2$ states, we decompose A into an *inner subTNFA* A_I and an *outer subTNFA* A_O. The inner subTNFA consists of a pair of *boundary states* θ_{A_I} and ϕ_{A_I} and all states and transitions that are reachable from θ_{A_I} without going through ϕ_{A_I}. Furthermore, if there is a path of ϵ-transitions from ϕ_{A_I} to θ_{A_I} in A_O, we add an ϵ-transition from ϕ_{A_I} to θ_{A_I} in A_I (following the rules from Thompson's construction). The outer subTNFA is obtained by removing all states and transitions of A_I except θ_{A_I} and ϕ_{A_I}. Between θ_{A_I} and ϕ_{A_I} we add a special transition labeled $\beta_{A_I} \notin \Sigma$ and if A_I accepts the empty string we also add an ϵ-transition (corresponding to the regular expression $(\beta_{A_I} \mid \epsilon)$). The decomposition has the following properties. Similar results are proved in [22, 3] (see also full version [5] for a the proof).

▶ **Lemma 2.** *Let A be any TNFA with $m > 2$ states. In $O(m)$ time we can decompose A into inner and outer subTNFAs A_O and A_I such that*

(i) *A_O and A_I have at most $\frac{2}{3}m + 8$ states each, and*

(ii) *any path from A_O to A_I crosses θ_{A_I} and any path from A_I to A_O crosses ϕ_{A_I}.*

3 String Decompositions

Let A be a TNFA decomposed into subTNFAs A_O and A_I and Q be a string accepted by A. We show how to efficiently decompose Q into substrings corresponding to subpaths matched in each subTNFA. The algorithm will be a key component in our recursive algorithm in the next section.

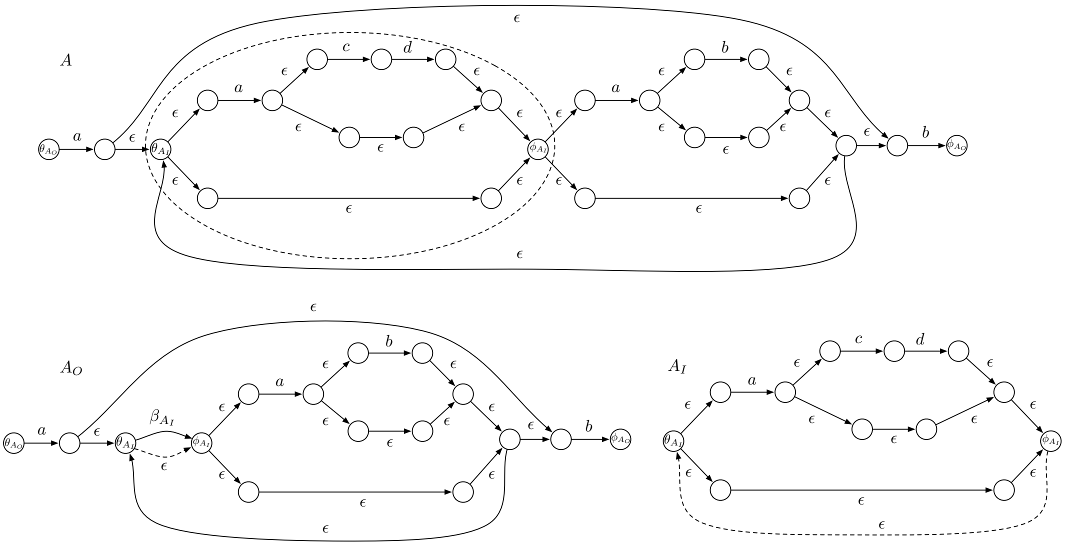


Figure 2 Decomposition of TNFA A into subTNFAs A_O and A_I. The dotted ϵ-transition in A_O exists since A_I accepts the empty string, and the dotted ϵ-transition in A_I exists since there is a path of ϵ-transitions from ϕ_{A_I} to θ_{A_I}.

Given an accepting path P in A, we define the *path decomposition* of P wrt. A_I to be the partition of P into a sequence of subpaths $\mathcal{P} = \overline{p}_1, p_1, \overline{p}_2, p_2, \ldots, \overline{p}_\ell, p_\ell, \overline{p}_{\ell+1}$, where the *outer subpaths*, $\overline{p}_1, \ldots, \overline{p}_{\ell+1}$, are subpaths in A_O and the *inner subpaths*, $p_1, \ldots, p_\ell$ are the subpaths in A_I. The *string decomposition* induced by $\mathcal{P}$ is the sequence of substrings $\mathcal{Q} = \overline{q}_1, q_1, \overline{q}_2, q_2, \ldots, \overline{q}_\ell, q_\ell, \overline{q}_{\ell+1}$ formed by concatenating the labels of the corresponding subpath in A. A sequence of substrings is a substring decomposition wrt. to A_I if there exists an accepting path that induces it. Our goal is to compute a string decomposition in $O(nm)$ time and $O(n + m)$ space, where n is the length of Q and m is the number of states in A.

An immediate idea would be to process Q from left to right using state-set transitions and "collapse" the state set to a boundary state b of A_I whenever the state set contains b and there is a path from b to ϕ_A matching the rest of Q. Since A_O and A_I only interact at the boundary states, this effectively corresponds to alternating the simulation of A between A_O and A_I. However, because of potential cyclic dependencies from paths of ϵ-transition from ϕ_{A_I} to θ_{A_I} in A_O and θ_{A_I} to ϕ_{A_I} in A_I we cannot immediately determine which subTNFA we should proceed in and hence we cannot correctly compute the string decomposition. For instance, consider the string $Q = aaacdaabaacdacdaabab$ from Figure 3. After processing the first two characters (aa) both θ_{A_I} and ϕ_{A_I} are in the state set, and there is a path from both these states to ϕ_A matching the rest of Q. The same is true after processing the first six characters ($aaacda$). In the first case the substring consisting of the next three characters (acd) only matches a path in A_I, whereas in the second case the substring consisting of the next two characters (ab) only matches a path in A_O. A technical contribution in our algorithm in the next section is to efficiently overcome these issues by a two-step approach that first decomposes the string into substrings and labels the substrings greedily to find a correct string decomposition.

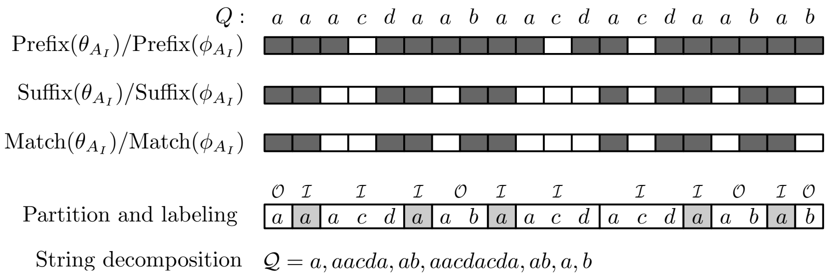

Q : a a a c d a a b a a c d a c d a a b a b
Prefix(θ_{A_I})/Prefix(ϕ_{A_I})
Suffix(θ_{A_I})/Suffix(ϕ_{A_I})
Match(θ_{A_I})/Match(ϕ_{A_I})
Partition and labeling
$\mathcal{O}$ $\mathcal{I}$ $\mathcal{I}$ $\mathcal{I}$ $\mathcal{O}$ $\mathcal{I}$ $\mathcal{I}$ $\mathcal{I}$ $\mathcal{I}$ $\mathcal{O}$ $\mathcal{I}$ $\mathcal{O}$
a a a c d a a b a a c d a c d a a b a b
String decomposition
$Q = a, aacda, ab, aacdacda, ab, a, b$

Figure 3 The string decomposition of the string $Q = aaacdaabaacdacdaabab$ wrt. A_I in Figure 2 and the corresponding suffix/prefix match sets. The dark grey blocks in the prefix, suffix and match sets are the positions contained in the sets. The blocks in the partition of the string are labeled $\mathcal{O}$ and $\mathcal{I}$ for outer and inner, respectively. The grey blocks in the partition are the substrings that can be parsed by both the inner and outer automaton. According to our procedure these blocks are labeled *inner*.

3.1 Computing String Decompositions

We need the following new definitions. Let i be a position in Q and let s be a state in A. We say that (i, s) is a *valid pair* if there is a path from θ_A to s matching $Q[1, i]$ and from s to ϕ_A matching $Q[i + 1, n]$. For any set of states X in A, we say that (i, X) is a valid pair if each pair (i, x), $x \in X$, is a valid pair. An accepting path P in A *intersects* a valid pair (i, X) if some state $x \in X$ is on the path, the subpath of P from θ_A to x matches $Q[1, i]$, and the subpath of P from x to ϕ_A matches $Q[i + 1, n]$.

Our algorithm consist of the following steps. In step 1, we process Q from left to right and right to left to compute and store the *match sets*, consisting of all valid pairs for the boundary states θ_{A_I} and ϕ_{A_I}. We then use the match sets in step 2 to process Q from left to right to build a sequence of valid pairs for the boundary states that all intersect a single accepting path P in A matching Q, and that has the property that all positions where the accepting path P contains θ_{A_I} or ϕ_{A_I} correspond to a valid pair in the sequence. Finally, in step 3 we construct the string decomposition using a greedy labeling of the sequence of valid pairs. See Figure 3 for an example of the computation in each step.

Step 1: Computing Match Sets

First, compute the *match sets* given by

$$\text{Match}(\theta_{A_I}) = \{i \mid (i, \theta_{A_I}) \text{ is a valid pair}\}$$
$$\text{Match}(\phi_{A_I}) = \{i \mid (i, \phi_{A_I}) \text{ is a valid pair}\}$$

Thus, Match(θ_{A_I}) and Match(ϕ_{A_I}) are the positions in Q that correspond to a valid pair for the boundary states θ_{A_I} and ϕ_{A_I}, respectively. To compute these, we first compute the *prefix match sets*, Prefix(s), and *suffix match sets*, Suffix(s), for $s \in \{\theta_{A_I}, \phi_{A_I}\}$. A position i is in Prefix(s) if there is a path from θ_A to s accepting the prefix $Q[1, i]$, and in Suffix(s) if there is a path from s to ϕ_A accepting the suffix $Q[i + 1, n]$. To compute the prefix match sets we perform state-set transitions on Q and A and whenever the current state-set contains either θ_{A_I} or ϕ_{A_I} we add the corresponding position to Prefix(s). We compute the suffix match sets in the same way, but now we perform the state-set transitions on Q from right to left and A with the direction of all transitions reversed. Each step of the state-set transition takes $O(m)$ time and hence we use $O(nm)$ time in total.

Finally, we compute the match sets Match(s), for $s \in \{\theta_{A_I}, \phi_{A_I}\}$, by taking the intersection of Prefix(s) and Suffix(s). In total we use $O(mn)$ time and $O(n+m)$ space to compute and store the match sets.

Step 2: Computing Valid Pairs

We now compute a sequence of valid pairs

$$V = (i_1, X_1), (i_2, X_2), \ldots, (i_k, X_k)$$

such that $0 \leq i_1 < \cdots < i_k \leq n$ and $X_j \subseteq \{\theta_{A_I}, \phi_{A_I}\}$ and with the property that the states of all pairs intersect a single accepting path P in A and at all places where P is equal to either θ_{A_I} or ϕ_{A_I} correspond to a valid pair in V.

To compute the sequence we run a slightly modified state-set transition algorithm: For $i = 0, 1, \ldots, n$ we set $S_i = \delta_A(S_{i-1}, Q[i])$ (for $i = 0$ set $S_0 := \delta_A(\theta_A, \epsilon)$) and compute the set

$$X := \{x \mid x \in \{\theta_{A_I}, \phi_{A_I}\} \text{ and } i \in \text{Match}(x)\} \cap S_i \ .$$

Thus X is the set of boundary states in S_i that corresponds to a valid pair computed in Step 1. If $X \neq \emptyset$ we add (i, X) to the sequence V and set $S_i := X$.

We argue that this produces a sequence V of valid pairs with the required properties. First note that by definition of X we inductively produce state-set $S_0, \ldots, S_n$ such that S_i contains the set of states reachable from θ_A that match $Q[1, i]$ and the paths used to reach S_i intersect the states of the valid pairs produced up to this point. Furthermore, we include all positions in V where S_i contains θ_{A_I} or ϕ_{A_I}. It follows that V satisfies the properties.

Each of modified state-set transition uses $O(m)$ time and hence we use $O(nm)$ time in total. The sequence V uses $O(n)$ space. In addition to this we store the match sets and a constant number of active state-sets using $O(n+m)$ space.

Step 3: Computing the String Decomposition

We now convert the sequence $V = (i_1, X_1), (i_2, X_2), \ldots, (i_k, X_k)$ into a string decomposition. First, we construct the partition $q_0, \ldots, q_{k+1}$ of Q such that $q_0 = Q[1, i_1]$, $q_j = Q[i_j + 1, i_{j+1}]$, and $q_{k+1} = Q[i_k + 1, n]$. Note that q_0 and q_{k+1} may be the empty string. Next, we convert this partition into a string decomposition by first labeling each substring as inner or outer and then greedily merging these substrings to obtain an alternating sequence forming a string decomposition. We discuss these steps in detail below.

Labeling. We label the substrings as follows. First label q_0 and q_{k+1} with outer. For the rest of the substrings, if $X_i = \{\theta_{A_I}\}$ and $X_{i+1} = \{\phi_{A_I}\}$ then label q_i with inner, and if $X_i = \{\phi_{A_I}\}$ and $X_{i+1} = \{\theta_{A_I}\}$ then label q_i with outer. If either X_i or X_{i+1} contain more than one boundary state then we use state-set transitions in A_I and A_O to determine if A_I accepts q_i or if there is a path in A_O from ϕ_{A_I} to θ_{A_I} that matches q_i. If a substring is accepted by A_I then it can be an inner substring and if there is a path in A_O from ϕ_{A_I} to θ_{A_I} that matches q_i then it can be an outer substring. If a substring can only be either an inner or an outer substring then it is labeled with inner or outer, respectively. Let q_i be a substring that can be both an inner or an outer substring. We divide this into two cases. If there is an ϵ-path from ϕ_{A_I} to θ_{A_I} then label all such q_i with inner. Otherwise label all such q_i with outer. See also Algorithm 1.

Algorithm 1 Labeling.

```
Input: A sequence $V$ of valid pairs $(i_1, X_1), \dots (i_k, V_k)$ and the corresponding
       partition $q_0, \dots, q_{k+1}$ of $Q$.
Output: A labeling of the partition
The (possible empty) substrings $q_0$ and $q_{k+1}$ are labeled outer.
for $i = 1$ to $k$ do
    if $X_i = \{\theta_{A_I}\}$ and $X_{i+1} = \{\phi_{A_I}\}$ then                        /* Case 1 */
        label $q_i$ inner.
    else if $X_i = \{\phi_{A_I}\}$ and $X_{i+1} = \{\theta_{A_I}\}$ then                   /* Case 2 */
        label $q_i$ outer.
    else if $X_i$ or $X_{i+1}$ contains more than one boundary node then /* Case 3 */
        Use standard state-set transitions in $A_I$ and $A_O$ to determine if $A_I$ accepts $q_i$
         or if there is a path in $A_O$ from $\phi_{A_I}$ to $\theta_{A_I}$ that matches $q_i$.
        if $q_i$ is only accepted by $A_I$ then                              /* Subcase 3a */
            label $q_i$ inner
        else if $q_i$ is only accepted by $A_O$ then                         /* Subcase 3b */
            label $q_i$ with outer
        else /* $q_i$ is accepted by both $A_I$ and $A_O$                                 */
            There are two cases.
            if there is an $\epsilon$-path from $\phi_{A_I}$ to $\theta_{A_I}$ then               /* Subcase 3c */
                label $q_i$ inner.
            else label $q_i$ outer.                                       /* Subcase 3d */
        end
    end
end
```

For correctness first note that q_0 and q_{k+1} are always (possibly empty) outer substrings. The cases where both $|X_i| = |X_{i+1}|$ (case 1 and 2) are correct by the correctness of the sequence of valid pairs V. Due to cyclic dependencies we may have that X_i and X_{i+1} contain more than one boundary state. This can happen if there is an ϵ-path from θ_{A_I} to ϕ_{A_I} and/or there is an ϵ-path from ϕ_{A_I} to θ_{A_I}. If a substring only is accepted by one of A_I (case 3a) or A_O (case 3b) then it follows from the correctness of V that the labeling is correct. It remains to argue that the labeling in the case where q_i is accepted by both A_I and A_O is correct. To see why the labeling in this case is consistent with a string decomposition of the accepting path consider case 3c. Here, it is safe to label q_i with inner, since if we are in ϕ_{A_I} after having read q_{i-1} we can just follow the ϵ-path from ϕ_{A_I} to θ_{A_I} and then start reading q_i from here.The argument for case 3d is similar.

Except for the state-set transitions in case 3 all cases takes constant time. The total time of all the state-set transitions is $O(nm)$. The space of V and the partition together with the labeling uses $O(n)$ space.

String decomposition. Now every substring has a label that is either inner or outer. We then merge adjacent substrings that have the same label. This produces an alternating sequence of inner and outer substrings, which is the final string decomposition. Such an alternating subsequence must always exist since each pair in V intersects an accepting path.

In summary, we have the following result.

▶ **Lemma 3.** *Given string Q of length n, and TNFA A with m states decomposed into A_O and A_I, we can compute a string decomposition wrt. A_I in $O(nm)$ time and $O(n+m)$ space.*

4 Computing Accepting Paths

Let Q be a string of length n accepted by a TNFA A with m states. In this section we show how to compute an accepting path for Q in A in $O(nm)$ time and $O(n+m)$ space. Since an accepting path may have length $\Omega(nm)$ (there may be $\Omega(m)$ ϵ-transitions between each character transition) we cannot afford to explicitly compute the path in our later $o(nm)$ time algorithms. Instead, our algorithm will compute *compressed paths* of size $O(n)$ obtained by deleting all ϵ-transitions from a path. Note that the compressed path stores precisely the information needed to solve the parsing problem.

To compute the compressed path we define a recursive algorithm $\text{Path}(A, Q)$ that takes a TNFA A and a string Q and returns a compressed accepting path in A matching Q as follows. If $n < \gamma_n$ or $m < \gamma_m$, for constants $\gamma_n, \gamma_m > 0$ that we will fix later, we simply run the naive algorithm that stores all state-sets during state-set simulation from left-to-right in Q. Since one of n or m is constant this uses $O(nm) = O(n+m)$ time and space. Otherwise, we proceed according to the following steps.

Step 1: Decompose. We compute a decomposition of A into inner and outer subTNFAs A_I and A_O and compute a corresponding string decomposition $\mathcal{Q} = \overline{q}_1, q_1, \overline{q}_2, q_2, \overline{q}_\ell, q_\ell, \overline{q}_{\ell+1}$ for Q.

Step 2: Recurse. We build a single substring corresponding to all the subpaths in A_O and ℓ substrings for A_I (one for each subpath in A_I) and recursively compute the compressed paths. To do so, construct $\overline{q} = \overline{q}_1 \cdot \beta_{A_I} \cdot \overline{q}_2 \cdot \beta_{A_I} \cdots \beta_{A_I} \cdot \overline{q}_{\ell+1}$. Recall, that β_{A_I} is the label of the special transition we added between θ_{A_I} and ϕ_{A_I}in A_O. Then, recursively compute the compressed paths

$$\begin{aligned} \overline{p} &= \text{Path}(A_O, \overline{q}) \\ p_i &= \text{Path}(A_I, q_i) \qquad 1 \leq i \leq \ell \end{aligned}$$

Step 3: Combine. Finally, extract the subpaths $\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_{\ell+1}$ from $\overline{p}$ corresponding to the substrings $\overline{q}_1, \overline{q}_2, \ldots, \overline{q}_{\ell+1}$ and return the combined compressed path

$$P = \overline{p}_0 \cdot p_1 \cdot \overline{p}_1 \cdot p_2 \cdot \overline{p}_2 \cdots p_\ell \cdot \overline{p}_{\ell+1}$$

Inductively, it directly follows that the returned compressed path is a compressed accepting path for Q in A.

4.1 Analysis

We now show that the total time $T(n,m)$ of the algorithm is $O(nm)$. If $n < \gamma_n$ or $m < \gamma_m$, we run the backtracking algorithm using $O(nm) = O(n+m)$ time and space. If $n \geq \gamma_n$ and $m \geq \gamma_m$, we implement the recursive step of the algorithm using $O(nm)$ time. Let n_i be the length of the inner string q_i in the string decomposition and let $n_0 = \sum_{i=1}^{\ell+1} |\overline{q}_i|$. Thus, $n = \sum_{i=1}^{\ell+1} n_i$ and $|\overline{q}| = n_0 + \ell$. In step 2, the recursive call to compute $\overline{p}$ takes $O(T(n_0{+}\ell, \frac{2}{3}m{+}8))$ time and the recursive calls to compute $p_1, \ldots, p_\ell$ take $\sum_{i=1}^{\ell} T(n_i, \frac{2}{3}m{+}8)$

time. The remaining steps of the algorithm all take $O(nm)$ time. Hence, we have the following recurrence for $T(n, m)$.

$$T(n,m) = \begin{cases} \sum_{i=1}^{\ell} T(n_i, \frac{2}{3}m + 8) + T(n_0 + \ell, \frac{2}{3}m + 8) + O(mn) & m \geq \gamma_m \text{ and } n \geq \gamma_n \\ O(m+n) & m < \gamma_m \text{ or } n < \gamma_n \end{cases}$$

It follows that $T(n, m) = O(nm)$ for $\gamma_n = 2$ and $\gamma_m = 25$ (see full version [5] for a detailed proof).

Next, we consider the space complexity. First, note that the total space for storing R and Q is $O(n + m)$. To analyse the space during the recursive calls of the algorithm, consider the recursion tree $\mathcal{T}_{\text{rec}}$ for $\text{Path}(A, Q)$. For a node v in $\mathcal{T}_{\text{rec}}$, we define Q_v of length n_v to be the string and A_v with m_v states to be the TNFA at v. Consider a path $v_1, \ldots, v_j$ of nodes in $\mathcal{T}_{\text{rec}}$ from the root to leaf v_j corresponding to a sequence of nested recursive calls of the algorithm. If we naively store the subTNFAs, the string decompositions, and the compressed paths, we use $O(n_{v_i} + m_{v_i})$ space at each node v_i, $1 \leq i \leq j$. By Lemma 2(i) the sum of the sizes of the subTNFAs on a path forms a geometrically decreasing sequence and hence the total space for the subTNFAs is $\sum_{i=1}^{j} m_{v_i} = O(m)$. However, since each string (and hence compressed path) at each node v_i, $1 \leq i \leq j$, may have length $\Omega(n)$ we may need $\Omega(n \log m)$ space in total for these. We show how to improve this to $O(n + m)$ space in the next section.

4.2 Squeezing into Linear Space

We now show how to improve the space to $O(n + m)$. To do so we show how to carefully implement the recursion to only store the strings for a selected subset of the nodes along any root to leaf path in $\mathcal{T}_{\text{rec}}$ that in total take no more than $O(n)$ space.

First, consider a node v in $\mathcal{T}_{\text{rec}}$ and the corresponding string Q_v and TNFA A_v. Define χ_v^Q to be the function that maps each character position in Q_v (ignoring β_{A_I} transitions) to the unique corresponding character in Q and χ_v^A to be the function that maps each character transition (non-ϵ transition) in A_v to the unique character transition in A. Note that these are well-defined by the construction of subproblems in the algorithm. At a node v, we represent χ_v^Q by storing for each character in Q_v a pointer to the corresponding character in Q. Similarly, we represent χ_v^A by storing for each character transition in A_v a pointer to the corresponding character transition in A. This uses $O(n_v + m_v)$ additional space. It is straightforward to compute these mappings during the algorithm directly from the string and TNFA decomposition in the same time. With the mappings we can now output transitions on the compressed path as pairs of position in Q and transitions in A immediately as they are computed in a leaf of $\mathcal{T}_{\text{rec}}$. Thus, when we have traversed a full subtree at a node we can free the space for that node since we do not have to wait to return to the root to combine the pieces of the subpath with other subpaths.

We combine the mappings with an ordering of the recursion according to a *heavy-path decomposition* of $\mathcal{T}_{\text{rec}}$. Let v be an internal node in $\mathcal{T}_{\text{rec}}$. The *string length* of v is n_v. We define the *heavy child* of v to be a child of v of maximum string length among the children of v. The remaining children are *light children* of v. We have the following key property of light children (see full version [5] for a proof).

▶ **Lemma 4.** *For any node v with light child u in $\mathcal{T}_{rec}$, we have that $n_u \leq \frac{3}{4}n_v + O(1)$.*

We order the recursive calls at each node v as follows. First, we recursively visit all the light children of v and upon completing each recursive call, we free the space for that node. Note that the mappings allow us to do this. We then construct the subproblem for the heavy child of v, free the space for v, and continue the recursion at the heavy child.

To analyse the space of the modified algorithm, consider a path a path $v_1, \ldots, v_j$ of nodes in $\mathcal{T}_{\text{rec}}$ from the root to a leaf v_j. We now have that only nodes v_i, $1 \leq i < j$ will explicitly store a string if v_{i+1} is a light child of v_i. By Lemma 4 the sum of these lengths form a geometrically decreasing sequence and hence the total space is now $O(n)$. In summary, we have shown the following result.

▶ **Theorem 5.** *Given a TNFA with m states and a string of length n, we can compute a compressed accepting path for Q in A in $O(nm)$ time and $O(n + m)$ space.*

Note that the algorithm works in a comparison-based, pointer model of computation. By our discussion this immediately implies the main result of Theorem 1.

5 Speeding up the Algorithm

We now show how to adapt the algorithm to use the faster state-set simulation algorithms such as Myers' algorithm [22] and later variants [4, 3] that improve the $O(m)$ bound for a single state-set transition. These results and ours all work on a unit-cost word RAM model of computation with w-bit words and a standard instruction set including addition, bitwise boolean operations, shifts, and multiplication. We can store a pointer to any position in the input and hence $w \geq \log(n + m)$. For simplicity, we will show how to adapt the tabulation-based algorithm of Bille and Farach-Colton [4].

5.1 Fast Matching

Let A be a TNFA with m states and let Q be a string of length n. Assume first that the alphabet is constant. We briefly describe the main features of the algorithm by Bille and Farach-Colton [4] that solves the matching problem in $O(nm/\log n + n + m)$ time and $O(n^{\varepsilon} + m)$ space, for any constant $\varepsilon > 0$. In the next section we show how to adapt the algorithm to compute an accepting path.

Given a parameter $t < w$, we will construct a global table of size $2^{ct} < 2^{w}$, for a constant c, to speed up the state-set transition. We decompose A into a tree MS of $O(\lceil m/t \rceil)$ micro TNFAs, each with at most t states. For each $M \in MS$, each child C is represented by its start and accepting state and a pseudo-transition connecting these. By similar arguments as in Lemma 2 we can always construct such a decomposition.

We represent each micro TNFA $M \in MS$ uniquely by its parse tree using $O(t)$ bits. Since M has at most t states, we can represent the state-set for M, called the *local state-set* and denoted S_M, using t bits. Hence, we can make a universal table of size $2^{O(t)}$ that for every possible micro TNFA M of size $\leq t$, local state-set S_M, and character $\alpha \in \Sigma \cup \{\epsilon\}$ computes the state-set transition $\delta_M(S_M, \alpha)$ in constant time.

We use the tabulation to efficiently implement a global state-set transition on A as follows. We represent the state-set for A as the union of the local state-sets in MS. Note that parents and children in MS share some local states, and these states will be copied in the local state-sets.

To implement a state-set transition on A for a character α, we first traverse all transitions labeled α in each micro TNFA from the current state-set. We then follow paths of ϵ transition in two depth-first left-to-right traversal of MS. At each micro TNFA M, we compute all states reachable via ϵ-transitions and propagate the shared states among parents and children in MS. Since any cycle free path in a TNFA contains at most one *back transition* (see [22, Lemma 1]) it follows that two such traversals suffices to to correctly compute all states in A reachable via ϵ-transitions.

With the universal table, we process each micro TNFA in constant time, leading to an algorithm using $O(|MS|/t + n + m) = O(nm/t + n + m)$ time and $O(2^t + m)$ space. Setting $t = \varepsilon \log n$ produces the stated result. Note that each state-set uses $O(\lceil m/t \rceil)$ space. To handle general alphabets, we store dictionaries for each micro TNFA with bit masks corresponding to characters appearing in the TNFA and combine these with an additional masking step in state-set transition. The leads a general solution with the same time and space bounds as above.[3]

5.2 Fast Parsing

We now show how to modify our algorithm from Section 4 to take advantage of the fast state-set transition algorithm. Let $t < w$ be the desired speed up as above. We have the following two cases.

If $n \geq t$ and $m \geq t$ we implement the recursive step of the algorithm but replace all state-set transitions, that is when we compute the match sets and valid pairs, by the fast state-set transition algorithm. To compute the suffix match sets we need to compute fast state-set transitions on A with the direction of all transitions reversed. To do so, we make a new universal table of size $2^{O(t)}$ for the micro TNFAs with the direction of transitions reversed. We traverse the tree of micro TNFAs with two depth traversals as before except that we now traverse children in right to left order to correctly compute all reachable states. It follows that this uses $O(nm/t)$ time.

Otherwise ($n < t$ or $m < t$), we use backtracking to compute the accepting path as follows. First, we process Q from left-to-right using fast state-set transitions to compute the sets $S_0, \ldots, S_n$ of states reachable via paths from θ_A for each prefix of Q. We store each of these state-sets. This uses $O(nm/t + n + m) = O(n + m)$ time and space. Then, we process Q from right-to-left to recover a compressed accepting path in A. Starting from ϕ_A we repeatedly do a fast state-set transition A with the direction of transition reversed, compute the intersection of the resulting state-set with the corresponding state-set from the first step, and update the state-set to a state in the intersection. We can compute the intersection of state-sets and the update in $O(m/t)$ time using standard bit wise operations. We do the state-set transitions on the TNFA with directions of transitions reversed as above. In total, this uses $O(nm/t + n + m) = O(n + m)$ time.

In summary, we have the following recurrence for the time $T(n, m)$.

$$T(n,m) = \begin{cases} \sum_{i=0}^{\ell} T(n_i, \frac{2}{3}m + 8) + T(n_0 + \ell, \frac{2}{3}m + 8) + O\left(\frac{nm}{t}\right) & n \geq t \text{ and } m \geq t \\ O\left(n + m\right) & m < t \text{ or } n < t \end{cases}$$

Similar to Section 4.1 it follows that $T(n, m) = O(nm/t + n + m)$, for $25 \leq t < w$. The space is linear as before. Plugging in $t = \epsilon \log n$ and including the preprocessing time and space for the universal tables we obtain the following logarithmic improvement of Theorem 1.

► **Theorem 6.** *Given a regular expression of length m, a string of length n, we can solve the regular expression parsing problem in $O(nm/\log n + n + m)$ time and $O(n + m)$ space.*

[3] Note that the time bound in the original paper has an additional $m \log m$ term [4]. Using atomic heaps [9] to represent dictionaries for micro TNFAs this term is straightforward to improve to $O(m)$. See also Bille and Thorup [6, Appendix A].

References

1 Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Inc., 1986.

2 Arturs Backurs and Piotr Indyk. Which Regular Expression Patterns are Hard to Match? In *Proc. 57th FOCS*, pages 457–466, 2016.

3 Philip Bille. New Algorithms for Regular Expression Matching. In *Proc. of the 33rd ICALP*, pages 643–654, 2006.

4 Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theor. Comput. Sci.*, 409(3):486–496, 2008.

5 Philip Bille and Inge Li Gørtz. From Regular Expression Matching to Parsing. *Arxiv preprint arXiv:1804.02906*, 2019.

6 Philip Bille and Mikkel Thorup. Faster Regular Expression Matching. In *Proc. 36th ICALP*, pages 171–182, 2009. Full version with appendix available at `http://www2.compute.dtu.dk/~phbi/files/publications/2009fremC.pdf`.

7 Philip Bille and Mikkel Thorup. Regular Expression Matching with Multi-Strings and Intervals. In *Proc. 21st SODA*, pages 1297–1308, 2010.

8 Danny Dubé and Marc Feeley. Efficiently building a parse tree from a regular expression. *Acta Informatica*, 37(2):121–144, 2000.

9 Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. System Sci.*, 48(3):533–551, 1994.

10 Alain Frisch and Luca Cardelli. Greedy regular expression matching. In *Proc. 31st ICALP*, volume 3142, pages 618–629, 2004.

11 Minos N Garofalakis, Rajeev Rastogi, and Kyuseok Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *Proc. 25th VLDB*, pages 223–234, 1999.

12 Victor M. Glushkov. The Abstract Theory of Automata. *Russian Math. Surveys*, 16(5):1–53, 1961.

13 D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, 1975.

14 Theodore Johnson, S. Muthukrishnan, and Irina Rozenbaum. Monitoring Regular Expressions on Out-of-Order Streams. In *Proc. 23nd ICDE*, pages 1315–1319, 2007.

15 Steven M Kearns. Extending regular expressions with context operators and parse extraction. *Software: Practice and Experience*, 21(8):787–804, 1991.

16 Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. Proton: multitouch gestures as regular expressions. In *Proc. SIGCHI*, pages 2885–2894, 2012.

17 Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In *Proc. SIGCOMM*, pages 339–350, 2006.

18 Ville Laurikari. NFAs with tagged transitions, their conversion to deterministic automata and application to regular expressions. In *Proc. 7th SPIRE*, pages 181–187, 2000.

19 Quanzhong Li and Bongki Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proc. 27th VLDB*, pages 361–370, 2001.

20 R. McNaughton and H. Yamada. Regular Expressions and State Graphs for Automata. *IRE Trans. on Electronic Computers*, 9(1):39–47, 1960.

21 Makoto Murata. Extended path expressions of XML. In *Proc. 20th PODS*, pages 126–137, 2001.

22 E. W. Myers. A Four-Russian Algorithm for Regular Expression Pattern Matching. *J. ACM*, 39(2):430–448, 1992.

23 Gonzalo Navarro and Mathieu Raffinot. Fast and Simple Character Classes and Bounded Gaps Pattern Matching, with Applications to Protein Searching. *J. Comp. Biology*, 10(6):903–923, 2003.

24 Lasse Nielsen and Fritz Henglein. Bit-coded Regular Expression Parsing. In *Proc. 5th LATA*, pages 402–413, 2011.

25 Martin Sulzmann and Kenny Zhuo Ming Lu. Regular expression sub-matching using partial derivatives. In *Proc. 14th PPDP*, pages 79–90, 2012.

26 K. Thompson. Regular Expression Search Algorithm. *Commun. ACM*, 11:419–422, 1968.

27 Fang Yu, Zhifeng Chen, Yanlei Diao, T. V. Lakshman, and Randy H. Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In *Proc. ANCS*, pages 93–102, 2006.

Solving Systems of Equations in Supernilpotent Algebras

Erhard Aichinger
Institute for Algebra, Johannes Kepler University Linz, Linz, Austria
https://www.jku.at/institut-fuer-algebra/
erhard@algebra.uni-linz.ac.at

Abstract

Recently, M. Kompatscher proved that for each finite supernilpotent algebra $\mathbf{A}$ in a congruence modular variety, there is a polynomial time algorithm to solve polynomial equations over this algebra. Let μ be the maximal arity of the fundamental operations of $\mathbf{A}$, and let

$$d := |A|^{\log_2 \mu + \log_2 |A| + 1}.$$

Applying a method that G. Károlyi and C. Szabó had used to solve equations over finite nilpotent rings, we show that for $\mathbf{A}$, there is $c \in \mathbb{N}$ such that a solution of every system of s equations in n variables can be found by testing at most cn^{sd} (instead of all $|A|^n$ possible) assignments to the variables. This also yields new information on some circuit satisfiability problems.

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

Keywords and phrases Supernilpotent algebras, polynomial equations, polynomial mappings, circuit satisfiability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.72

Related Version A preliminary version is available on https://arxiv.org/abs/1901.07862.

Funding Supported by the Austrian Science Fund FWF P29931: Clonoids: a unifying approach to equational logic and clones

Acknowledgements The author thanks G. Horváth and M. Kompatscher for dicussions on solving equations over nilpotent algebras. Several of these discussions took place during a workshop organized by P. Aglianò at the University of Siena in June 2018. The author also thanks A. Földvári, C. Szabó, M. Kompatscher, and S. Kreinecker for their comments on preliminary versions of the manuscript, and the anonymous referees for several useful suggestions.

1 Introduction

We study systems of polynomial equations over a finite algebraic structure $\mathbf{A}$. Such a system is given by equations of the form $p(x_1, \dots, x_n) \approx q(x_1, \dots, x_n)$, where p, q are polynomial terms of $\mathbf{A}$; a polynomial term of $\mathbf{A}$ is a term of the algebra $\mathbf{A}^*$ which is obtained by expanding $\mathbf{A}$ with one nullary function symbol for each $a \in A$. A *solution* to a system $p_i(x_1, \dots, x_n) \approx q_i(x_1, \dots, x_n)$ $(i = 1, \dots, s)$ is an element $\boldsymbol{a} = (a_1, \dots, a_n) \in A^n$ such that $p_i^{\mathbf{A}}(\boldsymbol{a}) = q_i^{\mathbf{A}}(\boldsymbol{a})$ for all $i \in \{1, \dots, s\}$. The problem to decide whether such a solution exists has been called PolSysSat($\mathbf{A}$), and PolSat($\mathbf{A}$) if the system consists of one single equation, and the terms of the input are encoded as strings over $\{x_1, \dots, x_n\} \cup A \cup F$, where F is the set of function symbols of $\mathbf{A}$. A survey of results on the computational complexity of this problem is given, e.g., in [13, 17]. In algebras such as groups, rings or Boolean algebras, one can reduce an equation $p(\boldsymbol{x}) \approx q(\boldsymbol{x})$ to an equation of the form $f(\boldsymbol{x}) \approx y$, where $y \in A$. A system of equations of this form then has the form $f_i(\boldsymbol{x}) \approx y_i$ $(i = 1, \dots, s)$. For $n \in \mathbb{N}$, let $\mathrm{Pol}_n(\mathbf{A})$ denote the n-ary polynomial functions on $\mathbf{A}$ [19, Definition 4.4]. For a finite

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 72; pp. 72:1–72:9
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

nilpotent ring or group $\mathbf{A}$, [10, 12] establish the existence of a natural number $d_{\mathbf{A}}$ such that for every $f \in \mathrm{Pol}_n(\mathbf{A})$ and for every $\boldsymbol{a} \in A^n$, there exists $\boldsymbol{b}$ such that $f^{\mathbf{A}}(\boldsymbol{a}) = f^{\mathbf{A}}(\boldsymbol{b})$ and $\boldsymbol{b}$ has at most $d_{\mathbf{A}}$ components that are different from 0. Hence the equation $f(\boldsymbol{x}) \approx y$ has a solution if and only if it has a solution with at most $d_{\mathbf{A}}$ nonzero entries. Thus for the algebra $\mathbf{A}$, testing only vectors with at most $d_{\mathbf{A}}$ nonzero entries is an algorithm, which, given an equation $f(\boldsymbol{x}) \approx y$ of length n, takes at most $c(\boldsymbol{A}) \cdot n^{d_{\mathbf{A}}+1}$ many steps to find whether this equation is solvable: there are at most $\sum_{i=0}^{d_{\mathbf{A}}} \binom{n}{i}(|A|-1)^i \leq c_1(\mathbf{A}) \cdot n^{d_{\mathbf{A}}}$ many evaluations to be done, each of them taking at most $c_2(\mathbf{A}) \cdot n$ many steps. The number $d_{\mathbf{A}}$ in [12] is obtained from Ramsey's Theorem and therefore rather large. In [17], it is proved that for every finite supernilpotent algebra in a congruence modular variety, such a number $d_{\mathbf{A}}$ exists, again using Ramsey's Theorem. For rings, lower values of $d_{\mathbf{A}}$ have been obtained in [15] (cf. [14]). In [7, 8], A. Földvári provides polynomial time algorithms for solving equations over finite nilpotent groups and rings relying on the structure theory of these algebras. In this paper, we extend the method developed in [15] from finite nilpotent rings to arbitrary finite supernilpotent algebras in congruence modular varieties. For such algebras, we compute $d_{\mathbf{A}}$ as $|A|^{\log_2 \mu + \log_2 |A| + 1}$, where μ is the maximal arity of the fundamental operations of $\mathbf{A}$ (Theorem 10). The technique that allows to generalize Károlyi's and Szabó's method is the coordinatization of nilpotent algebras of prime power order by elementary abelian groups from [1, Theorem 4.2]. The method can be generalized to systems of equations: we show for a given finite supernilpotent algebra $\mathbf{A}$ in a congruence modular variety, and a given $s \in \mathbb{N}_0$, there is a polynomial time algorithm to test whether a system of at most s polynomial equations over $\mathbf{A}$ has a solution. If s is not fixed in advance, then [18, Corollary 3.13] implies that if $\mathbf{A}$ is not abelian, PolSysSat($\mathbf{A}$) is NP-complete.

Let us finally explain to which class of algebras our results applies: A finite algebra $\mathbf{A}$ from a congruence modular variety with finitely many fundamental operations is supernilpotent if and only if it is a direct product of nilpotent algebras of prime power order; modulo notational differences explained, e.g., in [1, Lemma 2.4], this result has been proved in [16, Theorem 3.14]. Such an algebra is therefore always nilpotent, has a Mal'cev term (cf. [9, Theorem 6.2], [16, Theorem 2.7]), and hence generates a congruence permutable variety. For a more detailed introduction to supernilpotency and, for $k \in \mathbb{N}$, to k-supernilpotency, we refer to [2, 3, 1].

2 A theorem of Károlyi and Szabó

In this section, we state a special case of [15, Theorem 3.1]. Since their result is much more general than needed for our purpose, we also include a self-contained proof, which is a reduction Károlyi's and Szabó's proof to the case of elementary abelian groups.

For $n \in \mathbb{N} = \{1, 2, 3, \ldots\}$, we denote the set $\{1, 2, \ldots, n\}$ by $\underline{n}$. Let A be a set with an element $0 \in A$, and let $J \subseteq \underline{n}$. For $\boldsymbol{a} \in A^n$, $\boldsymbol{a}^{(J)}$ is defined by $\boldsymbol{a}^{(J)} \in A^n$, $\boldsymbol{a}^{(J)}(j) = \boldsymbol{a}(j)$ for $j \in J$ and $\boldsymbol{a}^{(J)}(j) = 0$ for $j \in \underline{n} \setminus J$. Suppose that 1 is an element of A. Then by $\boldsymbol{1}$, we denote the vector $(1, 1, \ldots, 1)$ in A^n, and for $J \subseteq \underline{n}$, $\boldsymbol{1}^{(J)}$ is the vector $(v_1, \ldots, v_n)$ with $v_j = 1$ if $j \in J$ and $v_j = 0$ if $j \notin J$. For any sets C, D, we write $C \subset D$ for ($C \subseteq D$ and $C \neq D$).

We first need the following variation of [5, Theorem 1] and [15, Theorem 3.2], which is proved using several arguments from the proof of [4, Theorem 3.1] and from [5].

▶ **Lemma 1.** *Let F be a finite field, let $k, m, n \in \mathbb{N}$, let $q := |F|$, let $p_1, \ldots, p_m \in F[x_1, \ldots, x_n]$ be polynomials such that for each $i \in \underline{m}$, each monomial of p_i contains at most k variables. Then there exists $J \subseteq \underline{n}$ such that $|J| \leq km(q-1)$ and $p_i(\boldsymbol{1}^{(J)}) = p_i(\boldsymbol{1})$ for all $i \in \underline{m}$.*

Proof. We proceed by induction on n. If $n \le km(q-1)$, then we take $J := \underline{n}$. For the induction step, we assume that $n > km(q-1)$.

We first produce a set $J_1 \subset \underline{n}$ such that $p_i(\boldsymbol{1}^{(J_1)}) = p_i(\boldsymbol{1})$ for all $i \in \underline{m}$. Seeking a contradiction, we suppose that no such J_1 exists. Following an idea from the proof of [4, Theorem 3.1], we consider the polynomials

$$\begin{array}{lcl} q_1(x_1,\dots,x_n) & := & \prod_{i=1}^{m}(1-(p_i(\boldsymbol{x})-p_i(\boldsymbol{1}))^{q-1}), \\ q_2(x_1,\dots,x_n) & := & x_1x_2\cdots x_n - q_1(x_1,\dots,x_n). \end{array}$$

We first show that for all $\boldsymbol{a} \in \{0,1\}^n$, $q_2(\boldsymbol{a}) = 0$. To this end, we first consider the case $\boldsymbol{a} = \boldsymbol{1}$. Then $q_2(\boldsymbol{a}) = 1 - \prod_{i=1}^{m} 1 = 0$. If $\boldsymbol{a} \in \{0,1\}^n \setminus \{\boldsymbol{1}\}$, then by the assumptions, there is $i \in \underline{m}$ such that $p_i(\boldsymbol{a}) \neq p_i(\boldsymbol{1})$. Then $1 - (p_i(\boldsymbol{a}) - p_i(\boldsymbol{1}))^{q-1} = 0$. Therefore $q_2(\boldsymbol{a}) = 0$. Hence the polynomial q_2 vanishes at $\{0,1\}^n$. By the Combinatorial Nullstellensatz [4, Theorem 1.1] applied to $g_j(x_j) := x_j^2 - x_j$, q_2 then lies in the ideal V of $F[x_1,\dots,x_n]$ generated by $G = \{x_j^2 - x_j \mid j \in \underline{n}\}$. Hence $x_1x_2\cdots x_n - q_1(x_1,\dots,x_n) \in V$. Since the leading monomials of the polynomials in G are coprime, G is a Gröbner basis of V (with respect to $x_1 > x_2 > \cdots > x_n$, lexicographic order, cf. [6, p.337]). Therefore, reducing $q_1(x_1,\dots,x_n)$ modulo G, we must obtain $x_1x_2\cdots x_n$ as the remainder (as defined, e.g., in [6, p.334]). Because of the form of all polynomials in G (all variables of g_j occur in the leading term of g_j), none of the reduction steps increases the number of variables in any monomial. Therefore, $q_1(x_1,\dots,x_n)$ must contain a monomial that contains all n variables. Computing the expansion of q_1 by multiplying out all products from its definition, we see that each monomial in q_1 contains at most $km(q-1)$ variables. Hence $n \le km(q-1)$, which contradicts the assumption $n > km(q-1)$. This contradiction shows that there is a set $J_1 \subset \underline{n}$ such that $p_i(\boldsymbol{1}^{(J_1)}) = p_i(\boldsymbol{1})$ for all $i \in \underline{m}$. Now we let $n' := |J_1|$, and we assume that $J_1 = \{j_1,\dots,j_{n'}\}$ with $j_1 < \cdots < j_{n'}$. For $i \in \underline{m}$, we define $p_i' \in F[y_1,\dots,y_{n'}]$ by

$$p_i'(x_{j_1},\dots,x_{j_{n'}}) = p_i(\boldsymbol{x}^{(J_1)}).$$

By the induction hypothesis, there exists $J_2 \subseteq \underline{n'}$ with $|J_2| \le km(q-1)$ such that $p_i'(\boldsymbol{1}^{(J_2)}) = p_i'(\boldsymbol{1})$ for all $i \in \underline{m}$. Now we define $J := \{j_t \mid t \in J_2\}$. We have $J \subseteq J_1$, and therefore $\boldsymbol{1}^{(J)} = (\boldsymbol{1}^{(J)})^{(J_1)}$. Then $p_i(\boldsymbol{1}^{(J)}) = p_i((\boldsymbol{1}^{(J)})^{(J_1)}) = p_i'(\boldsymbol{1}^{(J)}(j_1),\dots,\boldsymbol{1}^{(J)}(j_{n'})) = p_i'(\boldsymbol{1}^{(J_2)}) = p_i'(\boldsymbol{1}) = p_i(\boldsymbol{1}^{(J_1)}) = p_i(\boldsymbol{1})$, which completes the induction step. ◀

We will need the following special case of [15, Theorem 3.1]. For a set U, let $\mathcal{P}_{\le k}(U)$ denote the set $\{I \subseteq U : |I| \le k\}$ of subsets of U with at most k elements.

▶ **Theorem 2** (cf. [15, Theorem 3.1]). *Let $n \in \mathbb{N}$, let $k \in \mathbb{N}_0$, let p be a prime, and let $m \in \mathbb{N}$. Let $\varphi : \mathcal{P}_{\le k}(\underline{n}) \to \mathbb{Z}_p^m$. Then there is $U \subseteq \underline{n}$ with $|U| \le km(p-1)$ such that*

$$\sum_{J \in \mathcal{P}_{\le k}(\underline{n})} \varphi(J) = \sum_{J \in \mathcal{P}_{\le k}(U)} \varphi(J).$$

Proof. We denote the vector $\varphi(J)$ by $((\varphi(J))_1,\dots,(\varphi(J))_m)$, and we define m polynomial functions $f_1,\dots,f_m \in \mathbb{Z}_p[x_1,\dots,x_n]$ by

$$f_i(x_1,\dots x_n) := \sum_{J \in \mathcal{P}_{\le k}(\underline{n})} \big((\varphi(J))_i \cdot \prod_{j \in J} x_j\big).$$

for $i \in \underline{m}$. By Lemma 1, there is a subset U of $\underline{n}$ with $|U| \le km(p-1)$ such that for all $i \in \underline{m}$, we have $f_i(\boldsymbol{1}) = f_i(\boldsymbol{1}^{(U)})$. Hence $\sum_{J \in \mathcal{P}_{\le k}(\underline{n})} (\varphi(J))_i = f_i(\boldsymbol{1}) = f_i(\boldsymbol{1}^{(U)}) = \sum_{J \in \mathcal{P}_{\le k}(\underline{n}), J \subseteq U} (\varphi(J))_i = \sum_{J \in \mathcal{P}_{\le k}(U)} (\varphi(J))_i$. ◀

3 Absorbing components

Let A be a set, let 0_A be an element of A, let $\mathbf{B} = (B, +, -, 0)$ be an abelian group, let $n \in \mathbb{N}$, let $f : A^n \to B$, and let $I \subseteq \underline{n}$. By $\operatorname{Dep}(f)$ we denote the set $\{i \in \underline{n} \mid f \text{ depends on its } i\text{th argument}\}$. We say that f is *absorbing in its j th argument* if for all $\boldsymbol{a} = (\boldsymbol{a}(1), \dots, \boldsymbol{a}(n)) \in A^n$ with $\boldsymbol{a}(j) = 0_A$ we have $f(\boldsymbol{a}) = 0$. In the sequel, we will denote 0_A simply by 0. We say that f is *absorbing in I* if $\operatorname{Dep}(f) \subseteq I$ and for every $i \in I$, f is absorbing in its i th argument.

▶ **Lemma 3.** *Let A be a set, let 0 be an element of A, let $\mathbf{B} = (B, +, -, 0)$ be an abelian group, let $n \in \mathbb{N}$, and let $f : A^n \to B$. Then there is exactly one sequence $(f_I)_{I \subseteq \underline{n}}$ of functions from A^n to B such that for each $I \subseteq \underline{n}$, f_I is absorbing in I and $f = \sum_{I \subseteq \underline{n}} f_I$. Furthermore, each function f_I lies in the subgroup $\mathbf{F}$ of $\mathbf{B}^{A^n}$ that is generated by the functions $\boldsymbol{x} \mapsto f(\boldsymbol{x}^{(I)})$, where $I \subseteq \underline{n}$.*

Proof. We first prove the existence of such a sequence. To this end, we define f_I by recursion on $|I|$. We define $f_\varnothing(\boldsymbol{a}) := f(0, \dots, 0)$ and for $I \neq \varnothing$, we let

$$f_I(\boldsymbol{a}) := f(\boldsymbol{a}^{(I)}) - \sum_{J \subset I} f_J(\boldsymbol{a}).$$

By induction on $|I|$, we see that $\operatorname{Dep}(f_I) \subseteq I$ and that f_I lies in the subgroup $\mathbf{F}$. We will now show that each f_I is absorbing in I, and we again proceed by induction on $|I|$. Let $i \in I$, and let $\boldsymbol{a} \in A^n$ be such that $\boldsymbol{a}(i) = 0$. We have to show $f_I(\boldsymbol{a}) = 0$. We compute $f_I(\boldsymbol{a}) = f(\boldsymbol{a}^{(I)}) - \sum_{J \subset I} f_J(\boldsymbol{a})$. By the induction hypothesis, we have $f_J(\boldsymbol{a}) = 0$ for those J with $i \in J$. Hence $f(\boldsymbol{a}^{(I)}) - \sum_{J \subset I} f_J(\boldsymbol{a}) = f(\boldsymbol{a}^{(I)}) - \sum_{J \subseteq I \setminus \{i\}} f_J(\boldsymbol{a})$, and because of $\boldsymbol{a}^{(I)} = \boldsymbol{a}^{(I \setminus \{i\})}$, this is equal to $f(\boldsymbol{a}^{(I \setminus \{i\})}) - \sum_{J \subseteq I \setminus \{i\}} f_J(\boldsymbol{a}) = f(\boldsymbol{a}^{(I \setminus \{i\})}) - \sum_{J \subset I \setminus \{i\}} f_J(\boldsymbol{a}) - f_{I \setminus \{i\}}(\boldsymbol{a})$. By the definition of $f_{I \setminus \{i\}}$, the last expression is equal to $f_{I \setminus \{i\}}(\boldsymbol{a}) - f_{I \setminus \{i\}}(\boldsymbol{a}) = 0$. This completes the induction proof; hence each f_I is absorbing in I. In order to show $f = \sum_{I \subseteq \underline{n}} f_I$, we choose $\boldsymbol{a} \in A^n$ and compute $\sum_{I \subseteq \underline{n}} f_I(\boldsymbol{a}) = f_{\underline{n}}(\boldsymbol{a}) + \sum_{I \subset \underline{n}} f_I(\boldsymbol{a}) = f(\boldsymbol{a}^{(\underline{n})}) - \sum_{J \subset \underline{n}} f_J(\boldsymbol{a}) + \sum_{I \subset \underline{n}} f_I(\boldsymbol{a}) = f(\boldsymbol{a})$. This completes the proof of the existence of such a sequence.

For the uniqueness, assume that $f = \sum_{I \subseteq \underline{n}} f_I = \sum_{I \subseteq \underline{n}} g_I$ and that for all I, f_I and g_I are absorbing in I. We show by induction on $|I|$ that $f_I = g_I$. Let $I := \varnothing$. First we notice that $f(0, \dots, 0) = \sum_{J \subseteq \underline{n}} f_J(0, \dots, 0) = \sum_{J \subseteq \underline{n}} g_J(0, \dots, 0)$. Since f_J and g_J are absorbing, the summands with $J \neq \varnothing$ are 0, and thus $f_\varnothing(0, \dots, 0) = \sum_{J \subseteq \underline{n}} f_J(0, \dots, 0) = f(0, \dots, 0) = \sum_{J \subseteq \underline{n}} g_J(0, \dots, 0) = g_\varnothing(0, \dots, 0)$. Since both $f_\varnothing$ and $g_\varnothing$ are constant functions, they are equal. For the induction step, we assume $|I| \geq 1$. Let $\boldsymbol{a} \in A^n$. Then $\sum_{J \subseteq \underline{n}} f_J(\boldsymbol{a}^{(I)}) = \sum_{J \subseteq \underline{n}} g_J(\boldsymbol{a}^{(I)})$. Only the summands with $J \subseteq I$ can be nonzero, and therefore $\sum_{J \subseteq I} f_J(\boldsymbol{a}^{(I)}) = \sum_{J \subseteq I} g_J(\boldsymbol{a}^{(I)})$. By the induction hypothesis, $f_J = g_J$ for $J \subset I$. Therefore, $f_I(\boldsymbol{a}^{(I)}) = g_I(\boldsymbol{a}^{(I)})$. Since f_I and g_I depend only on the arguments at positions in I, we obtain $f_I(\boldsymbol{a}) = f_I(\boldsymbol{a}^{(I)}) = g_I(\boldsymbol{a}^{(I)}) = g_I(\boldsymbol{a})$. Thus $f_I = g_I$. ◀

Actually, the component f_I can be computed by $f_I(\boldsymbol{a}) = \sum_{J \subseteq I} (-1)^{|I|+|J|} f(\boldsymbol{a}^{(J)})$.

▶ **Definition 4.** *Let A be a set, let 0 be an element of A, let $\mathbf{B} = (B, +, -, 0)$ be an abelian group, let $n \in \mathbb{N}$, let $f : A^n \to B$, and let $J \subseteq \underline{n}$. Then we call the sequence $(f_I)_{I \subseteq \underline{n}}$ such that for each $I \subseteq \underline{n}$, f_I is absorbing in I, and $f = \sum_{I \subseteq \underline{n}} f_I$ the* absorbing decomposition *of f, and f_J the J-*absorbing component *of f. We define the* absorbing degree *of f by $\operatorname{adeg}(f) := \max\left(\{-1\} \cup \{|J| : J \subseteq \underline{n} \text{ and } f_J \neq 0\}\right)$.*

▶ **Theorem 5.** *Let A be a set, let 0 be an element of A, let p be a prime, let $k \in \mathbb{N}_0$, let $n \in \mathbb{N}$, and let $f_1, \dots, f_m : A^n \to \mathbb{Z}_p$. We assume that each f_i is of absorbing degree at most k. Let $\boldsymbol{a} \in A^n$. Then there is U with $|U| \le km(p-1)$ such that for all $i \in \underline{m}$, we have $f_i(\boldsymbol{a}) = f_i(\boldsymbol{a}^{(U)})$.*

Proof. We define a function $\varphi : \mathcal{P}_{\le k}(\underline{n}) \to \mathbb{Z}_p^m$ by $\varphi(J) := ((f_1)_J(\boldsymbol{a}), \dots, (f_m)_J(\boldsymbol{a}))$, where for $i \in \underline{m}$, $\big((f_i)_J\big)_{J \subseteq \underline{n}}$ is the absorbing decomposition of f_i. Then Theorem 2 yields a subset U of $\underline{n}$ with $|U| \le km(p-1)$ such that $\sum_{J \in \mathcal{P}_{\le k}(\underline{n})} \varphi(J) = \sum_{J \in \mathcal{P}_{\le k}(U)} \varphi(J)$. Since $(f_i)_J = 0$ for all J with $|J| > k$, we have $\sum_{J \in \mathcal{P}_{\le k}(\underline{n})} \varphi(J) = \sum_{J \in \mathcal{P}_{\le k}(\underline{n})} ((f_1)_J(\boldsymbol{a}), \dots, (f_m)_J(\boldsymbol{a})) = \sum_{J \subseteq \underline{n}} ((f_1)_J(\boldsymbol{a}), \dots, (f_m)_J(\boldsymbol{a})) = (f_1(\boldsymbol{a}), \dots, f_m(\boldsymbol{a}))$ and

$$\begin{aligned}\sum_{J \in \mathcal{P}_{\le k}(U)} \varphi(J) &= \sum_{J \in \mathcal{P}_{\le k}(U)} ((f_1)_J(\boldsymbol{a}), \dots, (f_m)_J(\boldsymbol{a})) \\ &= \sum_{J \subseteq U} ((f_1)_J(\boldsymbol{a}), \dots, (f_m)_J(\boldsymbol{a})) = \sum_{J \subseteq U} ((f_1)_J(\boldsymbol{a}^{(U)}), \dots, (f_m)_J(\boldsymbol{a}^{(U)})) \\ &= \sum_{J \subseteq \underline{n}} ((f_1)_J(\boldsymbol{a}^{(U)}), \dots, (f_m)_J(\boldsymbol{a}^{(U)})) = (f_1(\boldsymbol{a}^{(U)}), \dots, f_m(\boldsymbol{a}^{(U)})).\end{aligned}$$

◀

4 Polynomial mappings

In this section, we develop a property of polynomial mappings of finite supernilpotent algebras in congruence modular varieties. We call an algebra $\mathbf{A} = (A, +, -, 0, (f_i)_{i \in S})$ an *expanded group* if its reduct $\mathbf{A}^+ = (A, +, -, 0)$ is a group, an *expanded abelian group* if $\mathbf{A}^+$ is an abelian group, and an *expanded elementary abelian group* if $\mathbf{A}^+$ is elementary abelian, meaning that $\mathbf{A}^+$ is abelian and all its nonzero elements have the same prime order.

▶ **Lemma 6.** *Let $k, n \in \mathbb{N}$, let $\mathbf{A}$ be a k-supernilpotent expanded abelian group, and let $f \in \mathrm{Pol}_n(\mathbf{A})$. Then f is of absorbing degree at most k.*

Proof. Let $J \subseteq \underline{n}$ with $|J| > k$, and let f_J be the J-absorbing component of f. Let $m := |J|$ and let $J = \{i_1, \dots, i_m\}$. Using Lemma 3, we obtain that the function $g : A^m \to A$ defined by $g(a_{i_1}, \dots, a_{i_m}) := f_J(\mathbf{a})$ for $\mathbf{a} \in A^n$ is an absorbing function in $\mathrm{Pol}_m(\mathbf{A})$. Hence [1, Lemma 2.3] and the remark immediately preceding that Lemma yield that g is the zero function. Thus $f_J = 0$. Hence the absorbing degree of f is at most k. ◀

We first consider polynomial mappings of supernilpotent expanded elementary abelian groups of prime power order.

▶ **Theorem 7.** *Let $k, n, s, \alpha \in \mathbb{N}$, let $p \in \mathbb{P}$, and let $\mathbf{A}$ be a k-supernilpotent expanded elementary abelian group of order p^α. Let $F = (f_1, \dots, f_s) \in \mathrm{Pol}_n(\mathbf{A})^s$, and let $\boldsymbol{a} \in A^n$. Then there is $U \subseteq \underline{n}$ with $|U| \le ks\alpha(p-1)$ such that $F(\boldsymbol{a}) = F(\boldsymbol{a}^{(U)})$.*

Proof. We let π be a group isomorphism from $(A, +, -, 0)$ to $\mathbb{Z}_p^\alpha$, and for $a \in A$, we denote $\pi(a)$ by $(\pi_1(a), \dots, \pi_\alpha(a))$. For each $r \in \underline{s}$ and each $\beta \in \underline{\alpha}$, let $f_{r,\beta} : A^n \to \mathbb{Z}_p$ be defined by $f_{r,\beta}(\boldsymbol{a}) = \pi_\beta(f_r(\boldsymbol{a}))$; hence $f_{r,\beta}(\boldsymbol{a})$ is the βth component of $f_r(\boldsymbol{a})$. Since $f_r \in \mathrm{Pol}_n(\mathbf{A})$ and $\mathbf{A}$ is k-supernilpotent, Lemma 6 implies that each of these $f_{r,\beta}$ is of absorbing degree at most k. Setting $m := s\alpha$, Theorem 5 yields U with $|U| \le ks\alpha(p-1)$ such that $f_{r,\beta}(\boldsymbol{a}) = f_{r,\beta}(\boldsymbol{a}^{(U)})$ for all $r \in \underline{s}$ and $\beta \in \underline{\alpha}$. Then clearly $F(\boldsymbol{a}) = F(\boldsymbol{a}^{(U)})$. ◀

We apply this result to polynomial mappings of *direct products* of finite supernilpotent expanded elementary abelian groups. For a vector $\boldsymbol{a} \in A^n$, we call the number of its nonzero entries the *weight* of $\mathbf{a}$; formally, $\mathrm{wt}(\boldsymbol{a}) := |\{j \in \underline{n} : \boldsymbol{a}(j) \neq 0\}|$.

▶ **Theorem 8.** *Let $n, s, t, k_1, \dots, k_t \in \mathbb{N}$. For each $i \in \underline{t}$, let $\mathbf{B}_i$ a k_i-supernilpotent expanded elementary abelian group with $|\mathbf{B}_i| = p_i^{\alpha_i}$, where p_i is a prime and $\alpha_i \in \mathbb{N}$. Let $\mathbf{A} := \prod_{i=1}^{t} \mathbf{B}_i$, let $F \in \mathrm{Pol}_n(\mathbf{A})^s$, and let $\boldsymbol{a} \in A^n$. Then there is $\boldsymbol{y} \in A^n$ with $\mathrm{wt}(\boldsymbol{y}) \leq \sum_{i=1}^{t} k_i s \alpha_i (p_i - 1)$ such that $F(\boldsymbol{a}) = F(\boldsymbol{y})$.*

Proof. For $i \in \underline{t}$, let ν_i be the ith projection kernel. Applying Theorem 7 to $\mathbf{A}/\nu_i$, which is isomorphic to $\mathbf{B}_i$, and $\boldsymbol{b} := \boldsymbol{a}/\nu_i$, we obtain $U_i \subseteq \underline{n}$ with $|U_i| \leq k_i s \alpha_i (p_i - 1)$ such that $F^{\mathbf{A}/\nu_i}(\boldsymbol{b}^{(U_i)}) = F^{\mathbf{A}/\nu_i}(\boldsymbol{b})$. Lifting $\boldsymbol{b}^{(U_i)}$ to A, we obtain $(x_{i,1}, \dots, x_{i,n}) \in A^n$ such that $(x_{i,1}, \dots, x_{i,n})/\nu_i = \boldsymbol{b}^{(U_i)}$ and $x_{i,j} = 0$ for $j \in \underline{n} \setminus U_i$. Now for every $j \in \underline{n}$, we define $y_j \in A$ by the equations

$$y_j \equiv_{\nu_i} x_{i,j} \text{ for all } i \in \underline{t}.$$

For each $i \in \underline{t}$, we have $F(y_1, \dots, y_n)/\nu_i = F^{\mathbf{A}/\nu_i}(x_{i,1}/\nu_i, \dots, x_{i,n}/\nu_i) = F^{\mathbf{A}/\nu_i}(\boldsymbol{b}^{(U_i)}) = F^{\mathbf{A}/\nu_i}(\boldsymbol{b}) = F^{\mathbf{A}/\nu_i}(\boldsymbol{a}/\nu_i) = F(\boldsymbol{a})/\nu_i$. Hence $F(\boldsymbol{y}) = F(\boldsymbol{a})$. For $j \in \underline{n} \setminus (U_1 \cup \dots \cup U_t)$, and for all $i \in \underline{t}$, we have $x_{i,j} = 0$, and therefore $y_j = 0$. Hence the number of nonzero entries in $\boldsymbol{y}$ is at most $\sum_{i=1}^{t} |U_i| \leq \sum_{i=1}^{t} k_i s \alpha_i (p_i - 1)$. ◀

Now we consider *arbitrary* finite supernilpotent algebras in congruence modular varieties. In these algebras, we can introduce group operations preserving nilpotency using [1].

▶ **Lemma 9.** *Let $\mu \in \mathbb{N}$, let $\mathbf{A} = (A, (f_i)_{i \in S})$ be a finite supernilpotent algebra in a congruence modular variety all of whose fundamental operations have arity at most μ, and let $z \in A$. Let $t \in \mathbb{N}_0$, let $p_1, \dots, p_t$ be different primes, and let $\alpha_1, \dots, \alpha_t \in \mathbb{N}$ such that $|\mathbf{A}| = \prod_{i=1}^{t} p_i^{\alpha_i}$. For $i \in \underline{t}$, let $k_i := (\mu(p_i^{\alpha_i} - 1))^{\alpha_i - 1}$. Then there are operations $+$ (binary), $-$ (unary), 0 (nullary) on A such that $\mathbf{A}' = (A, +, -, 0, (f_i)_{i \in S})$ is isomorphic to a direct product $\prod_{i=1}^{t} \mathbf{B}'_i$, where each $\mathbf{B}'_i$ is a k_i-supernilpotent expanded elementary abelian group, and $0^{\mathbf{A}'} = z$.*

Proof. Since the result is true for $|A| = 1$, we henceforth assume $|A| \geq 2$. By [16], $\mathbf{A}$ is isomorphic to a direct product $\prod_{i=1}^{t} \mathbf{B}_i$ of nilpotent algebras of prime power order. We let $(\pi_1(a), \dots, \pi_t(a))$ denote the image of a of the underlying isomorphism. As a finite supernilpotent algebra in a congruence modular variety, $\mathbf{A}$ is nilpotent (cf. [1, Lemma 2.4]) and therefore has a Mal'cev term [9, Theorem 6.2]. We use [1, Theorem 4.2] to expand each $\mathbf{B}_i$ with operations $+_i$ and $-_i$ such that the expansion $\mathbf{B}'_i$ is a nilpotent expanded elementary abelian group with zero element $\pi_i(z)$. By [1, Theorem 1.2], $\mathbf{B}'_i$ is k_i-supernilpotent. ◀

We note that the supernilpotency degree of $\mathbf{A}'$ may be strictly larger than the supernilpotency degree of $\mathbf{A}$.

Combining these results, we obtain the following result on polynomial mappings on arbitrary finite supernilpotent algebras in congruence modular varieties.

▶ **Theorem 10.** *Let $\mu \in \mathbb{N}$, let $\mathbf{A}$ be a finite supernilpotent algebra in a congruence modular variety all of whose fundamental operations have arity at most μ. Let $p_1, \dots, p_t$ be distinct primes, and let $\alpha_1, \dots, \alpha_t \in \mathbb{N}$ such that $|\mathbf{A}| = \prod_{i=1}^{t} p_i^{\alpha_i}$. Let $F \in \mathrm{Pol}_n(\mathbf{A})^s$ be a polynomial map from A^n to A^s, and let $z \in A$. Then for every $\boldsymbol{a} \in A^n$ there is $\boldsymbol{y} \in A^n$ such that $F(\boldsymbol{y}) = F(\boldsymbol{a})$ and $|\{j \in \underline{n} : \boldsymbol{y}(j) \neq z\}| \leq s \sum_{i=1}^{t} (\mu(p_i^{\alpha_i} - 1))^{\alpha_i - 1} \alpha_i (p_i - 1) \leq s\mu^{-1} |A|^{\log_2 \mu + \log_2 |A|} \log_2 |A| \leq s|A|^{\log_2 \mu + \log_2 |A| + 1}$.*

Proof. Let $\mathbf{A}' = \prod_{i=1}^t \mathbf{B}'_i$ with $|\mathbf{B}'_i| = p_i^{\alpha_i}$ be the expansion of $\mathbf{A}$ produced by Lemma 9. Clearly, F is a also a polynomial map of $\mathbf{A}'$. Let $k_i = (\mu(p_i^{\alpha_i} - 1))^{\alpha_i - 1}$. Then Theorem 8 yields $\boldsymbol{y} \in A^n$ such that $|\{j \in \underline{n} : \boldsymbol{y}(j) \neq z\}| \leq \sum_{i=1}^t k_i s \alpha_i (p_i - 1)$. Using the obvious estimate $\alpha_i \leq \log_2 |A|$, we obtain

$$\begin{aligned}
&\sum_{i=1}^t k_i s \alpha_i (p_i - 1) = s \sum_{i=1}^t (\mu(p_i^{\alpha_i} - 1))^{\alpha_i - 1} \log_2 |A| (p_i - 1) \\
&\leq s \log_2 |A| \sum_{i=1}^t \mu^{\alpha_i - 1} (p_i^{\alpha_i})^{\alpha_i - 1} p_i^{\alpha_i} \leq s \log_2 |A| \sum_{i=1}^t \mu^{\log_2 |A| - 1} (p_i^{\alpha_i})^{\alpha_i} \\
&\leq s \mu^{\log_2 |A| - 1} \log_2 |A| \sum_{i=1}^t (p_i^{\alpha_i})^{\log_2 |A|} \leq s \mu^{\log_2 |A| - 1} \log_2 |A| (\sum_{i=1}^t p_i^{\alpha_i})^{\log_2 |A|} \\
&\leq s \mu^{\log_2 |A| - 1} \log_2 |A| (\prod_{i=1}^t p_i^{\alpha_i})^{\log_2 |A|} \leq s \mu^{\log_2 |A| - 1} \log_2 |A| \, |A|^{\log_2 |A|} \\
&\qquad\qquad\qquad = s \mu^{-1} |A|^{\log_2 \mu + \log_2 |A|} \log_2 |A| \leq s |A|^{\log_2 \mu + \log_2 |A| + 1}.
\end{aligned}$$

◀

5 Systems of equations

We will now explain how these results give a polynomial time algorithm for solving systems of a fixed number of equations over the finite supernilpotent algebra $\mathbf{A}$. The size m of a system of polynomial equations is measured as the length of the polynomial terms used to represent the system. For measuring the "running time" of our algorithm, we count the number of $\mathbf{A}$-operations: each such $\mathbf{A}$-operation, may, for example, be done by looking up one value in the operation tables defining $\mathbf{A}$.

▶ **Theorem 11.** *Let $\mathbf{A}$ be a finite supernilpotent algebra in a congruence modular variety all of whose fundamental operations are of arity at most μ, and let $s \in \mathbb{N}$. We consider the following algorithmic problem s-*PolSysSat*$(\mathbf{A})$:*

Given: *$2s$ polynomial terms $f_1, g_1, \ldots, f_s, g_s$ over $\mathbf{A}$.*
Asked: *Does the system $f_1 \approx g_1, \ldots, f_s \approx g_s$ have a solution in $\mathbf{A}$?*

Let m be the length of the input of this system, and let

$$e := s|A|^{\log_2 \mu + \log_2 |A| + 1} + 1.$$

*Then we can decide s-*PolSysSat*$(\mathbf{A})$ using at most $O(m^{e-1})$ evaluations of all terms occuring in the system. Therefore, we have an algorithm that determines whether a system of s polynomial equations over $\mathbf{A}$ has a solution using $O(m^e)$ many $\mathbf{A}$-operations.*

Proof. Let n be the number of different variables that occur in the given system. We may assume that these variables are $x_1, \ldots, x_n$, and that our system is $\bigwedge_{i=1}^s f_i(x_1, \ldots, x_n) \approx g_i(x_1, \ldots, x_n)$. We choose an element $z \in A$, and we will show: if this system has a solution in A^n, then it has a solution in

$$C := \{\boldsymbol{y} \in A^n : |\{j \in \underline{n} : \boldsymbol{y}(j) \neq z\}| \leq e - 1\}.$$

For proving this claim, we first observe that $\mathbf{A}$ is a finite nilpotent algebra in a congruence modular variety, and it therefore has a Mal'cev term d. We consider the polynomial map

$H = (h_1, \dots, h_s)$, where $h_i(\boldsymbol{x}) := d(f_i(\boldsymbol{x}), g_i(\boldsymbol{x}), z)$ for $i \in \underline{s}$ and $\boldsymbol{x} \in A^n$. Since $\boldsymbol{a}$ is a solution of the system, $H(\boldsymbol{a}) = (z, z, \dots, z)$. By Theorem 10, there is $\boldsymbol{y} \in C$ such that $H(\boldsymbol{y}) = H(\boldsymbol{a})$. Then for every $i \in \underline{s}$, we have $d(f_i(\boldsymbol{y}), g_i(\boldsymbol{y}), z) = z$. By [9, Corollary 7.4], the function $x \mapsto d(x, g_i(\boldsymbol{y}), z)$ is injective. Since $d(f_i(\boldsymbol{y}), g_i(\boldsymbol{y}), z) = z = d(g_i(\boldsymbol{y}), g_i(\boldsymbol{y}), z)$, this injectivity implies that $f_i(\boldsymbol{y}) = g_i(\boldsymbol{y})$. Hence $\boldsymbol{y}$ is a solution that lies in C.

The algorithm for solving the system now simply evaluates the system at all places in C; if a solution is found, the answer is "yes". If we find no solution inside C , we answer "no", and by the argument above, we know that in this case, the system has no solution inside $\mathbf{A}^n$ at all.

We now estimate the complexity of this procedure: There is a $c \in \mathbb{N}$ such that for all $n \in \mathbb{N}$, $|C| \le cn^{e-1}$, hence we have to do $O(n^{e-1})$ evaluations of all the terms f_i, g_i in the system. Such an evaluation can be done using at most $O(m)$ many $\mathbf{A}$-operations. Since the length of the input m is at least the number of variables n occuring in it, this solves s-PolSysSat($\mathbf{A}$) using at most $O(m^e)$ many $\mathbf{A}$-operations. ◀

We remark that the exponent e involves neither the nilpotency nor the supernilpotency degree of the algebra $\mathbf{A}$; this is a result of the fact that Theorem 1.2 from [1] bounds the supernilpotency degree of $\mathbf{A}$ in terms of only μ, $|A|$, and the height of the congruence lattice of $\mathbf{A}$, which is bounded from above by $\log_2 |A|$. In the same vein, we can now also bound the exponent in the complexity bound for the *identity checking* or *polynomial equivalence* problem for supernilpotent algebras (cf. [2, Theorem 2.2]). For every algebra $\mathbf{A}$ satisfying the assumptions of Theorem 11, there is $c \in \mathbb{N}$ such that for any two n-ary polynomial terms $s(x_1, \dots, x_n)$ and $t(x_1, \dots, x_n)$, we can check whether $\mathbf{A} \models s \approx t$ using at most cn^d evaluations of both s and t, where $d := (\mu(|A| - 1))^{\log_2 |A| - 1}$ comes from Corollary 1.3 of [1].

6 Circuit satisfiability

With every finite algebra $\mathbf{A}$, [13] associates a number of computational problems that involve circuits whose gates are taken from the fundamental operations of $\mathbf{A}$. One of these problems is SCsat($\mathbf{A}$). It takes as an input $2s$ circuits $f_1, g_1, \dots, f_s, g_s$ over $\mathbf{A}$ with n input variables, and asks whether there is an $\boldsymbol{a} \in A^n$ such that the evaluations at $\boldsymbol{a}$ satisfy $f_i(\boldsymbol{a}) = g_i(\boldsymbol{a})$ for all $i \in \underline{s}$. For finite algebras of finite type (i.e., with finitely many fundamental operations) in congruence modular varieties, [18, Corollary 3.13] implies that SCsat($\mathbf{A}$) is in P when $\mathbf{A}$ is abelian, and NP-complete otherwise. However, if we restrict the number s of circuits, we obtain a different problem, which we call s-SCsat($\mathbf{A}$) in the sequel. Obviously, 1-SCsat($\mathbf{A}$) is the circuit satisfiability problem called Csat($\mathbf{A}$) in [13]. The method used to prove Theorem 11 immediately yields:

▶ **Theorem 12.** *Let $\mathbf{A}$ be a finite supernilpotent algebra of finite type in a congruence modular variety, and let $s \in \mathbb{N}$. Then s-SCsat($\mathbf{A}$) is in* P.

Hence a supernilpotent, but not abelian algebra $\mathbf{A}$ has s-SCsat($\mathbf{A}$) in P, whereas SCsat is NP-complete. In the converse direction, Theorem 9.1 from [13] has the following corollary.

▶ **Corollary 13.** *Let $\mathbf{A}$ be a finite algebra of finite type from a congruence modular variety. If $\mathbf{A}$ has no homomorphic image $\mathbf{A}'$ such that* 2-SCsat($\mathbf{A}'$) *is* NP-*complete, then $\mathbf{A}$ is nilpotent.*

Proof. Suppose that $\mathbf{A}$ has a homomorphic image $\mathbf{A}'$ for which Csat($\mathbf{A}'$) is NP-complete. Then also 2-SCsat($\mathbf{A}'$) is NP-complete because an algorithm solving 2-SCsat can be used to solve an instance $(\exists \boldsymbol{a})(f(\boldsymbol{a}) = g(\boldsymbol{a}))$ of Csat($\mathbf{A}'$) by solving 2-SCsat on the input

$(\exists \boldsymbol{a})(f(\boldsymbol{a}) = g(\boldsymbol{a})\ \&\ f(\boldsymbol{a}) = g(\boldsymbol{a}))$. Thus the assumptions imply that for no homomorphic image $\mathbf{A}'$ of $\mathbf{A}$, the problem CSat($\mathbf{A}'$) is NP-complete. Now by [13, Theorem 9.1], $\mathbf{A}$ is isomorphic to $\mathbf{N} \times \mathbf{D}$, where $\mathbf{N}$ is nilpotent and $\mathbf{D}$ is a subdirect product of 2-element algebras each of which is polynomially equivalent to a two element lattice. If $|\mathbf{D}| > 1$, then there is a homomorphic image $\mathbf{A}_2$ of $\mathbf{A}$ such that $\mathbf{A}_2$ is polynomially equivalent to a two element lattice. By [11], 2-SCSat($\mathbf{A}_2$) is NP-complete, contradicting the assumptions. Hence $|\mathbf{D}| = 1$, and therefore $\mathbf{A}$ is nilpotent. ◀

References

1 E. Aichinger. Bounding the free spectrum of nilpotent algebras of prime power order. *Israel J. Math.*, 230(2):919–947, 2019.

2 E. Aichinger and N. Mudrinski. Some applications of higher commutators in Mal'cev algebras. *Algebra Universalis*, 63(4):367–403, 2010.

3 E. Aichinger, N. Mudrinski, and J. Opršal. Complexity of term representations of finitary functions. *Internat. J. Algebra Comput.*, 28(6):1101–1118, 2018.

4 N. Alon. Combinatorial Nullstellensatz. *Combin. Probab. Comput.*, 8(1-2):7–29, 1999. Recent trends in combinatorics (Mátraháza, 1995).

5 D. Brink. Chevalley's theorem with restricted variables. *Combinatorica*, 31(1):127–130, 2011.

6 D. Eisenbud. *Commutative algebra*. Springer-Verlag, New York, 1995.

7 A. Földvári. The complexity of the equation solvability problem over semipattern groups. *Internat. J. Algebra Comput.*, 27(2):259–272, 2017.

8 A. Földvári. The complexity of the equation solvability problem over nilpotent groups. *Journal of Algebra*, 495:289–303, 2018.

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

10 M. Goldmann and A. Russell. The complexity of solving equations over finite groups. *Inform. and Comput.*, 178(1):253–262, 2002.

11 T. Gorazd and J. Krzaczkowski. The complexity of problems connected with two-element algebras. *Rep. Math. Logic*, 46:91–108, 2011.

12 G. Horváth. The complexity of the equivalence and equation solvability problems over nilpotent rings and groups. *Algebra Universalis*, 66(4):391–403, 2011.

13 P. M. Idziak and J. Krzaczkowski. Satisfiability in multi-valued circuits. In *LICS '18—33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 550–558. ACM, New York, 2018.

14 G. Károlyi and C. Szabó. The complexity of the equation solvability problem over nilpotent rings. Manuscript available at `http://web.cs.elte.hu/~csaba/publications/`, 2015.

15 G. Károlyi and C. Szabó. Evaluation of Polynomials over Finite Rings via Additive Combinatorics. *ArXiv e-prints*, 1809.06543, 2018. `arXiv:1809.06543`.

16 K. A. Kearnes. Congruence modular varieties with small free spectra. *Algebra Universalis*, 42(3):165–181, 1999.

17 M. Kompatscher. The equation solvability problem over supernilpotent algebras with Mal'cev term. *Internat. J. Algebra Comput.*, 28(6):1005–1015, 2018.

18 B. Larose and L. Zádori. Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. *Internat. J. Algebra Comput.*, 16(3):563–581, 2006.

19 R. N. McKenzie, G. F. McNulty, and W. F. Taylor. *Algebras, lattices, varieties, Volume I*. Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, California, 1987.

Listing Induced Steiner Subgraphs as a Compact Way to Discover Steiner Trees in Graphs

Alessio Conte
Dipartimento di Informatica, Università di Pisa, Italy
conte@di.unipi.it

Roberto Grossi
Dipartimento di Informatica, Università di Pisa, Italy
grossi@di.unipi.it

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

Andrea Marino
Dipartimento di Statistica, Informatica, Applicazioni, Università di Firenze, Italy
andrea.marino@unifi.it

Takeaki Uno
National Institute of Informatics, Tokyo, Japan
uno@nii.ac.jp

Kunihiro Wasa
National Institute of Informatics, Tokyo, Japan
wasa@nii.ac.jp

Abstract

This paper investigates induced Steiner subgraphs as a variant of the classical Steiner trees, so as to compactly represent the (exponentially many) Steiner trees sharing the same underlying induced subgraph. We prove that the enumeration of all (inclusion-minimal) induced Steiner subgraphs is harder than the well-known Hypergraph Transversal enumeration problem if the number of terminals is not fixed. When the number of terminals is fixed, we propose a polynomial delay algorithm for listing all induced Steiner subgraphs of minimum size. We also propose a polynomial delay algorithm for listing the set of minimal induced Steiner subgraphs when the number of terminals is 3.

2012 ACM Subject Classification Mathematics of computing → Graph enumeration

Keywords and phrases Graph algorithms, enumeration, listing and counting, Steiner trees, induced subgraphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.73

Funding This work was partially supported by JST CREST grant number JPMJCR1401 and JPMJCR18K3 Japan, by JSPS KAKENHI Grant Number JP19K20350 Japan, and by MIUR, Italy. Mamadou M. Kanté is supported by French Agency for Research under the GraphEN project (ANR-15-CE40-0009).

1 Introduction

Paths and cycles are the basic components for connecting nodes in undirected graphs. It is natural to ask, for a given subset of nodes, how to connect them in a component of minimal cost, where the cost is, e.g., the total sum of the weights on the edges or the number of nodes.

Classical instances of this problem follow two main patterns [7]. Some are *edge*-centric components in (usually) weighted graphs, such as shortest paths/cycles, spanning trees, Steiner trees, k-edge-connected components. Others are *node*-centric components in unweighted graphs, such as induced paths, chordless cycles (holes), k-node-connected components.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 73; pp. 73:1–73:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The above is not actually a dichotomy of edge- vs. node-centric components as, for example, node-weighted Steiner trees lay at the cross of the two. Nevertheless, it seems that most of the literature has been devoted to edge-centric components.

For instance, in the rich research area on Steiner trees, the online compendium in [12] reports over 60 variations of the Steiner tree problem, but just a couple are node-centric (survivable network design problem [3] and node weighted generalized Steiner tree [11]). The interest in the Steiner tree problem and its variants is well rooted in real world applications. Some examples can be seen in the 11th DIMACS challenge [8], which aimed at identifying a benchmark for algorithms and data generators, with emphasis on network optimization, computer-aided design, phylogenetic tree reconstruction and other applications. It is also of great interest in implementation challenges of the parameterized community, *e.g.*, [16], thus we believe it is worth investigating its natural extensions in node-centric scenarios.

Induced Steiner subgraphs. Special node-centric components have been introduced by Telle and Villanger [17] as a generalization of the induced paths.[1] In this paper, we study these components, hereafter renamed *induced Steiner subgraphs* as they seem to us a natural extension to the scenario of Steiner trees. Given an undirected graph $G = (V(G), E(G))$ and a subset $W \subseteq V(G)$ of $t = |W|$ nodes, called *terminals*, we want a set of nodes, minimal under inclusion, that connects all nodes in W.

▶ **Definition 1** (Induced Steiner Subgraph). *Given $W \subseteq V(G)$, an Induced Steiner Subgraph of G is a set of nodes $S \subseteq V(G) \setminus W$ such that the induced subgraph $G[S \cup W]$ is connected. We say that S is a (inclusion wise)* Minimal *Induced Steiner Subgraph (*MISS*) if there is no $S' \subset S$ such that $G[S' \cup W]$ is connected. The nodes in S are called Steiner points.*[2]

An example of a MISS is given in Figure 1, where we illustrate also its connection with the classical notion of Steiner tree. Similarly to how Steiner trees generalize paths between two nodes, it can be noted that MISS's generalize induced paths, which correspond to MISS's for $|W| = 2$. On the other extreme, for $W = V(G)$, Steiner trees correspond to spanning trees of the graph, while there would be a single empty MISS corresponding to the whole graph.

Definition 1 makes sense in domains where nodes are preferred to edges in components, for example in network design [11] and graph database query optimization [2, 14]. In these cases, rather than a minimal set of edges that ensures connection (i.e. a tree), a minimal set of connected nodes and their induced subgraph (i.e. a MISS) are sought for. Note that the notion of induced Steiner subgraph can be easily extended to graphs with weights on the nodes, and MISS's correspond to solutions of minimal cost (otherwise their cost could be reduced by removing nodes). Our approach focuses on unweighted graphs, where Steiner trees also have many applications [12].

Results. In this paper we consider the problem of *output-sensitive* enumeration of MISS's, which corresponds to bounding the cost to a polynomial of the input and output size.

- In Section 3 we show that the problem is harder than the well-known TRANS-ENUM problem [10] for an arbitrary number of terminals. To classify its output-sensitive

[1] In [17], these are called MINIMAL T-CONNECTING SETS and are motivated by the study of the 2-DISJOINT CONNECTED SUBGRAPHS problem which is the following: given a connected graph $G = (V, E)$ and two disjoint subsets of terminal vertices $Z_1, Z_2 \subseteq V$, decide whether there exists a partition A_1, A_2 of $V(G)$, with $Z_1 \subseteq A_1$, $Z_2 \subseteq A_2$ and $G[A_1], G[A_2]$ both connected.

[2] We abuse the terminology a bit and refer by S also to the induced subgraph $G[S \cup W]$, as we enumerate all feasible S's while keeping the same set W. This simplifies notation in the rest of the paper.

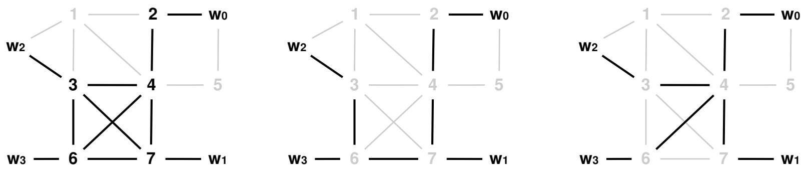


Figure 1 A MISS (left) and two Steiner trees corresponding to the same MISS (center and right).

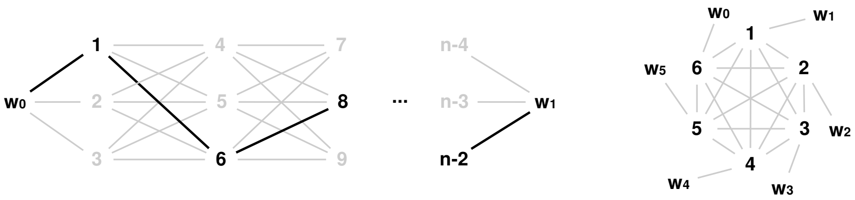


Figure 2 Left: A graph with $\Omega(3^{n/3})$ minimum Steiner trees and MISS's. Right: A graph with 1 MISS and n^{n-2} minimum Steiner trees (by Cayley's formula).

enumeration complexity, we consider the problem when the number of terminals $t = |W|$ is fixed. Concerning this, we show that the EXTENSION PROBLEM (see Section 3.2) is NP-complete already for $t \geq 2$, meaning that the sub-problems of any backtracking algorithm [10] should be carefully designed to avoid incurring instances of NP-complete problems.

- In this scenario, we propose a polynomial delay algorithm in Section 4 when the number of terminals is $t = 3$. (Note that the delay is the time required to list the "next" solution.) This is the first non-trivial case, as for $t = 1$ the problem is not defined, and for $t = 2$ it is the well-known problem of listing induced paths.
- Furthermore, we investigate in Section 2 the relationships of MISS's and Steiner trees, proving properties which allow us to use the algorithm of Dourado et al. [9] as a subroutine in order to list just the MISS's of *minimum* size with output-sensitive guarantees, for a bounded number t of terminals.

Discussion. Given a MISS S for W, we observe that each spanning tree in the induced subgraph $G[S \cup W]$ is a Steiner tree for W. Also the vice versa holds, so this could be useful for the packing edge-disjoint Steiner tree problem, which received attention in recent years [1, 4, 13] and consists of finding a largest possible set of Steiner trees that do not share edges or nodes.

If we look for an *individual* min-size Steiner subgraph for W (i.e., with the least number of nodes), it can be obtained as the subgraph induced by the Steiner points of a min-size Steiner tree for W. A different situation occurs instead if we look for *all* min-size induced Steiner subgraphs for W. A number of deep results for Steiner trees could be employed: for example, [9] proposes an algorithm for listing them with linear delay, assuming that the number $t = |W|$ of terminals is $O(1)$; [15] introduces a general enumeration framework for matroids with certain properties, which can be applied to generalized Steiner trees, containing as special cases minimal Steiner trees and minimal Steiner forests. Unfortunately, this approach gives rise over and over to the same min-size Steiner subgraph in our case, as there can be exponentially many Steiner trees for the same set of Steiner points (Fig. 2(left)).

Here comes an interesting feature of a MISS, as it is a compact representation of many Steiner trees. Observing that a min-size Steiner subgraph for W is a MISS for W of minimum size, we can thus list the MISS's for W twice: the first time to discover the minimum size, and the second time to single out the MISS's of that size (i.e. min-size induced Steiner subgraphs).

This approach can be dramatically more efficient than listing min-size Steiner trees to catch min-size induced Steiner subgraphs (see Fig. 2 (right)). On the other hand, the results in this paper (see Section 2.2) show the latter to be a viable approach if W has bounded size.

As for weighted graphs, we observe that the min-edge-weight Steiner tree problem reduces to min-node-weight Steiner subgraph problem.[3] Once we can list MISS's efficiently, we can also find min-node-weight induced Steiner subgraphs and thus min-edge-weight Steiner trees.

In summary, finding MISS's subsumes finding min-node-weight, min-edge-weight, and min-size Steiner trees, and becomes a sort of "universal" problem for this family of connectivity problems on graphs that compactly represent these trees. This motivates the problem of *listing* MISS's efficiently, which is central to this paper.

Previous results on MISS enumeration were already obtained in [17] from an *input-sensitive* point of view:[4] the authors proved that the number of MISS's for t terminals, in any graph of n nodes, is at most $\binom{n-t}{t-2} \cdot 3^{\frac{n-t}{3}}$, and propose an *input-sensitive* algorithm running in $O^*(\binom{n-t}{t-2} \cdot 3^{\frac{n-t}{3}})$ total time, where the O^* notation ignores factors polynomial in n. Here we consider the opposite direction, i.e., output-sensitive enumeration.

Preliminaries. We refer to [7] for our graph terminology. Let $G = (V(G), E(G))$ be an undirected graph, where $V(G)$ is its node set and $E(G)$ its edge set. $N(v)$ is the set of neighbors of a node v. When G is clear from the context we will refer to node and edge sets simply as V and E. For a set of nodes $A \subseteq V(G)$, let $E[A] \subseteq E(G)$ be the set of edges whose endpoints are both in A, and denote by $G[A] = (A, E[A])$ the graph thus obtained, also called the subgraph *induced* by A. A subgraph is a forest if it is acyclic, and a tree if it is a connected forest. A node $v \in I$ is a *leaf* if v has degree 1 in $G[I]$. Removing a leaf from a tree yields a tree. Regarding the Steiner Tree/Subgraph problem, we will refer to the set of terminal nodes as W, and to its size $|W|$ as t. We recall that an induced Steiner subgraph is a set of nodes S such that $G[S \cup W]$ is connected, and is minimal if inclusion-minimal. For a MISS S, we call *junctions* the nodes in $S \cup W$ which, in $G[S \cup W]$, have either degree ≥ 3, or are terminals with degree ≥ 2.

The notions for induced Steiner subgraphs and MISS given in Definition 1 can be applied to any subset of the terminals in W: for example, S is a MISS for $X \subseteq W$ if $G[S \cup X]$ is connected and so on. Without loss of generality, we can assume that no two terminals in W are connected by an edge: otherwise we can contract that edge into a single node and eliminate the resulting duplicated edges, preserving a one-to-one correspondence between solutions of the old graph and the new. In the following, given a set of nodes S, let $W(S)$ be the set of terminals in W that have neighbors in S. Note that S is an induced Steiner subgraph for $X = W(S)$ as long as $G[S \cup W(S)]$ is connected. Let us make the following observation before continuing.

▶ **Fact 1.** *Let S be a* MISS *for a set W of terminals. Then, $G[S \cup W]$ is not 2-connected. Moreover, $G[S \cup W]$ cannot contain a leaf vertex $v \in S$.*

Proof. Because no two terminals are adjacent, if $G[S \cup W]$ is 2-connected, then $|S| \geq 2$. By definition of 2-connectedness, for any vertex $v \in S$, we have that $G[(S \setminus \{v\}) \cup W]$ is connected, contradicting the minimality of S.

If $G[S \cup W]$ contains a leaf vertex $v \in S$, then $G[(S \setminus \{v\}) \cup W]$ is still connected as no path can contain v as an internal vertex, contradicting the minimality of S. ◀

[3] It suffices to replace each each edge $\{x, y\}$ with weight w by two edges $\{x, z\}$ and $\{z, y\}$ where z is a new node of weight w and x and y have weight zero [11]

[4] In exact exponential-time algorithms, a input-sensitive cost measures the running time in the input length.

2 Properties of Steiner Subgraphs and Listing Minimum Steiner Subgraphs

We are interested in this section in the enumeration of MISS's of minimum size, and refer to them as *minimum* MISS*'s*. In this section we prove some useful properties of MISS's in general, and some special properties of minimum MISS's, to outline the relationship between minimum Steiner trees and MISS's (Section 2.1) and to get an output-sensitive algorithm for minimum MISS's (Section 2.2).

2.1 Relationship with minimum Steiner trees

We first prove that, for a given set W of terminals, one minimum MISS corresponds to many Steiner trees (Lemma 2). We then prove an upper bound (in terms of t) for the number of junctions and the degree of the nodes involved in a MISS. We highlight here that both upper bounds hold for any MISS, minimum or not. Aside of telling that MISS's have strong topological properties, these bounds allow us to obtain the converse of Lemma 2, that is how to turn MISS's into Steiner trees (Lemma 5).

One-to-many correspondence to minimum Steiner trees. A minimum Steiner tree T for the set W of terminals is clearly a spanning tree of the subgraph $G[M \cup W]$ induced by the set $M = V(T) \setminus W$ of nodes plus the terminals, and thus M is a minimum MISS for W. On the other hand, consider any minimum MISS M for W: each spanning tree T of $G[M \cup W]$ is actually a minimum Steiner tree for W. For example, the two trees in Figure 1 correspond to the same MISS.

▶ **Lemma 2.** *Given a set W of terminals and a minimum* MISS *M for W, any spanning tree T of the induced subgraph $G[M \cup W]$ is a minimum Steiner tree for W.*

Proof. All the terminals in W are connected through T by definition of spanning tree, so T is a Steiner tree, not necessarily of minimum size (and the terminals are not necessarily all in its leaves). Suppose by contradiction that there exists another Steiner tree T' of size smaller than T (where T' can use nodes or edges outside $G[M \cup W]$). Then the subgraph $G[M' \cup W]$ induced by the set $M = V(T') \setminus W$ of nodes of T' would be a MISS with fewer nodes than $G[M \cup W]$, contradicting the fact that the latter is a minimum MISS. ◀

Upper bound on the number of junctions. Consider a MISS S (not necessarily minimum), and recall that a *junction* is either a node of S with degree at least 3 in the induced subgraph $G[S \cup W]$, or a terminal in W with degree at least 2 in $G[S \cup W]$. Although S is minimal, it can contain arbitrarily long induced paths independently of the number t of terminals. Still, we can bound the number of junctions in terms of t.

▶ **Lemma 3.** *Given a set W of terminals, any* MISS *S contains at most $3t$ junctions.*

Proof. By induction on the number of terminals. Recalling that $t = |W|$, if $t \leq 3$ then the claim is trivially true (see also Fact 2). Assume then that $t \geq 4$.

Consider the induced subgraph $G[S \cup W]$ ignoring the rest of G. In order to prove our claim, we replace each junction terminal v' with a new node v, and link v' and v together with an edge, so that v' becomes a leaf terminal and v is a junction. Now, as no junction can have degree 2, we replace each induced path $x_1, x_2, \ldots, x_\ell$ by an edge $\{x_1, x_\ell\}$ when x_1 and x_ℓ have degree $\neq 2$.

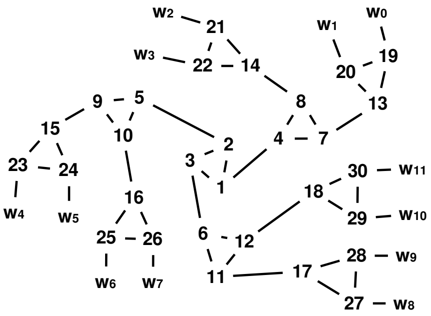


Figure 3 A MISS with 12 terminals and 30 junctions. Suitably enlarging the structure sets the junctions/terminals ratio arbitrarily close to 3.

Let G' be the resulting graph. Note that G' and $G[S \cup W]$ have the same number of terminals and junctions by construction. As a result, it suffices to prove our claim using G'. Note also that each node v in G' has degree 1 if and only if it is a terminal, and at least 3 otherwise. If G' is a tree, then the claim holds as the number of internal nodes of degree 3 in a tree is upper bounded by the number of leaves.

Let us assume that G' is not a tree. By Fact 1 we know that $G[S \cup W]$ is not 2-connected and by construction G' is also not 2-connected. Let T be the tree of bi-connected components of G' (called blocks). Because all non-terminal nodes in G' are junctions, each leaf block C of T is an edge $\{v, w\}$ with v a junction node and w a terminal because all the terminal nodes are non-adjacent and a leaf block has at most one non-terminal node. Let C be a block all of whose neighbors, except possibly 1, are leaf blocks. Such a block always exists, e.g., we could take a non-leaf block of maximum depth after rooting the tree arbitrarily. If except C, all the other blocks are leaf blocks (meaning that T is a star), then we are done because C is 2-connected, meaning any node removal does not disconnect it, meaning in turn any node must be connected to a leaf-terminal in order for it not to be redundant, so C contains at most t nodes. Otherwise, T is not a star and then for all possible choices for C let us take the one with biggest size. Let $w_1, \ldots, w_p$ be the terminals adjacent to junction nodes in C. Clearly, by the choice of C, $p \geq 2$ because either C has size ≥ 3, and at least two of the junction nodes are adjacent to terminals, or C has size 2 and at least one of the junction nodes is adjacent to two terminals. Let v be the junction node of C belonging to another block that is not a leaf block (v exists by the choice of C). Notice that C has size at most $p+1$ because except possibly v, all the junction nodes in C are adjacent to at least one terminal. Let $G'' = G' \setminus (\{w_1, \ldots, w_p\} \cup (C \setminus \{v\}))$ and consider v as a terminal in G''. It is not hard to check that G'' is a MISS for $(W \setminus \{w_1, \ldots, w_p\}) \cup \{v\}$. By induction G'' has at most $3(t-p+1)$ junctions because G'' has $t-p+1$ terminals. And then G has at most $3(t-p+1)+(p+1)$ junctions, which is bounded by $3t$. ◀

Furthermore, Figure 3 shows Lemma 3 to be tight (up to constant additive factors).

Upper bound on the node degree. Although any MISS S can be arbitrarily large independently of the number t of terminals, we prove that any node cannot have degree larger than t in $G[S \cup W]$.

▶ **Lemma 4.** *Given a set W of terminals and any* MISS *S for W and any node $v \in S \cup W$, $|N(v) \cap (S \cup W)| \leq t$.*

Proof. Let G' be the induced subgraph $G[S \cup W]$. Let us start with the case $v \in S$. For each $w \in W$, consider one shortest path between v and w in G', and let S' be the set of nodes containing all the nodes on these t shortest paths, excluding the nodes in W. Note that $S' \subseteq S$ by definition. As they are all shortest paths (hence induced paths with no shortcut edges), v can only be adjacent to the next node in each such path, and no other: thus $|N(v) \cap (S' \cup W)| \leq t$. Since all terminals are connected to each other using these paths, clearly S' is a Steiner induced subgraph. Since S is minimal, the only option is $S' = S$. The case $v \in W$ follows along the same lines. ◀

Removing edges to get a minimum Steiner trees. Consider any MISS S and select a subset of edges in the induced subgraph $G[S \cup W]$ to obtain a Steiner tree T. Lemma 3 and 4 imply the result that any T can be obtained from S by removing at most $3t^2$ edges from $G[S \cup W]$. Note that the number of edges in $G[S \cup W]$ could be much larger, but most of them are incident to degree-2 nodes of $G[S \cup W]$, and should be taken to form T (otherwise some terminals in W would be disconnected from the others). This property holds for the special case of minimum MISS, and hence minimum Steiner tree, immediately giving the following result.

▶ **Lemma 5.** *Given a set W of terminals and any* MISS *S for W, every Steiner tree T contained in $G[S \cup W]$ can be obtained by removing at most $3t^2$ edges.*

2.2 Listing minimum Steiner subgraphs

The properties of Steiner subgraphs seen so far allow us to design an algorithm for listing minimum Steiner subgraphs, parameterized in t. Consider a minimum MISS M, and the subgraph $G_M = G[M \cup W]$. By Lemma 2 each spanning tree of G_M is a minimum Steiner tree. By Lemma 5, G_M is turned into a spanning tree by removing α edges, for some $\alpha \leq 3t^2$. A closer look gives a suggestion on how to do that: even if the number of nodes and edges in G_M can be much larger than $3t^2$, there are at most $3t$ junctions by Lemma 3, with degree greater than 2; the rest of the (possibly many) nodes have all degree 2, and thus their incident edges must be taken to keep G_M connected. Hence, the α edges are incident to these junctions, each of degree at most t in G_M by Lemma 4, thus giving at most $3t^2$ edges to select from. As there are at most 2^{3t^2} possible choices for the α edges, there are $O(2^{3t^2})$ minimum Steiner trees yielding the same minimum Steiner subgraph M.

We exploit this observation by using [9] to list minimum Steiner trees of G with $O(n^2m + n^{t-2})$ preprocessing time and space, and $O(n)$ delay. To avoid duplications, as the same minimum MISS M can be obtained $O(2^{3t^2})$ times with distinct minimum Steiner trees, we simply define a “canonical spanning tree” T^* of M as an arbitrary spanning tree obtained from G_M with some deterministic procedure: when the algorithm in [9] yields T, we output its corresponding minimum MISS only if $T = T^*$, and discard it otherwise (note that this computation takes an additional $O(m)$ time per solution).

▶ **Theorem 6.** *Minimum Steiner subgraphs of G for a terminal set W can be listed with $O(n^2m + n^{t-2})$ preprocessing time and space, in $O(m \cdot 2^{3t^2})$ time per solution.*

3 Negative Results on Minimal Steiner Subgraphs

As for the classical Steiner tree problem, finding just one MISS is easy, but finding the smallest MISS, i.e., the one having minimum number of nodes, is NP-hard: as we saw in Section 2 any spanning tree of a MISS with k nodes is a Steiner tree with exactly $k - 1$ edges. This means that there is a one-to-many correspondence between MISS and Steiner trees of smallest size, and that finding a smallest MISS solves the unweighted Steiner tree problem.

Concerning listing MISS's, in Section 3.1 we show that this problem is actually harder than listing minimal hypergraph transversals. In particular, this result implies that an output-polynomial algorithm for MISS's enumeration would imply an output-polynomial algorithm for minimal hypergraph transversals, which is a long-standing open problem (see the survey [10] for more information on the problem).

In Section 3.2 we investigate the reason behind the difficulty of listing MISS's, showing that classical techniques for listing algorithms which make use of the so-called *extension problem* are difficult to apply, as the extension problem for MISS's is actually NP-complete. In particular, we prove that deciding whether a partial MISS can be completed into a MISS is NP-complete. Surprisingly, our reduction shows that this result holds even if the number of terminals, i.e. t, is bounded. This means that a general output-polynomial algorithm for MISS's enumeration which works for any number of terminals and which is polynomial both in the size of the graph and the number of terminals should rely on different techniques.

3.1 Listing reduction from Hypergraph Transversal Enumeration

We show that the problem of listing MISS's is at least as hard as a well-known problem, known as HYPERGRAPH TRANSVERSAL ENUMERATION (in short TRANS-ENUM). The latter problem deals with a hypergraph $\mathcal{H} = (X, H)$, where $H \subseteq 2^X$ is the set of hyperedges. A transversal for $\mathcal{H}$ is a hitting set $S \subseteq X$ for the hyperedges in H, namely, $S \cap e \neq \emptyset$ for each hyperedge $e \in H$. A transversal S is (inclusion-wise) minimal if no other $S' \subset S$ is a transversal. The TRANS-ENUM problem asks to list all the minimal transversals for $\mathcal{H}$, and the existence of an output-polynomial algorithm for it is a long-standing open question in the area of enumeration algorithms. Indeed, the latter would provide also the solutions for many enumeration problems arising in several diverse areas such as data-mining, integer programming, biology, databases, game theory, learning theory, and so on (see [10]).

In the following we show how to transform an instance of TRANS-ENUM, i.e. an hypergraph $\mathcal{H}$, to an instance of MISS's enumeration, i.e. a graph $G = (V, E)$ and a set $W \subseteq V$ of terminals, so for each minimal transversal of $\mathcal{H}$ there is a MISS for W in G, and vice versa.

▶ **Lemma 7.** *There is a one-to-one reduction from* TRANS-ENUM *to* MISS*'s enumeration (even in split graphs).*

Proof. We refer to the bipartite formulation of TRANS-ENUM, that is we consider the bipartite graph B whose vertices are partitioned into X and H, and whose edges represent the membership of each vertex of X in the hyperedges of H in which it is involved, and we look at the minimal subsets of X hitting all the vertices of H. We transform B into an instance of MISS's enumeration, i.e. a graph G and a set W, in the following way: choose G as a copy of B, add to G edges between all pairs of vertices of X, i.e., forming a clique, and choose as W the vertices in H. G is by construction a split graph. A minimal hitting set $S \subseteq X$ hits all the vertices in H and is such that there is no $S' \subset S$ doing the same. As $G[S \cup W]$ is connected, while $G[S' \cup W]$ is not, S corresponds to a MISS. Conversely, consider a MISS S in G. Thus $G[S \cup W]$ is connected and each $S' \subset S$ is such that $G[S' \cup W]$ is not. This means that for each vertex in W, i.e., a vertex in H, one of its neighbor is in S, i.e., an element hitting the vertex. Moreover, as S is a clique, each subset S' is still connected, meaning that $G[S' \cup W]$ disconnects some terminal in W, that is S' does not hit some vertex in W. Hence, S corresponds to a minimal hitting set. ◀

3.2 Hardness of the extension problem for miss's

Providing an output-polynomial algorithm for MISS's seems to be a challenging problem, even for a small number of terminals. As the Extension problem is a core block of many classical output-sensitive algorithms such as those based on *binary partition* or *backtracking*, we argue whether we can follow this path: roughly speaking, this is the problem of deciding whether a partial solution, e.g., a subset of nodes, can be completed into a final solution. In this section we show the hardness of the Extension problem for MISS's.

▶ **Problem 1** (Extension problem for MISS's). *Given a graph G, a terminal set W, and $P \subseteq V(G) \setminus W$, is there a minimal induced Steiner subgraph (*MISS*) S such that $P \subseteq S$?*

Note that, if $P = \emptyset$, the answer is trivially always yes, as we can find a MISS by simply setting $S = V(G) \setminus W$, and removing nodes from it until it is possible to do so without disconnecting W. For $t = 1$ the problem is not well defined (or could be considered trivially true if and only if $P = \emptyset$). One may think that the hardness would be influenced by the size of W, or that of P, however, this is not the case.

For $t = 2$ and $|P| = 1$, the smallest non-trivial sizes, the problem is already hard: let $W = \{w_0, w_1\}$ and $P = \{p\}$; as MISS's for $t = 2$ are induced paths, we obtain the problem of finding an induced path between w_0 and w_1 passing through p. Finding such an induced path has been proved to be NP-complete in [6]. This immediately yields the following result.

▶ **Theorem 8.** *The extension problem for* MISS*'s is NP-complete, even for two terminals.*

The difficulty of the extension problem excludes the possibility of applying some well known general techniques, motivating the interest in this combinatorial structure and in ad-hoc techniques for its enumeration.

4 Listing Minimal Steiner Subgraphs with Three Terminals

The problem of listing MISS's seems to be a challenging one as we have shown that the extension problem, which is a core block of many output-sensitive listing algorithms based on the backtracking and binary partition techniques, is NP-complete for MISS's already when the number of terminals is 2 (see Section 3.2). Indeed, in the case of $t = 2$, if the partial solution is just an arbitrary node which is not a terminal, deciding whether there is an induced path involving that node is NP-complete [6]. However, the problem can be solved with polynomial delay [18]: since we are listing induced paths from a source to a target, this limitation can be overcome by suitably defining the partial solution; in this case, a partial solution always includes a terminal, i.e. the source. The same strategy, i.e. forcing each partial solution to include a terminal, clearly does not apply when $t > 2$, as extending that solution will lead again to the hard problem of deciding whether there is an induced path involving that terminal. We show in this section that a workaround is still possible for the special case $t = 3$.

We will characterize some special instances of the extension problem, extending what we will call a *partial Steiner subgraph* (PSS for short), showing that we can indeed find a solution in polynomial time, and that it is possible to obtain a complete binary partition algorithm by just relying on these restricted instances.

To characterize a partial Steiner subgraph, let us define a removable node:

▶ **Definition 9.** *Let S be a Steiner subgraph for $W(S)$, the set of terminals in W that have neighbors in S. Then $v \in S$ is a* removable node *for S if $G[S \cup W(S) \setminus \{v\}]$ is connected.*

Note that if $v \in S$ is a removable node then $W(S) = W(S \setminus \{v\})$ (i.e., $S \setminus \{v\}$ is still a Steiner subgraph for $W(S)$).

As an example, consider Figure 1 (left), with $W = \{w_0, w_1, w_2, w_3\}$: $S = \{2, 3, 4, 6, 7\}$ has no removable node, as $W(S) = W$ and each removal of a node of S would disconnect at least one of the terminals from the others. On the other hand, $S' = \{3, 4, 6, 7\}$ has 4 as removable node, since $W(S) = \{w_1, w_2, w_3\}$.

We can now define a PSS:

▶ **Definition 10** (Partial Steiner Subgraph). *$S \neq \emptyset$ is a partial Steiner subgraph (PSS) if*

- *$W(S) \neq \emptyset$,*
- *S is a Steiner subgraph for $W(S)$,*
- *there is* at most *1 removable node in S.*

Note that a MISS is also a partial Steiner subgraph, as it has no removable nodes. We can now detail the listing procedure in Algorithm 1:

Algorithm 1 List all MISS's of G with $t = 3$.

```
Input  : A graph $G = (V(G), E(G))$ and a terminal set $W \subseteq V(G)$ with $t = |W| = 3$
Output : All MISS's of $G$ and $W$
Call 3-ENUM$(G, W, \emptyset)$
Function 3-ENUM$(G, W, S)$
    if $W(S) = W$ then
        output $S$    // $S \neq \emptyset$
    else
        $X \leftarrow \emptyset$
        foreach $v \in V(G) \setminus (S \cup W) : S \cup \{v\}$ is a PSS do
            if $\exists S' \supseteq (S \cup \{v\}) : S'$ is a MISS in $G$ then
                3-ENUM$(G \setminus X, W, S \cup \{v\})$
            $X \leftarrow X \cup \{v\}$
```

In essence, every recursive node of Algorithm 1 considers a partial Steiner subgraph S: at each step, it identifies all nodes v for which $S \cup \{v\}$ is also a partial Steiner subgraph, and check whether it is possible to eventually extend it into a solution, if so, it recurs adding v to S; afterwards, v is removed from the graph in subsequent recursive calls to avoid duplication (this is modeled by the set X).

To prove that the algorithm is correct, we will use the following fact and lemma.

▶ **Fact 2.** *Let S be a MISS for W with $t = |W| = 3$. Then $G[S \cup W]$ is either (1) a subdivision of the claw, (2) a triangle with 3 pending paths or (3) an induced path.*

Proof. Assume that $G[S \cup W]$ is neither an induced path nor a subdivision of the claw and let P be an induced path in $G[S \cup W]$ from two terminals w_a and w_b. Let P' be the shortest path in $G[S \cup W]$ from w_c to P, and let v be the penultimate vertex of P' not in P. Because $G[P \cup P' \cup W]$ is connected, we can conclude that $S = P \cup P'$, and no vertex of P', but v, is adjacent to a vertex in P. If v is adjacent to two non-adjacent vertices x and x' of P', then any vertex in P between x and x' can be removed without disconnecting the rest, contradicting the minimality of S. Therefore, S is as stated. ◀

▶ **Lemma 11.** *Given a graph G and a set of terminals $W \subseteq V(G)$ with $t = |W| = 3$, let S be a* PSS *and S' a* MISS *such that $S \subset S'$. Then there exists $v \in S' \setminus S$ such that $S \cup \{v\}$ is a* PSS.

Proof. Since $S \cup W(S)$ and $S' \cup W$ are connected subgraphs, and recalling from the preliminaries that no two terminals are adjacent, there must exist at least a node $x \in S' \setminus S$ such that $S \cup \{x\} \cup W(S)$ is connected. Let P be the set of all such nodes, and let us consider the possible cases. To complete the proof, we need to show that for some node $v \in P$ we have that $S \cup \{v\}$ has at most one removable node, meaning it meets all conditions of a PSS. We separately prove 3 comprehensive cases:

- **S has no removable node**. Then it must be that $W(S) = 2$ and S is exactly an induced path between two terminals w_a and w_b.[5] Consider a shortest path from the third terminal w_c to $S \cup \{w_a, w_b\}$ in $G[S' \cup W]$, and let v be the last vertex of the path not belonging to $S \cup \{w_a, w_b\}$. It clearly belongs to P. Consider now $S \cup \{v\}$: either v is connected to w_c, so $S \cup \{v\}$ is a MISS, i.e., $S' = S \cup \{v\}$, or v is removable and $S' \neq S \cup \{v\}$. If v is removable in $S \cup \{v\}$ and makes another vertex x removable in $S \cup \{v\}$, then x is in a path between two non-adjacent neighbors of v in $S \cup \{w_a, w_b\}$. But then $S' \setminus \{x\}$ would be a Steiner subgraph of W, a contradiction. Thus $S \cup \{v\}$ is a PSS.
- $|\mathbf{P}| = \mathbf{1}$. Then all other nodes of S' may reach S only via the single node $v \in P$. This means that if any node of $S \cup \{v\}$ other than v is removable, i.e., does not disconnect v from the rest of S and $W(S)$, then it is also removable in S', contradicting the minimality of S'. Thus the only possible removable node in $S \cup \{v\}$ is v and $S \cup \{v\}$ is a PSS.
- $|\mathbf{P}| \geq \mathbf{2}$ **and S has a removable node** b. We first note that $|W(S)| = 1$. Indeed, if $|W(S)| = 2$, then in any case of Fact 2, a connected subset S connecting two terminals cannot be extended into a connected subset with more than one vertex from $S' \setminus S$. Therefore, we can conclude that S is a path from a vertex b to a terminal w, with b the removable node of S. Because $|W(S)| = 1$ and $b \in S'$, then again by Fact 2 b should have a neighbor v not adjacent to any vertex in $S \setminus \{b\}$. Therefore, $S \cup \{v\}$ is a path included in S', and is then a PSS. ◀

Furthermore, let us prove that the extension problem for PSSs is solvable in polynomial time when $t = 3$.

▶ **Lemma 12.** *Given a graph G, terminal set $W \subseteq V(G)$ with $t = |W| = 3$ and a* PSS *S, the question "is there a* MISS *S' of G such that $S \subseteq S'$?" can be answered in $O(mn)$ time.*

Proof. Let us consider the three possible cases based on $W(S)$:

- $|\mathbf{W(S)}| = \mathbf{1}$. Let $W(S) = \{w_a\}$; as $S \neq \emptyset$, S consists of a path from w_a to the only removable node b of S. Let R be the set of nodes not in S or W that are neighbors of either w_a or some node in $S \setminus \{b\}$, i.e., $R = \{v \in V(G) \setminus (S \cup W) : N(v) \cap (W(S) \cup S \setminus \{b\}) \neq \emptyset\}$. Note that connecting terminals via nodes of R makes b redundant, as such nodes can reach w_a without using b; thus we must connect at least one terminal to w_a without using nodes of R, i.e., we may use at most one node of R. Consider $G \setminus R$: note that b is the only node in $S \cup \{w_a\}$ with neighbors outside the solution. How many terminals distinct from w_a are in the same connected component of $G \setminus R$ as b? If *zero*, we can answer **no**, as we will need to connect the remaining two terminals via R, making b redundant (thus the result not minimal). If *two*, we can answer **yes**, as the three terminals are in the same connected component, and any MISS of $G \setminus R$ will fully contain S. Finally, if *one*,

[5] If $W(S) = 1$, S is an induced path from a terminal, and the node opposite to it is always removable.

let us call it w_b and let C_1 be the connected component of $G \setminus R$ containing w_a, b and w_b, and C_2 the one containing the final terminal w_c; recall that we may use one node $c \in R$, and this node must connect S to C_2. We try all possible choices for c, noting that c cannot make any node of S removable other than b (otherwise it would remain removable in S'), and when testing c we must remove all its neighbors in $C_1 \setminus S \cup W$ from the graph (otherwise we may use them to bypass b, making b redundant). If any such suitable c exists, we get a MISS S' by taking any induced path from c to w_c in C_2 and any induced path from b to w_b in $C_1 \setminus (N(c) \setminus (S \cup W))$, and adding them to S, and thus we can answer **yes**. Otherwise, there is no other possibility of minimally connecting w_b and w_c to w_a without making b or some other node redundant, so we answer **no**.

- $|\mathbf{W(S)}| = \mathbf{2}$. Let $W(S) = \{w_a, w_b\}$. If there is a removable node b in S, define as before $R = \{v \in V(G) \setminus (S \cup W) : N(v) \cap (W(S) \cup S \setminus \{b\}) \neq \emptyset\}$. In this case, we may not use any node of R to connect the remaining terminal w_c, as this would allow us to bypass b, making it redundant. Thus if w_c is in the same connected component as b in $G \setminus R$, we can answer **yes**, as any induced path between b and w_c in this graph yields a MISS when added to S. Otherwise, there is no connection without using some node of R and making b redundant, so we can answer **no**. Otherwise, if there is no removable node in S, define $R' = \{v \in V(G) \setminus (S \cup W) : N(v) \cap (W(S) \cup S) \neq \emptyset\}$: clearly any MISS extending S must contain at least one (actually, exactly one) node from R', and for any $v \in R'$, v is removable in $S \cup \{v\}$, so we can test all possible nodes in R' as above, since $S \cup \{v\}$ has a removable node (trivially accounting for the special cases where v directly connects to the third terminal resulting directly in a MISS, or where v makes some other node in S removable and thus cannot be added to S).
- $|\mathbf{W(S)}| = \mathbf{3}$. Then either S is a MISS and the answer is **yes**, or it has removable nodes and the answer is **no**.

As for the time complexity, removing a set of nodes and computing the connected components of a graph can be done in $O(m)$ time. The cost is dominated by cases $|\mathbf{W(S)}| = \mathbf{1}$ and $|\mathbf{W(S)}| = \mathbf{2}$, where we need to perform such a test for each node in R or R', respectively. As these sets have size $O(n)$ we can answer the question in $O(mn)$ time. ◀

We can finally claim correctness and complexity of the algorithm:

▶ **Theorem 13.** *Given G and a set of terminals $W \subseteq V(G)$ with $t = |W| = 3$, Algorithm 1 lists all MISS's of G for W in $O(mn^3)$ time per solution.*

Proof. Firstly, the initial call with $S = \emptyset$ clearly does not trigger the output in Line 4 as $W(S) = \emptyset$. On all other recursive calls, S is a PSS such that there exist some MISS S' including S (due to Line 8 in the parent call). Whenever $W(S) = W$, it follows that $S' = S$ because S' is minimal and cannot contain other (redundant) nodes. So the algorithm only outputs MISS's.

Furthermore, let S^* be an arbitrary MISS, we show that it is found: let X be any recursion node of Algorithm 1 such that no node of S^* has been removed from G, and the current S, called here S_X, is fully contained in S^* (X exists as both conditions are met in the beginning, when no node has been removed from G and $S = \emptyset$). Let x be the first node of $S^* \setminus S_X$ considered on Line 7 such that $S_X \cup \{x\}$ is a PSS (x exists by Lemma 11): when x is considered, clearly no node of S^* has yet been removed from G; furthermore, the condition in Line 8 is true by the existence of S^*, so a child recursive call is generated with $S_Y = S_X \cup \{x\}$ and G still fully containing S^*. By induction, the algorithm will, in some path of the recursion tree, at each step add one more node of S^* without removing any from G, until eventually S^* is found.

Finally, duplication is impossible by the binary partition argument, as all solutions generated from a recursive call on Line 9 will contain v, and all subsequent recursive calls will remove v from G (see Line 10), thus will find different solutions.

As for the cost per solution, since all leaves of the recursion tree output solutions, it is bound by the cost of the recursion nodes of a root-to-leaf path in the recursion tree, which are $O(n)$ as each node adds a node to S when recurring. In each node, we must identify all v such that $S \cup \{v\}$ is a PSS: some trivial case analysis (using Fact 2) can reveal that this is true only when v is a neighbor only of the removable node of S, or when S has no removable node and v has at most two neighbors in S, adjacent to each other, thus this takes $O(|N(v)|)$, for a total of $O(\sum_{v \in V(G)} |N(v)|) = O(m)$ time. Furthermore, the test in Line 8 takes $O(mn)$ time by Lemma 12 and is performed $O(n)$ times, for a total cost per recursion node of (mn^2). The cost per solution of Algorithm 1 is thus $O(mn^3)$. ◀

5 Conclusions

We considered minimal Steiner induced subgraphs, a natural variant of Steiner trees, shifting our focus on a node-centric view. We studied these combinatorial objects providing an output-sensitive algorithm for minimum MISS's. On the other hand, getting all MISS's with an output-sensitive algorithm seems to be more challenging for unbounded number of terminals, as we have shown that this problem is actually harder than listing all the minimal hypergraph transversals. Moreover, the fact that the extension problem is actually NP-complete, even with bounded number of terminals, makes us thinking that an *ad hoc* strategy should be thought for different sizes of W. For $t = 2$, the problem can be solved by known listing algorithm for induced paths. In this paper we have provided an output-sensitive algorithm for the case $t = 3$.

As a final remark, we observe that for many problems, like, for instance, listing maximum size and maximal independent sets [5], listing minimum/maximum solutions is harder than listing minimal/maximal solutions. In contrast, in the case of MISS's with a bounded number of terminals, listing minimum size MISS's seems to be easier than getting all MISS's, as we could efficiently solve the former case for $t = O(1)$, while the latter only for $t \leq 3$.

References

1 A. Aazami, J. Cheriyan, and K. R. Jampani. Approximation Algorithms and Hardness Results for Packing Element-Disjoint Steiner Trees in Planar Graphs. *Algorithmica*, 63(1–2):425–456, June 2012.

2 Lijun Chang, Xuemin Lin, Lu Qin, Jeffrey Xu Yu, and Wenjie Zhang. Index-based Optimal Algorithms for Computing Steiner Components with Maximum Connectivity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 459–474, New York, NY, USA, 2015. ACM. doi:10.1145/2723372.2746486.

3 Chandra Chekuri, Alina Ene, and Ali Vakilian. Node-Weighted Network Design in Planar and Minor-Closed Families of Graphs. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2012.

4 Joseph Cheriyan and Mohammad R. Salavatipour. Packing element-disjoint Steiner trees. *ACM Transactions on Algorithms*, 3(4):47:1–47:10, November 2007. doi:10.1145/1290672.1290684.

5 Alessio Conte, Roberto Grossi, Andrea Marino, Takeaki Uno, and Luca Versari. Listing Maximal Independent Sets with Minimal Space and Bounded Delay. In *String Processing and*

Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26-29, 2017, Proceedings, pages 144–160, 2017. doi:10.1007/978-3-319-67428-5_13.

6 Nicolas Derhy and Christophe Picouleau. Finding induced trees. *Discrete Applied Mathematics*, 157(17):3552–3557, 2009.

7 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

8 DIMACS. 11th DIMACS Implementation Challenge. http://dimacs11.zib.de/, 2014 (page accessed April 2019).

9 Mitre C. Dourado, Rodolfo A. Oliveira, and Fábio Protti. Algorithmic aspects of Steiner convexity and enumeration of Steiner trees. *Annals of Operations Research*, 223(1):155–171, December 2014. doi:10.1007/s10479-014-1607-5.

10 Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational Aspects of Monotone Dualization: A Brief Survey. *Discrete Appl. Math.*, 156(11):2035–2049, June 2008.

11 Sudipto Guha and Samir Khuller. Improved Methods for Approximating Node Weighted Steiner Trees and Connected Dominating Sets. *Information and Computation*, 150(1):57–74, 1999. doi:10.1006/inco.1998.2754.

12 Mathias Hauptmann and Marek Karpinski. A Compendium on Steiner Tree Problems. http://theory.cs.uni-bonn.de/info5/steinerkompendium/, Accessed February 2018.

13 Daiki Hoshika and Eiji Miyano. Approximation Algorithms for Packing Element-Disjoint Steiner Trees on Bounded Terminal Nodes. *IEICE Transactions*, 99-A(6):1059–1066, 2016. URL: http://search.ieice.org/bin/summary.php?id=e99-a_6_1059.

14 Jiafeng Hu, Xiaowei Wu, Reynold Cheng, Siqiang Luo, and Yixiang Fang. On Minimal Steiner Maximum-Connected Subgraph Queries. *IEEE Trans. Knowl. Data Eng*, 29(11):2455–2469, 2017. doi:10.1109/TKDE.2017.2730873.

15 L. Khachiyan, E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. On the Complexity of Some Enumeration Problems for Matroids. *SIAM Journal on Discrete Mathematics*, 19(4):966–984, January 2005. doi:10.1137/S0895480103428338.

16 PACE. The parametrized Algorithms and Computational Experiments Challenge. https://pacechallenge.wordpress.com/pace-2018/, 2018.

17 Jan Arne Telle and Yngve Villanger. Connecting Terminals and 2-Disjoint Connected Subgraphs. In *Graph-Theoretic Concepts in Computer Science*, pages 418–428. Springer Berlin Heidelberg, 2013.

18 Takeaki Uno. An Output Linear Time Algorithm for Enumerating Chordless Cycles. *IPSJ SIG Notes*, 2003(110):47–53, November 2003.

Enumeration of Preferred Extensions in Almost Oriented Digraphs

Serge Gaspers

UNSW Sydney, Sydney, Australia
Data61, CSIRO, Australia
sergeg@cse.unsw.edu.au

Ray Li

UNSW Sydney, Sydney, Australia
rayli.main@gmail.com

Abstract

In this paper, we present enumeration algorithms to list all preferred extensions of an argumentation framework. This task is equivalent to enumerating all maximal semikernels of a directed graph. For directed graphs on n vertices, all preferred extensions can be enumerated in $O^*(3^{n/3})$ time and there are directed graphs with $\Omega(3^{n/3})$ preferred extensions. We give faster enumeration algorithms for directed graphs with at most $0.8004 \cdot n$ vertices occurring in 2-cycles. In particular, for oriented graphs (digraphs with no 2-cycles) one of our algorithms runs in time $O(1.2321^n)$, and we show that there are oriented graphs with $\Omega(3^{n/6}) > \Omega(1.2009^n)$ preferred extensions.

A combination of three algorithms leads to the fastest enumeration times for various proportions of the number of vertices in 2-cycles. The most innovative one is a new 2-stage sampling algorithm, combined with a new parameterized enumeration algorithm, analyzed with a combination of the recent monotone local search technique (STOC 2016) and an extension thereof (ICALP 2017).

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning; Theory of computation → Fixed parameter tractability; Mathematics of computing → Enumeration

Keywords and phrases abstract argumentation, exact algorithms, exponential time algorithms, parameterized algorithms, enumeration algorithms, semikernels in digraphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.74

Related Version https://arxiv.org/abs/1907.01006

Acknowledgements We thank Oliver Fisher for fruitful discussions and collaboration on preliminary results in the early stages of this work.

1 Introduction

In Dung's theory of abstract argumentation [15], an *argumentation framework* (AF) is a digraph $G = (V, E)$, where each vertex represents an argument, and an arc $(u, v) \in E$ denotes that argument u *attacks* argument v. There are various semantics that express what properties a set of arguments should have for a rational agent to stand by that set of arguments. One of the most central semantics is the *preferred semantics* that was already proposed by Dung in his foundational paper [15]. Let $S \subseteq V$ be a subset of vertices (also called *extension*) of a digraph $G = (V, E)$. The set S is *conflict-free* if no arc has both endpoints in S. A vertex $v \in V$ is *acceptable* with respect to S if for each arc $(u, v) \in E$ there is an arc $(w, u) \in E$ with $w \in S$. In other words, for each argument u that attacks v, there is an argument w in S that attacks u. We say in this case that w *defends* v against u. The set S is *admissible* if it is conflict-free and each argument in S is acceptable with respect to S. The set S is *preferred* if it is an inclusion-wise maximal admissible set.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 74; pp. 74:1–74:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

While we will use the language of abstract argumentation, we remark that such vertex sets have also been studied in graph theory. Neumann-Lara [27] (see also [21]) defined the notion of *semikernels*. Maximal *semikernels* are equal to the preferred extensions in the directed graph where all arcs are reversed. The related notion of *kernels* [32] has the same correspondence with stable extensions in abstract argumentation, and was introduced as an abstract solution concept in cooperative game theory, but has been extensively studied in the theory of directed graphs. In particular, various issues around the enumeration of kernels and semikernels have been considered in previous work [2, 5, 20, 29].

Motivation. A central problem in abstract argumentation is the enumeration of extensions prescribed by a given semantics. The enumeration of preferred extensions is of particular interest, firstly for its own sake, but also in the study of other semantics as it forms the basis of several other semantics refining this set. A number of existing algorithms and implementations enumerate all preferred extensions of a digraph (see, e.g., [6, 7, 9, 10, 11, 12, 14, 25, 28, 30, 31]). The enumeration of preferred extensions is also a part of the biennial International Competition on Computational Models of Argumentation (ICCMA). Computational problems where the enumeration of extensions are used involve answering the questions: is a given argument in some / all preferred extensions and what is the number of preferred extensions containing a given argument / in total. Upper bounds on the number of extensions under various semantics have also been proposed as fundamental characteristics to compare various semantics in abstract argumentation [3, 16].

We study the enumeration of preferred extensions in digraphs with no, or relatively few, 2-cycles (i.e., bidirectional arcs). Our aim is to determine how much the presence of 2-cycles affects the number of preferred extensions of an AF. Mutually attacking arguments play a special role in abstract argumentation [24], but this conflict is often resolved rather easily if the strength of the two attacks can be evaluated [4], or the user's preference between the two arguments can be elicited [26, 1]. These methods of resolving conflicts motivate the study of problems, and in particular enumeration problems, for AFs with no or few 2-cycles.

Our results. Define the *resolution order* of a digraph $G = (V, E)$, denoted $r(G)$, as the number of vertices that belong to a 2-cycle in G. We study enumeration algorithms and combinatorial upper bounds on the number of preferred extensions in *oriented graphs* (digraphs without 2-cycles) and digraphs that have small resolution order.

Our main result is an algorithm that, for any $\varepsilon > 0$, enumerates all preferred extensions of a digraph G on n vertices in time

$$O^*\left(\left(\min\left(\varphi^{2r}\cdot\varphi^{1-r},\ \left(\left(1+2^{\frac{1}{4}}-\frac{1}{\sqrt{2}}\right)^{r}\cdot\left(2-\frac{1}{\sqrt{2}}\right)^{1-r}\right)^{1+\varepsilon},\ 3^{1/3}\right)\right)^{n}\right)$$

$$\le O^*\left(\left(\min\left(1.5180^{r}\cdot 1.2321^{1-r},\ 1.4822^{r}\cdot 1.2929^{1-r},\ 1.4423\right)\right)^{n}\right),$$

where $r = r(G)/n$, and $\varphi \approx 1.2321$ is the positive root of $1 - x^{-1} - x^{-8}$. The O^* notation hides factors that are polynomial in the input size. See Figure 1, which plots the base α of the running time expressed as $O^*(\alpha^n)$ for r varying from 0 to 1. For $r = 1$, this is best possible and follows from the work in [16, 26]. At the other end of the spectrum, i.e., for oriented graphs where $r = 0$, the upper bound is $O^*(\varphi^n) \le O(1.2321^n)$ and is obtained via a carefully constructed branching algorithm and running time analysis. We also give a lower bound on the largest number of preferred extensions an oriented graph on n vertices may have of $\Omega(3^{n/6}) \ge \Omega(1.2009^n)$. A construction, which we call the *Oriented Translation*, reducing an arbitrary digraph $G = (V, E)$ to an oriented graph with $|V| + r(G)$ vertices,

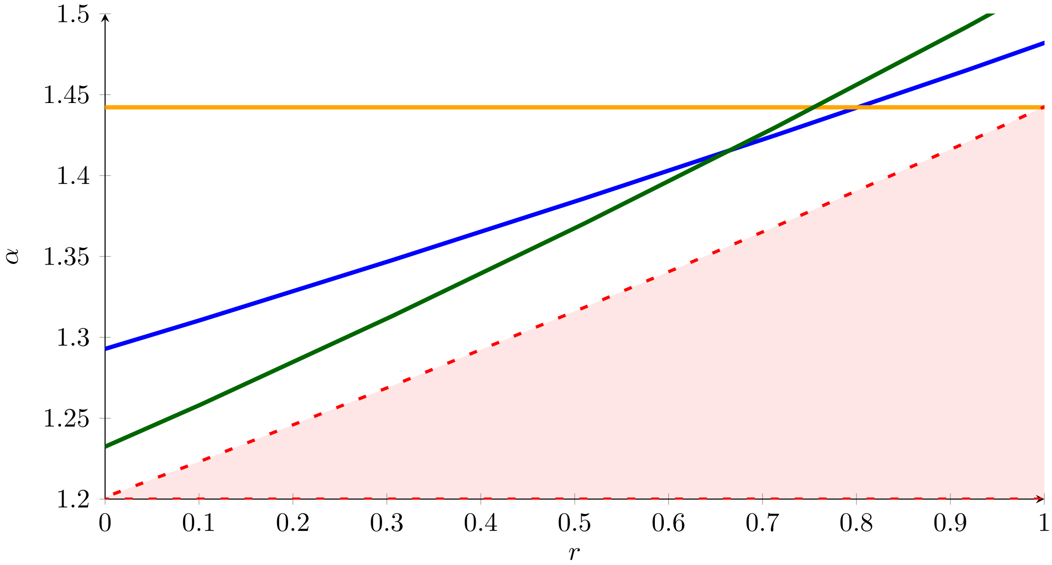


Figure 1 The graph depicts the base of the exponential running times $O^*(\alpha^n)$ of the three enumeration algorithms, according to $r = r(G)/n$. When $r < 0.6684$, our branching algorithm for oriented graphs together with the Oriented Translation (dark green) gives the fastest algorithm. For $r > 0.8005$, the algorithm based on previous work [16, 26] (orange) is fastest. In the middle range, the combination of the 2-phase monotone local search with the parameterized enumeration algorithm (blue) is fastest. Our lower bound on the largest number of preferred extensions is drawn with a dashed red line.

such that there is a bijection between their preferred extensions, allows us to generalize these upper and lower bounds to $O^*\left(\varphi^{2\cdot r(G)} \cdot \varphi^{n-r(G)}\right) \leq O(1.5180^{r(G)} \cdot 1.2321^{n-r(G)})$ and $\Omega(3^{r(G)/3} \cdot 3^{(n-r(G))/6}) \geq \Omega(1.4422^{r(G)} \cdot 1.2009^{n-r(G)})$, respectively.

Our main technical contribution is the third algorithm. It relies on a parameterized enumeration algorithm and extensions of the recent monotone local search framework [17]. The parameterized enumeration algorithm has as input a digraph $G = (V, E)$, a set of arguments S, and a non-negative integer k, and it enumerates all maximal admissible extensions $T \subseteq S$ of G within distance k of S. Its running time can be upper bounded by $O^*(2^{k/2+r(G[S])/4})$. This is optimal, since there are instances for which the solution consists of $\Omega(2^{k/2+r(G[S])/4})$ preferred extensions at distance at most k from S. Furthermore, under the Strong Exponential Time Hypothesis, the corresponding decision problem has no $O^*(2^{(1-\varepsilon)(k/2+r(G[S])/4)})$ time solution for any $\varepsilon > 0$. We use this parameterized enumeration algorithm in a new 2-phase monotone local search procedure, where we separately sample vertices from B, the set of vertices in at least one 2-cycle, and $V \setminus B$ and then apply the parameterized enumeration algorithm. The running time analysis is a new combination of the results in [17] for the first sampling phase and [23] for the second sampling phase, combined with the parameterized subroutine. From a technical point of view, this is the most innovative part of this paper. (From a conceptual point of view, the most innovative contribution is probably the synergy between modern enumeration algorithmics and the theory of abstract argumentation.) This results in an algorithm enumerating all preferred extensions of a given digraph G in time $O^*\left(\left(1+2^{1/4}-2^{-1/2}\right)^{(1+\varepsilon)\cdot r(G)} \cdot \left(2-2^{-1/2}\right)^{(1+\varepsilon)\cdot(n-r(G))}\right)$.

Interpretation of results. Figure 1 indicates we have narrowed the gap between best known lower and upper bounds for a wide range of r and improved algorithms and combinatorial upper bounds whenever $r \leq 0.8004$. The result for $r = 0$ shows that for oriented graphs, our new algorithm allows to handle instances with 75% more arguments, compared with the previous best $O(3^{n/3})$ upper bound. (We have that $\log_{1.2321}(3^{1/3}) \approx 1.7545$.)

Outline. Sections 3 and 4 describe our monotone local search algorithm. Section 5 describes our branching algorithm for oriented graphs.

First we introduce the parameterized enumeration problem that will form the subroutine of our Monotone Local Search.

Maximal Admissible Subset Enumeration (MASE)

Input:	Graph G, set $S \subseteq V(G)$, integer k
Parameter:	k
Output:	Enumerate all maximal admissible sets $T \subseteq S$ such that $\lvert S \setminus T\rvert \leq k$.

There is a subtlety here with how we define maximal. We say T is a maximal admissible subset of S if there does not exist an admissible set U such that $T \subsetneq U \subseteq S$. Notably T is not necessarily a preferred extension (though T is if $S = V(G)$).

In Section 3, we present an algorithm for MASE, parameterized by k and $r(G[S])$, the resolution order of the subgraph of G induced by S.

▶ **Theorem 1.** *For an instance $I = (G, S, k)$ of MASE, let $\mu(I) := \frac{k}{2} + \frac{r(G[S])}{4}$. Then MASE can be solved in $O^*(2^{\mu(I)})$ time. Furthermore, there are at most $2^{\mu(I)}$ maximal admissible subsets of S within distance k of S. Hence, there are at most $2^{\mu(I)}$ preferred extensions that are subsets of S with size $\geq |S| - k$.*

Our algorithm is a standard parameterized branching algorithm. Compared to the enumeration of independent sets, the primary additional tool we have is a powerful simplification rule, **(Undefendable)**, for vertices with in-degree 0.

In Section 4, we extend our parameterized algorithm into a general enumeration algorithm through a novel 2-phase application of Monotone Local Search. Since 2-cycles increase the run time of our MASE subroutine, we modify Monotone Local Search to sample separately between a set of "bad vertices" (ones contained in a 2-cycle) and "good vertices" (ones not contained in any 2-cycle). This presents a speed up compared to a more direct application of the Monotone Local Search framework. We believe this may be useful for other problems.

Separately, Section 5 presents a branching algorithm for oriented graphs. Again, **(Undefendable)** plays a critical role. By carefully setting up the state we also have a simplification rule for vertices with out-degree 0. We tailor our branching rules to take full advantage of these two simplification rules. This is combined with a lot of careful case analysis and ad-hoc methods (including a graph classification theorem for Case 4).

Each of our algorithms also provide a corresponding combinatorial upper bound on the number of preferred extensions. The various enumeration algorithms and bounds are collected in Section 6. To extend the results from Section 5 to general graphs we require the following result which we call the *Oriented Translation.*

▶ **Theorem 2.** *There is a linear time algorithm that transforms any AF G into an oriented AF G' with $|V(G')| = |V(G)| + r(G) + 3$ such that there is a bijection between the preferred extensions of G and the preferred extensions of G' that can be applied in linear time.*

The basic idea of the construction is carefully converting 2-cycles into 4-cycles by duplicating vertices contained in at least one 2-cycle. The construction can be found in the full version.

In the full version we include all omitted proofs (including, in particular, the case analysis of our branching algorithms) and a few extra results:

We apply Theorem 2 to derive complexity results on oriented graphs by extending constructions for directed graphs. We show that:

- Unless P=NP, no algorithm enumerates the admissible or preferred extensions of an AF in output-polynomial time, even when the AF is an oriented graph.
- Assuming the Strong Exponential Time Hypothesis, our algorithm for the decision variant of MASE is optimal.
- The enumeration bound in Theorem 1 is tight (i.e: there exist instances with $2^{\mu(I)}$ maximal admissible subsets within distance k).

We also briefly justify the choice of the measure μ we use for our MASE algorithm by comparing it against similar measures.

Notation. An *oriented graph* is a digraph with no 2-cycles.

We will use the notation $N(v)$ to denote the set of vertices adjacent to v and $N[v]$ to denote $N(v) \cup \{v\}$.

We will assume throughout that AFs have no self loops. In the full version we present the Loopless Translation which, with a (small) constant overhead, transforms any AF into a loopless AF with the same lattice of admissible extensions (under inclusion).

2 Background on Analysis of Branching Algorithms

All results here can be found in standard textbooks. We mostly follow Fomin & Kratsch [19].

To analyze our branching algorithms we use Measure & Conquer [18]. A *measure* is a function assigning a non-negative number to each instance I.

We define the following standard terminology (e.g: see [19, 13]).

▶ **Definition 3** (Branching Vector and Number [19]). *Let μ be a measure. Let b be a branching rule that for any input instance, say I, branches into r instances with measures $\mu(I) - t_1, \mu(I) - t_2, \ldots, \mu(I) - t_r$ such that for all i, $t_i > 0$. Then we call $\mathbf{b} = (t_1, t_2, \ldots, t_r)$ the* branching vector *of branching rule b.*

Let α be the unique positive real root of $x^n - x^{n-t_1} - x^{n-t_2} - \ldots - x^{n-t_r}$.

We call α the branching number *of the branching vector $\mathbf{b}$.*

The following lemma from [22] forms the basis for the analysis of our branching algorithms.

▶ **Lemma 4** (Combine Analysis Lemma [22]). *Let A be an algorithm for a problem which for each instance, say I, either directly solves the instance in $O^*(\alpha_0^{\mu(I)})$ time or after polynomial time, applies one of r branching rules b_i, the i-th of which has branching number α_i. Then A has running time $O^*(\alpha^{\mu(I)})$ where $\alpha = \max_{0 \le i \le r}(\alpha_i)$.*

The following lemma forms the basis of our enumeration bounds. The search tree of a branching algorithm is the tree formed by the recursive calls, with the leaves being cases that can be solved directly. In all our applications the number of leaves is an upper bound on the number of objects being enumerated.

▶ **Lemma 5** (Combine Analysis Lemma for Enumeration). *Let A be an algorithm for a problem which for each instance, say I, either directly solves the instance with a branching algorithm that generates at most $\alpha_0^{\mu(I)}$ leaves or applies one of r branching rules b_i, the i-th of which has branching number α_i. Then the search tree for applying A to I has at most $\alpha^{\mu(I)}$ leaves where $\alpha = \max_i(\alpha_i)$.*

3 Parameterized Enumeration Problems

Our algorithm for MASE follows a standard template for parameterized branching algorithms (see Section 2 for notation). We will consider the measure $\mu(I) = \frac{k}{2} + \frac{b}{4}$ where k is the number of vertices we are allowed to remove and $b := r(G[S])$ is the resolution order of $G[S]$. We will design a branching algorithm with run time $O^*(2^{\mu(I)})$ which recurses into subinstances and collates their results to obtain the maximal admissible subsets of S.

However, there is a technical difficulty in the collation step arising from the fact that a maximal admissible subset of $S' \subsetneq S$ may not be a maximal admissible subset of S. This gives rise to the following subproblem:

Maximal Subset Collation

Input:	Graph G, c pairs (S_i, C_i) where for each i, $S_i \subseteq V(G)$ and C_i is a set containing only maximal admissible subsets of S_i
Output:	A set containing all maximal elements of $\bigcup_{i=1}^{c} C_i$

The main result we need is:

▶ **Lemma 6.** *Maximal Subset Collation can be solved in* $O(c\sum_{i=1}^{c}|C_i| \cdot \mathrm{poly}(|V|))$ *time.*

An algorithm for Maximal Subset Collation can be found in the full version. It is a consequence of Lemma 7.

3.1 An $O^*(2^{\mu(I)})$ algorithm for Maximal Admissible Subset Enumeration

The overall structure of our algorithm is described in Algorithm 1.

The following table lists the branching rules in application order with the first row being the base case. The second column describes the instances each rule is applicable to. After applying simplification rules, our branching algorithm will always apply the first rule that is applicable to the input instance.

Case	Requirement to apply	Running Time
Base	S is conflict-free.	Solves in $O^*(1)$ time, returns ≤ 1 set.
1	$G[S]$ is oriented with maximum total degree ≤ 2.	Branching vector $(1,1)$, branching number 2.
2	There is a 2-cycle in $G[S]$.	Branching vector $(1,1)$, branching number 2.
3	$G[S]$ has maximum total degree ≥ 4.	Branching vector $(2, \frac{1}{2})$, branching number ≈ 1.91.
4	$G[S]$ contains a vertex with total degree 3.	Branching vector $(1, \frac{3}{2})$, branching number ≈ 1.76.

We note that these requirements are exhaustive, hence there will always be at least one applicable rule in any instance.

The base case follows trivially from the following more general result:

▶ **Lemma 7.** *Suppose $S \subseteq V(G)$ induces a DAG in G. Then S has exactly one maximal admissible subset and it can be found in polynomial time.*

The main tool for this is the following Simplification Rule which we will use repeatedly:

▶ **Simplification Rule 1 (Undefendable).** *Let $u \in S$ be a vertex such that there exists a vertex $a \in V(G)$, a attacks u and no vertex in S attacks a. Then there is no admissible subset of S that contains u so we can safely set $S \leftarrow S \setminus \{u\}$.*

Algorithm 1 Structure of MASE branching algorithm.

```
Ensure: Returns all maximal admissible subsets $T \subseteq S$ such that $|S \setminus T| \leq k$
  function MASE($S, k$)
    if $k < 0$ then
      return $\emptyset$
    while (Undefendable) applies do
      Apply (Undefendable)
    if Base Case applies then
      Solve the instance directly through the base case subroutine.
    else
      Let $b_i$ be the first branching rule that applies.
      Let $(S_1, k_1) \ldots (S_r, k_r)$ be the subinstances obtained from applying $b_i$.
      Let $C_i = \text{MASE}(S_i, k_i)$, for all $1 \leq i \leq r$.
      return Maximal_Subset_Collation($(S_1, C_1), \ldots, (S_r, C_r)$)
```

Lemma 7 follows from applying **(Undefendable)** to S until it is no longer applicable. Then S is acceptable with respect to S (all vertices in V attacking S are also attacked by S) and conflict-free (else **(Undefendable)** would be applicable to any vertex attacked by a maximal vertex in $G[S]$). Hence, by definition, S is admissible. It is the unique maximal admissible subset as **(Undefendable)** only removes vertices in no admissible subsets of S.

The above results are essentially all well known (see [14, 8] for earlier applications of **(Undefendable)**). The full version contains the above argument in more detail.

Proofs of the branching cases can be found in the full version. A few short remarks:

Case 1 follows from noting $G[S]$ must be a family of cycles.

Cases 2 and 3 follow from picking a specific vertex v then applying a 2-way branch, in one branch enumerating all maximal admissible subsets that include v, in the other branch enumerating all that do not include v. In case 2 we pick the vertex in the 2-cycle with higher total degree. In case 3 we pick any vertex with total degree at least 4.

Case 4 is probably the most instructive. It best showcases the power of **(Undefendable)**. First we show a vertex v exists with in-degree 1, out-degree 2. Then a 2-way branch is applied to the vertex attacking v, using **(Undefendable)** to improve the branch where the vertex is excluded and v is left with in-degree 0.

3.2 Running Time Analysis

The base case is solvable in polynomial time. Each of our branching rules has branching number ≤ 2. Hence, if we ignore the calls to Maximal Subset Collation, by the Combine Analysis Lemma our algorithm has running time $O^*(2^{\mu(I)})$.

Maximal Subset Collation is applied to each admissible subset encountered by the MASE algorithm (i.e: each leaf in the search tree) at most d times, where d is the maximum depth of the search tree. By Lemma 6 each application incurs a $O(\text{poly}(|V|))$ cost ($c \leq 2$ for our branching rules). Hence the overall cost incurred by the Maximal Subset Collation step is

$$O(A \cdot d \cdot \text{poly}(|V|))$$

where A is the total number of admissible subsets encountered by the MASE algorithm (equivalently, the number of leaves in the search tree). By the Combine Analysis Lemma for Enumeration, A is $O(2^{\mu(I)})$. The maximum depth d is $O(\mu(I))$ which we may take to be $O(|V|)$. Hence Maximal Subset Collation incurs an overhead cost of $O^*(2^{\mu(I)})$.

As noted, our algorithm also satisfies the requirements for applying the Combine Analysis Lemma for Enumeration. We summarize all the above results in the following theorem.

▶ **Theorem 8.** *Let $\mu(I) = \frac{k}{2} + \frac{b}{4}$, where b is the resolution order of $G[S]$. Then MASE can be solved in $O^*(2^{\mu(I)})$ time. Furthermore, there are at most $2^{\mu(I)}$ maximal admissible subsets of S within distance k of S. Hence, there are at most $2^{\mu(I)}$ preferred extensions that are subsets of S with size $\geq |S| - k$.*

4 Monotone Local Search

In this section, we apply Monotone Local Search to our $O^*(2^{\mu(I)})$ algorithm for Maximal Admissible Subset Enumeration. A basic exposition of Monotone Local Search will be provided here, see [17] for additional background. We adopt the notation of [17].

The framework normally applies to extension problems. However we can just as easily apply it to removal problems (formally by focusing on the complement of each set, we can turn any removal problem into an extension problem). Hence, we will freely use the framework with removal problems instead.

For our application, the instance I is the graph G and the family we are looking to enumerate, $\mathcal{F}_I$, is the set of all preferred extensions of G. We will apply Monotone Local Search using MASE as our subroutine.

For a MASE instance $I' = (G, X, k)$, let $\mathcal{F}^k_{I,X}$ denote the set of all maximal admissible sets $T \subseteq X$ such that $|X \setminus T| \leq k$. Hence $\mathcal{F}^k_{I,X}$ contains all preferred extensions that are subsets of X within distance k, but may also contain admissible extensions that are not preferred extensions.

Because of this, our Monotone Local Search will enumerate F_I, however, it may also enumerate some non-maximal admissible extensions. Our result, Theorem 10, will account for this, however, for simplicity we will just speak of enumerating F_I throughout this section.

A naive application of the Monotone Local Search framework with our $O^*\left(2^{\frac{k}{2}+\frac{b}{4}}\right)$ algorithm for enumerating $\mathcal{F}^k_{I,X}$ yields an $O^*\left(2^{\frac{b}{4}}\left(2 - \frac{1}{\sqrt{2}}\right)^{n+o(n)}\right) \approx O^*(1.1893^b \cdot 1.2929^n)$ time algorithm that enumerates all preferred extensions of G. To improve this we need a basic understanding of how Monotone Local Search works.

4.1 Basic Overview of Monotone Local Search

We need the following definition from [17] (slightly modified to account for our preference for removal problems):

▶ **Definition 9** ([17]). *Let U be a universe of size n and let $0 \leq p \leq q \leq n$. A family $C \subseteq \binom{U}{q}$ is an (n, p, q)-set-containing-family if for every set $S \in \binom{U}{p}$, there exists a $Y \in C$ such that $S \subseteq Y$.*

For any fixed s, a value $t \geq s$ will somehow be chosen. Then, a (n, s, t)-set-containing-family C_s is constructed. For each set in C_s, its subsets from $\mathcal{F}_I$ obtained by removing at most $t - s$ elements are then enumerated using an enumeration subroutine. This enumerates all elements of $\mathcal{F}_I$ with size s. Supposing the subroutine has run time $O^*(\alpha^k)$ (where k is the parameter), this step of Monotone Local Search has run time $O^*(|C_s| \cdot \alpha^{t-s})$.

Repeating this for all s, Monotone Local Search has running time $O^*(\max_{1\leq s\leq n} |C_s| \cdot \alpha^{t-s})$. With the right choices of t and C_s, [17] shows the running time $O^*((2 - \frac{1}{\alpha})^{n+o(n)})$.

Algorithm 2 Structure of Improved Monotone Local Search algorithm.

```
Ensure: Returns a set containing all preferred extensions of $G$ (and possibly some non-preferred extensions)
  function ImprovedMLS($G$)
      Let $\mathcal{F}_I = \emptyset$.
      for $b = 1$ to $|B|$ do
          Let $b' = \text{determine\_b}'(b)$.
          Let $C_b$ be a $(|B|, b, b')$-set-containing-family.
          for $d = 1$ to $|D|$ do
              Let $d' = \text{determine\_d}'(d)$.
              Let $C_d$ be a $(|D|, d, d')$-set-containing-family.
              for all pairs $(S, T)$ with $S \in C_b, T \in C_d$ do
                  Let $\mathcal{F}_I = \mathcal{F}_I \cup \text{MASE}(S \cup T, (b' - b) + (d' - d))$.
      return $\mathcal{F}_I$
```

4.2 Improving our Monotone Local Search

We start with some notation. For any digraph $G = (V, E)$, let B be the set of vertices in V in at least one 2-cycle and let $D := V \setminus B$. The key idea is we will sample vertices from B and D separately. The overall structure is described in Algorithm 2. Except for the separate sampling, it is identical to a standard application of Monotone Local Search.

First, we argue correctness, i.e: that every preferred extension is enumerated at least once. Fix a preferred extension, say U, and suppose U contains b vertices in B and d vertices in D. Then, by the definition of set-containing-families, there exists a $S \in C_b$ such that $U \cap B \subseteq S$ and a $T \in C_d$ such that $U \setminus B \subseteq T$. Now we note that $S \subseteq B, T \subseteq V \setminus B$ to get:

$$|(S \sqcup T) \setminus U| = |S \setminus (U \cap B)| + |T \setminus (U \setminus B)| = (b' - b) + (d' - d)$$

Hence U is enumerated in the call to $\text{MASE}(S \cup T, (b' - b) + (d' - d))$ as required.

Now we argue the runtime. For a fixed b, d the calls to MASE have total run time $O^*(|C_b| \cdot |C_d| \cdot 2^{\frac{(b'-b)+(d'-d)}{2} + \frac{b'}{4}})$ for some choice of b', d'. Our overall running time is

$$O^*\left(\max_{0 \le b \le |B|} \max_{0 \le d \le |D|} |C_b| \cdot |C_d| \cdot 2^{\frac{(b'-b)+(d'-d)}{2} + \frac{b'}{4}} \right)$$

We can split this into two terms to get a complexity of $O^*\left(\max_{0 \le d \le |D|} |C_d| \cdot 2^{\frac{(d'-d)}{2}} \right)$ multiplied by $O^*\left(\max_{0 \le b \le |B|} |C_b| \cdot 2^{\frac{(b'-b)}{2} + \frac{b'}{4}} \right)$.

We will now analyze the running time with the right choices of b' and d'.

The first of these two terms can be analyzed using the analysis in [17]. This term is the complexity one attains for Monotone Local Search with a $O^*(2^{\frac{k}{2}})$ subroutine. Hence the analysis in [17] gives a complexity of $O^*((2 - \frac{1}{\sqrt{2}})^{|D|+o(|D|)})$.

We need the extended analysis presented in [23] to analyze the second term. This term is the complexity one attains for Monotone Local Search with a $O^*(2^{\frac{k}{2} + \frac{n-|X|}{4}})$ subroutine (where k is the parameter, $n = |U|$ the size of the underlying set and X is the set we are extending). Hence the analysis in [23] gives a complexity of $O^*((1 + 2^{\frac{1}{4}} - \frac{1}{\sqrt{2}})^{|B|+o(|B|)})$.

The papers we have cited also give a corresponding combinatorial upper bound on the number of preferred extensions. We summarize the above results as follows.

▶ **Theorem 10.** *Let $G = (V, E)$ be a digraph. Let r be the proportion of vertices in V that are in at least one 2-cycle.*

Then there exists a $O^*(((1+2^{\frac{1}{4}}-\frac{1}{\sqrt{2}})^r(2-\frac{1}{\sqrt{2}})^{1-r})^{|V|+o(|V|)}) \approx O^*((1.4822^r 1.2929^{1-r})^{|V|})$ time algorithm that enumerates all preferred extensions of G; however it may also enumerate some non-maximal admissible extensions. Furthermore, there are at most $O^*(((1+2^{\frac{1}{4}}-\frac{1}{\sqrt{2}})^r(2-\frac{1}{\sqrt{2}})^{1-r})^{|V|}) \approx O^*((1.4822^r 1.2929^{1-r})^{|V|})$ preferred extensions in G.

5 Improved Enumeration Algorithm for Oriented Graphs

Finally, we outline a branching algorithm with a finer analysis for oriented graphs. As with MASE, we will follow a standard template for branching algorithms. A summary of the necessary concepts can be found in Section 2. Our algorithm creates a search tree with at most φ^n leaves where $\varphi \approx 1.2321$ is the branching number for branching vector $(8, 1)$.

In Section 6 we will use the Oriented Translation (Theorem 2) to obtain a general enumeration algorithm parameterized by the number of vertices in at least one 2-cycle.

5.1 Overview

The overall structure of our algorithm is described in Algorithm 3.

The state of the algorithm consists of the subset Und $\subseteq V(G)$ and a queue of vertices Def (Und for undecided and Def for deferred). Und is the set of vertices we have yet to make a decision on whether we should include them. Def is a queue of vertices which have no outgoing arcs to Und. These vertices will be handled in the base case. While branching we can essentially assume the vertices in Def do not exist.

We maintain the following invariants. We outline why they are invariant, it is straightforward to verify that each case of our branching algorithm maintains these invariants.

1. Und and Def are disjoint. This holds as vertices are only ever deleted from or moved between Und and Def, never copied.
2. $G[\text{Def}]$ is a DAG where each vertex $v \in$ Def only attacks vertices that were added to Def before v. This is crucial for the base case since a DAG has 1 maximal admissible subset. This holds as only vertices with out-degree 0 in $G[\text{Und}]$ are ever moved to Def.
3. There is no attack from any vertex in Def to any vertex in Und. This holds for the same reason as Invariant 2.

Our measure is $\mu = |\text{Und}|$. For any instance, our algorithm creates a search tree with at most φ^μ leaves. Hence, calling OrientedEnumeration$(V(G), [])$ will return all preferred extensions of G by traversing a search tree with at most $\varphi^{|V(G)|}$ leaves.

5.2 Extra Notation

We will say a vertex has degree $(a, -)$ if it has in-degree a, a vertex has degree $(-, b)$ if it has out-degree b and a vertex has degree (a, b) if it has in-degree a and out-degree b.

5.3 Simplification Rules

Both of these are applicable in polynomial time and decrease $\mu = |\text{Und}|$.

▶ **Simplification Rule 2** (Out-degree 0)**.** *Let v be a vertex in $G[\text{Und}]$ with out-degree 0. Move v from* Und *to the end of the queue* Def*.*

Algorithm 3 Structure of Oriented maximal admissible enumeration algorithm.

```
Require: $\mathrm{Und} \cap \mathrm{Def} = \emptyset$.
Require: $G[\mathrm{Def}]$ is a DAG where each $v \in \mathrm{Def}$ only attacks vertices added to Def before $v$.
Require: There is no attack from any vertex in Def to any vertex in Und.
Ensure: Returns all maximal admissible subsets of $\mathrm{Und} \cup \mathrm{Def}$.
  function OrientedEnumeration(Und, Def)
    while Any simplification rule applies do
      Apply said simplification rule
    if Base Case applies then
      Solve the instance directly through the base case subroutine.
    else
      Let $b_i$ be the first branching rule that applies.
      Let $(U_1, D_1) \dots (U_r, D_r)$ be the subinstances obtained from applying $b_i$.
      Let $C_i = \mathrm{OrientedEnumeration}(U_i, D_i)$, for all $1 \le i \le r$.
      return Maximal_Subset_Collation($(U_1 \cup D_1, C_1), \dots, (U_r \cup D_r, C_r)$)
```

▶ **Simplification Rule 3** (In-degree 0). *Let v be any vertex in $G[\mathrm{Und}]$ with in-degree 0. Then by Invariant 3, v has in-degree 0 in $G[\mathrm{Und} \cup \mathrm{Def}]$. Applying Simplification Rule* ***(Undefendable)*** *we can set $\mathrm{Und} \leftarrow \mathrm{Und} \setminus N(v)$. After that, v has out-degree 0 in $G[\mathrm{Und}]$ and hence we move v from Und to Def.*

Our new instance $I' = (\mathrm{Und}', \mathrm{Def}')$ has:

- $\mathrm{Und}' = \mathrm{Und} \setminus N[v]$.
- $\mathrm{Def}' = \mathrm{Def} \cup \{v\}$.

Due to these rules, henceforth we may assume each vertex in $G[\mathrm{Und}]$ has in-degree ≥ 1, out-degree ≥ 1 and (total) degree ≥ 2.

5.4 Branching Rules

Our algorithm will apply the first rule applicable to the instance:

Case	Requirement to apply	Worst case branching number
Base	$\mathrm{Und} = \emptyset$.	Solves in $O^*(1)$, returns 1 set.
1	$\exists v \in G[\mathrm{Und}]$ with total degree ≥ 7.	Branching vector $(8,1)$, branching number $\varphi \approx 1.2321$.
2	$\exists v \in G[\mathrm{Und}]$ with degree $(1,-)$.	Branching vector $(4,3)$, branching number ≈ 1.221.
3	$\exists v \in G[\mathrm{Und}]$ with in-degree $\neq$ out-degree.	Branching vector $(6,5,5)$, branching number ≈ 1.2298.
4	$G[\mathrm{Und}]$ has a weakly connected component where every vertex has degree $(2,2)$.	Branching vector $(6,5,5)$, branching number ≈ 1.2298.
5	$G[\mathrm{Und}]$ has a weakly connected component where every vertex has degree $(3,3)$.	Branching vector $(7,7,7,7)$, branching number ≈ 1.219.
6	There is a weakly connected component in $G[\mathrm{Und}]$ where every vertex has in-degree $=$ out-degree.	Branching vector $(7,5,5)$, branching number ≈ 1.218.

We note the base case and cases 3 and 6 cover all possible inputs. Hence there will always be at least one applicable rule.

Our primary strategy is to pick a specific vertex v and do a 2-way branch, separately enumerating the maximal admissible subsets that include v and the ones that exclude v.

- In the branch where we include v we must exclude v's neighbors. Hence we create a new instance $I' = (\text{Und}', \text{Def})$ where $\text{Und}' = \text{Und} \setminus N(v)$.
  Now v is isolated in $G[\text{Und}']$ so by Simplification Rule 2 we may move v from Und' to Def'. Hence in our new instance we finally have:
  - $\text{Und}' = \text{Und} \setminus N[v]$.
  - $\text{Def}' = \text{Def} \cup \{v\}$.

  and hence $\mu(I') = \mu(I) - |N[v]| = \mu(I) - \deg(v) - 1$.
  As a technical note, not every subset enumerated in this branch contains v, however, every maximal admissible subset that contains v will be enumerated in this branch.
- In the branch where we exclude v our new instance $I' = (\text{Und}', \text{Def}')$ is:
  - $\text{Und}' = \text{Und} \setminus \{v\}$.
  - $\text{Def}' = \text{Def}$.

  and hence $\mu(I') = \mu(I) - 1$.

Hence our branching vector is $(\deg(v) + 1, 1)$. However, often our choice of v allows us to immediately apply our simplification rules to improve the branch where v is excluded.

Full proofs of each case can be found in the full version. A few short remarks:

In the base case, by Invariant 2, $G[\text{Und} \cup \text{Def}]$ is a DAG. The base case then follows trivially from Lemma 7.

Case 2 is a good example of the strategy of picking a specific v to apply a 2-way branch on. We will pick a v that allows us to apply our simplification rules in the branch where v is excluded. However, the choice of v will depend on some case analysis on degrees of vertices.

Case 3 is similar to Case 2, a specific v is chosen on which we apply a 2-way branch.

Case 4 is done through a graph classification theorem. We classify the family of oriented graphs where each vertex has in-degree 2, out-degree 2 and both in-neighbors adjacent (if there is a vertex with independent in-neighbors then we instead apply a 3-way branch on that vertex and its in-neighbors).

Case 5 is just a 4-way branch on any vertex v and its 3 in-neighbors.

For Case 6, we first show there is a vertex with degree $(3, 3)$ attacking a vertex with degree $(2, 2)$, say b. Then we apply a 3-way branch on b and its 2 in-neighbors.

5.5 Summary of results

In our base case we enumerate 1 extension in polynomial time.

Otherwise, we apply a branching rule with branching number $\leq \varphi$.

As in MASE, the Maximal Subset Collation subroutine does not incur additional overhead as the search tree's depth is bounded by $|V(G)|$ (see Subsection 3.2 for the argument which we can apply verbatim).

Applying the Combine Analysis Lemma and Combine Analysis Lemma for Enumeration we obtain:

► **Theorem 11.** *Let $G = (V, E)$ be an oriented graph. Then there is an algorithm that enumerates all preferred extensions of G with running time $O^*(\varphi^{|V|})$ where φ is the unique positive root of $1 - x^{-1} - x^{-8} = 0$, $\varphi \approx 1.23205 < 1.2321$.*

Furthermore, G has at most $\varphi^{|V|}$ preferred extensions.

6 Bounds on number of preferred extensions

In this section, we collect our results bounding the number of preferred extensions.

6.1 Bounds on general directed graphs

A tight upper bound is $O(3^{\frac{|V|}{3}})$ [16]. This bound is easily attainable since preferred extensions coincide with maximal independent sets(MIS) in graphs where every edge is in a 2-cycle.

We can also enumerate them in $O^*(3^{\frac{|V|}{3}})$. We note each MIS has a single maximal admissible subset. We then take a branching algorithm for MIS[22], allowing us to apply the Maximal Subset Collation subroutine to remove the non preferred extensions without additional overhead. Hence, all preferred extensions can be enumerated in $O^*(3^{\frac{|V|}{3}})$.

6.2 Parameterizing by Resolution Order

We will give bounds based on r, the proportion of vertices that are in at least one 2-cycle.

6.2.1 Lower Bound

An undirected 3-cycle has 3 preferred extensions. Hence, applying the Oriented Translation to it, we obtain an oriented structure with 6 vertices and 3 preferred extensions.

Our construction for lower bounding will be to include as many undirected 3-cycles as possible and then include as many oriented translations of 3-cycles as possible. We can include $\frac{r|V|}{3}$ undirected 3-cycles and $\frac{(1-r)|V|}{6}$ oriented translations, obtaining an AF with $\Omega((3^{\frac{r}{3}}3^{\frac{1-r}{6}})^{|V|}) \approx ((1.44^r 1.2^{1-r})^{|V|})$ preferred extensions.

6.2.2 Upper Bound

There are 3 different upper bounds that are all optimal in a different range. See also Figure 1.

- $(\varphi^{2r}\varphi^{1-r})^{|V|} \approx O((1.5180^r 1.2321^{1-r})^{|V|})$ where φ is the unique positive root of $1-x^{-1}-x^{-8}$. This bound is obtained from using the Oriented Translation along with Theorem 11. This is best for r up to around 0.6684.
- $O^*(((1+2^{\frac{1}{4}}-\frac{1}{\sqrt{2}})^r(2-\frac{1}{\sqrt{2}})^{1-r})^{|V|}) \approx O^*((1.4822^r 1.2929^{1-r})^{|V|})$, the bound from our 2-phase Monotone Local Search. This is best for a small range where $0.6685 \leq r \leq 0.8004$.
- $3^{\frac{|V|}{3}}$. This is best for $r \geq 0.8005$.

7 Conclusion

We again note that the concept of an admissible (resp. preferred) extension has also been studied as a *semikernel*[27] (resp. maximal *semikernel*) in graph theory. Hence our result may be interpreted as a combinatorial upper bound on the number of maximal *semikernels*, parameterized by the proportion of vertices in at least one 2-cycle.

References

1 Leila Amgoud and Claudette Cayrol. A Reasoning Model Based on the Production of Acceptable Arguments. *Annals of Mathematics and Artificial Intelligence*, 34(1-3):197–215, 2002. doi:10.1023/A:1014490210693.

2 Cyril Banderier, Jean-Marie Le Bars, and Vlady Ravelomanana. Generating functions for kernels of digraphs (Enumeration & asymptotics for a constraint from game theory). In *Proceedings of the 16th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2004)*, pages 91–105, 2004.

3 Ringo Baumann and Hannes Strass. Open Problems in Abstract Argumentation. In *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *Lecture Notes in Computer Science*, pages 325–339. Springer, 2015.

4 Trevor J. M. Bench-Capon. Value Based Argumentation Frameworks. In *Proceedings of the 9th International Workshop on Non-Monotonic Reasoning (NMR 2002)*, volume cs.AI/0207059, pages 443–454, 2002. URL: http://arxiv.org/abs/cs.AI/0207059.

5 Raymond Bisdorff. On enumerating the kernels in a bipolar-valued outranking digraph. Technical Report 6, Annales du Lamsade, 2006. hal-00118995.

6 Stefano Bistarelli, Fabio Rossi, and Francesco Santini. A Comparative Test on the Enumeration of Extensions in Abstract Argumentation. *Fundamenta Informaticae*, 140(3-4):263–278, 2015.

7 Martin Caminada. An Algorithm for Computing Semi-stable Semantics. In *Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2007)*, volume 4724 of *Lecture Notes in Computer Science*, pages 222–234. Springer, 2007.

8 Martin Caminada. An Algorithm for Computing Semi-stable Semantics. In *ECSQARU 2007: Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, volume 4724 of *Lecture Notes in Computer Science*, pages 222–234. Springer, Berlin, Heidelberg, 2007.

9 Federico Cerutti, Paul E. Dunne, Massimiliano Giacomin, and Mauro Vallati. Computing Preferred Extensions in Abstract Argumentation: A SAT-Based Approach. In *Proceedings of the 2nd International Workshop on Theory and Applications of Formal Argumentation (TAFA 2013)*, volume 8306 of *Lecture Notes in Computer Science*, pages 176–193. Springer, 2013.

10 Federico Cerutti, Massimiliano Giacomin, and Mauro Vallati. Algorithm Selection for Preferred Extensions Enumeration. In *Proceedings of the 5th International Conference on Computational Models of Argument (COMMA 2014)*, volume 266 of *Frontiers in Artificial Intelligence and Applications*, pages 221–232. IOS Press, 2014.

11 Federico Cerutti, Mauro Vallati, and Massimiliano Giacomin. On the impact of configuration on abstract argumentation automated reasoning. *International Journal of Approximate Reasoning*, 92:120–138, 2018.

12 Günther Charwat, Wolfgang Dvorák, Sarah Alice Gaggl, Johannes Peter Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation - A survey. *Artificial Intelligence*, 220:28–63, 2015.

13 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.

14 Sylvie Doutre and Jérôme Mengin. Preferred Extensions of Argumentation Frameworks: Query, Answering, and Computation. In *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2001.

15 Phan Minh Dung. On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.

16 Paul E. Dunne, Wolfgang Dvorák, Thomas Linsbichler, and Stefan Woltran. Characteristics of multiple viewpoints in abstract argumentation. *Artificial Intelligence*, 228:153–178, 2015.

17 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2016)*, pages 764–775. ACM, 2016.

18 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56(5):25:1–25:32, 2009.

19 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms.* Springer, 2010.

20 Hortensia Galeana-Sánchez and Xueliang Li. Semikernels and (k, l)-Kernels in Digraphs. *SIAM Journal on Discrete Mathematics*, 11(2):340–346, 1998.

21 Hortensia Galeana-Sánchez and Victor Neumann-Lara. On kernels and semikernels of digraphs. *Discrete Mathematics*, 48(1):67–76, 1984.

22 Serge Gaspers. *Exponential time algorithms: Structures, measures, and bounds.* PhD thesis, University of Bergen, 2008.

23 Serge Gaspers and Edward J. Lee. Exact Algorithms via Multivariate Subroutines. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 69:1–69:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.69.

24 Sanjay Modgil. Hierarchical Argumentation. In *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA 2006)*, pages 319–332, 2006. doi:10.1007/11853886_27.

25 Sanjay Modgil and Martin Caminada. Proof Theories and Algorithms for Abstract Argumentation Frameworks. In *Argumentation in Artificial Intelligence*, pages 105–129. Springer, 2009.

26 Sanjay Modgil and Henry Prakken. Resolutions in Structured Argumentation. In *Proceedings of the 4th International Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 310–321. IOS Press, 2012.

27 Victor Neumann-Lara. Seminúcleos de una digráfica. Technical report, Anales del Instituto de Matemáticas II, Universidad Nacional Autónoma México, 1971.

28 Samer Nofal, Katie Atkinson, and Paul E. Dunne. Algorithms for decision problems in argument systems under preferred semantics. *Artificial Intelligence*, 207:23–51, 2014.

29 Jayme Luiz Szwarcfiter and Guy Chaty. Enumerating the Kernels of a Directed Graph with no Odd Circuits. *Information Processing Letters*, 51(3):149–153, 1994.

30 Mauro Vallati, Federico Cerutti, and Massimiliano Giacomin. Argumentation Extensions Enumeration as a Constraint Satisfaction Problem: a Performance Overview. In *Proceedings of the International Workshop on Defeasible and Ampliative Reasoning (DARe@ECAI 2014)*, volume 1212 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.

31 Mauro Vallati, Federico Cerutti, and Massimiliano Giacomin. Argumentation Frameworks Features: an Initial Study. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 1117–1118. IOS Press, 2014.

32 John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1944.

Determinisation of Finitely-Ambiguous Copyless Cost Register Automata

Théodore Lopez
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
theodore.lopez@univ-amu.fr

Benjamin Monmege
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
benjamin.monmege@univ-amu.fr

Jean-Marc Talbot
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
jean-marc.talbot@univ-amu.fr

Abstract

Cost register automata (CRA) are machines reading an input word while computing values using write-only registers: values from registers are combined using the two operations, as well as the constants, of a semiring. Particularly interesting is the subclass of copyless CRAs where the content of a register cannot be used twice for updating the registers. Originally deterministic, non-deterministic variant of CRA may also be defined: the semantics is then obtained by combining the values of all accepting runs with the additive operation of the semiring (as for weighted automata). We show that finitely-ambiguous copyless non-deterministic CRAs (i.e. the ones that admit a bounded number of accepting runs on every input word) can be effectively transformed into an equivalent copyless (deterministic) CRA, without requiring any specific property on the semiring. As a corollary, this also shows that regular look-ahead can effectively be removed from copyless CRAs.

2012 ACM Subject Classification Theory of computation → Quantitative automata

Keywords and phrases Cost-register automata, Look-ahead removal, Unambiguity, Determinisation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.75

Funding This work has been funded by the DeLTA project (ANR-16-CE40-0007).

1 Introduction

Quantitative languages extend "classical" languages by associating with each word a weight or a cost from an algebraic structure. Such algebraic structures could be monoids, semirings, fields or any other convenient structures. Quantitative languages have been successfully applied to various domains such as natural language processing [15] or modelisation of stochastic systems [17, 16]. The seminal work on quantitative languages from Schützenberger [18] introduces the model of weighted automata, that associates with each word a weight from a semiring. The weight of a run is the *product* of its transition weights whereas the weights of the multiple runs on a single word (due to non-determinism) are combined by *sum*. Classical word languages are then the particular case of weighted languages over the Boolean semiring. There have been a long line of research that studied properties of weighted automata [10]. As for finite-state automata, two-way [6] and alternating [12, 7] automata have been considered, as well as extensions to infinite words [8].

Recently, Alur et al. [3] introduced another automata model for defining mappings from words to some algebraic structures (in particular semirings), named Cost Register Automata (CRA). These are deterministic machines equipped with a finite collection of registers storing values: while reading the input word, each transition reads an input letter and updates the registers by combining the current contents of registers and values from the considered

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 75; pp. 75:1–75:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

algebraic structure. A distinguished subclass of CRAs is the one of *copyless* CRAs: intuitively, for such a machine, the content of a register cannot be used twice for updating the registers. Properties regarding expressiveness of this class have been studied in [14, 13]. The decision problem of equivalence of copyless CRA on the tropical semiring is proved undecidable in [1]. Some other works have also considered the register minimisation problem (computing the smallest number of registers to define a particular function) for copyless CRAs [5, 9].

Following [14], we consider both deterministic and non-deterministic copyless CRAs on semirings (to distinguish them, we call NCRA the class of non-deterministic CRAs). Non-determinism is resolved, as in weighted automata, by the sum operation of the semiring: the value associated with some input word is computed as the sum of the values computed by each accepting run on this input. We investigate a limited form of non-determinism, k-ambiguity: a non-deterministic CRA is said to be k-ambiguous if it has at most k accepting runs per input. In the context of weighted automata, k-ambiguous weighted automata are strictly more powerful than deterministic weighted automata, yet less powerful than weighted automata, leading to an appealing class of weighted languages with good decision properties (the equivalence problem becomes decidable for k-ambiguous weighted automata over the $(\max, +)$-semiring [11]). Surprisingly, in the context of CRAs, our main result is:

▶ **Theorem 1.** *Every finitely-ambiguous copyless* NCRA *can be effectively transformed into an equivalent copyless (deterministic)* CRA*.*

Moreover, the example developed in [14, Theorem 2] allows one to build a linearly-ambiguous copyless NCRA that cannot be recognised with a finitely-ambiguous copyless NCRA, showing that our result cannot be improved regarding ambiguity. An alternative way to resolve non-determinism is to consider a regular look-ahead. When reading a word from left to right, the look-ahead provides some (regular) information about the unread suffix of the word that allows to determine the unique transition to be applied at each step. In this case, the machine is said to be deterministic with look-ahead. In [3], this class is introduced and named CRA-RLA. It is proved there that for copyless CRA-RLA, the look-ahead can be removed preserving the copyless property provided that the considered algebraic structure is extended with unary mappings (using the so-called *streaming string-to-tree transducers* [2] as an intermediate step). In [14], Mazowiecki and Riveros proved that copyless CRA, unlike weighted automata, are not closed under reverse but claimed that “Like for unambiguous copyless CRAs, we do not know if extending copyless CRAs with regular look-ahead results in a more expressive model”. However, they defined the subclass of bounded-alternation copyless CRAs which are closed under reverse and for which deterministic look-aheads do not increase expressiveness.

A look-ahead can be given as a complete co-deterministic automaton B and transitions of the CRA A are then parameterised by some state of B. It is folklore that one can compute the product of A and B to obtain a machine equivalent to A which is look-ahead free, now non-deterministic, but still unambiguous. This construction still applies when A is a copyless CRA and the product yields an unambiguous copyless CRA. Therefore, by Theorem 1, we close the open problem stated in [14]:

▶ **Theorem 2.** *Every copyless CRA with look-ahead can be effectively transformed into an equivalent copyless CRA.*

The article is structured as follows: in Section 2, we define CRAs as well as several subclasses (copyless CRAs, $\diamond$-less CRAs, and bounded-copy CRAs). They are defined by means of flow graphs representing the flow of registers during runs in an abstract way. The proof of Theorem 1 is given as a cascade of three transformations leading from one subclass to another one, that are described successively in Sections 3, 4 and 5.

2 Cost register automata

Terms and substitutions. Fix a semiring $\mathbb{S} = (S, +, \times, 0, 1)$ and a finite set of variables $\mathcal{X}$ disjoint from S. We denote by $\mathrm{Term}(\mathcal{X})$ the set of *terms* generated by the grammar:

$t ::= s \mid x \mid t + t \mid t \times t$

where $s \in S$ and $x \in \mathcal{X}$. For a term $t \in \mathrm{Term}(\mathcal{X})$, we denote by $\mathrm{Var}(t)$ its set of variables. We call t a *ground term* if $\mathrm{Var}(t) = \emptyset$, and then define $[\![t]\!] \in S$ to be the evaluation of t with respect to $\mathbb{S}$. We call a term $t \in \mathrm{Term}(\mathcal{X})$ *copyless* if every variable appears at most once in t. In the following, we will represent terms as binary trees, where leaves are labelled by variables or constants, and internal nodes are labelled by operations of the semiring.

▶ **Example 3.** On the semiring $(\mathbb{N}, +, \times, 0, 1)$, examples of terms are $3x + 1$ (we often make implicit the product operator) or $y + 2zx + 3$, making use of the associativity of both operators to drop useless parentheses. On the semiring $(\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$, the same terms are written $\min(3 + x, 1)$ and $\min(y, 2 + z + x, 3)$. Other examples of terms on the non-commutative semiring $(\mathfrak{P}(\{a, b\}^*), \cup, \cdot, \emptyset, \{\varepsilon\})$ of languages over alphabet $\{a, b\}$ are $\{a\}x\{\varepsilon, aab\} \cup y \cup \{b\}$. All these terms are copyless.

A *substitution* is a mapping $\sigma : \mathcal{X} \to \mathrm{Term}(\mathcal{X})$. We denote the set of all substitutions over $\mathcal{X}$ by $\mathrm{Subs}(\mathcal{X})$. If t is a term, we let $[x \mapsto t]$ be the substitution defined by $[x \mapsto t](x) = t$ and $[x \mapsto t](y) = y$ for all variables $y \neq x$. A *ground substitution* σ is a substitution where the term $\sigma(x)$ is ground for every $x \in \mathcal{X}$. Substitutions are extended canonically to a term morphism, and may thus be composed: $\sigma_1 \circ \sigma_2(x) = \sigma_1(\sigma_2(x))$.

A *valuation* is defined as a substitution of the form $\nu \colon \mathcal{X} \to S$. We denote the set of all valuations over $\mathcal{X}$ by $\mathrm{Val}(\mathcal{X})$. Clearly, any valuation ν composed with a substitution σ defines a ground substitution. We say that two terms t_1 and t_2 are equivalent (denoted by $t_1 \equiv t_2$) if $[\![\nu(t_1)]\!] = [\![\nu(t_2)]\!]$ for every valuation $\nu \in \mathrm{Val}(\mathcal{X})$. Similarly, we say that two substitutions σ_1 and σ_2 are equivalent (denoted by $\sigma_1 \equiv \sigma_2$) if $\sigma_1(x) \equiv \sigma_2(x)$ for every $x \in \mathcal{X}$.

Non-deterministic cost register automata. A non-deterministic CRA (NCRA) over the semiring $\mathbb{S}$ is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \Delta, I, \nu_{ini}, F, \varphi)$, where Q is a finite set of states, Σ is the input alphabet, $\mathcal{X}$ a finite set of registers, $\Delta \subseteq Q \times \Sigma \times \mathrm{Subs}(\mathcal{X}) \times Q$ is the finite transition relation with updates of the registers, $I \subseteq Q$ is the set of initial states, $\nu_{ini} \colon I \to \mathrm{Val}(\mathcal{X})$ defines an initial valuation of the registers for each initial state, F is the set of final states and $\varphi \colon F \to \mathrm{Term}(\mathcal{X})$ the final output function. Transition (q, a, σ, q') is denoted by $q \xrightarrow{a|\sigma} q'$.

A configuration of $\mathcal{A}$ is a tuple (q, ν) where $q \in Q$ and $\nu \in \mathrm{Val}(\mathcal{X})$ represents the current values in the registers of $\mathcal{A}$. Given a word $w = a_1 \cdots a_n \in \Sigma^*$, a *run* ρ of $\mathcal{A}$ over w is a sequence of configurations linked by transitions $(q_0, \nu_0) \xrightarrow{a_1|\sigma_1} (q_1, \nu_1) \xrightarrow{a_2|\sigma_2} \cdots \xrightarrow{a_n|\sigma_n} (q_n, \nu_n)$ such that $q_0 \in I$, $\nu_0 = \nu_{ini}(q_0)$, for $1 \leq i \leq n$, $q_{i-1} \xrightarrow{a_i|\sigma_i} q_i$ is a transition and $\nu_i(x) = [\![\nu_{i-1} \circ \sigma_i(x)]\!]$ for each $x \in \mathcal{X}$. A run is accepting if it ends in a final state $q_n \in F$. The output of an accepting run $\rho = (q_0, \nu_0) \xrightarrow{a_1|\sigma_1} \cdots \xrightarrow{a_n|\sigma_n} (q_n, \nu_n)$, denoted by $[\![\rho]\!]$, is the value $[\![\nu_n(\varphi(q_n))]\!]$. The output of $\mathcal{A}$ over w is defined as $[\![\mathcal{A}]\!](w) = 0$ if there is no accepting run of $\mathcal{A}$ over w, and $[\![\mathcal{A}]\!](w) = \sum_\rho [\![\rho]\!]$ over all accepting runs ρ over w otherwise. Two NCRAs are said *equivalent* if they compute the same output on every input word.

For some positive k, a NCRA $\mathcal{A}$ is *k-ambiguous* if there are at most k accepting runs over each word w. $\mathcal{A}$ is *finitely-ambiguous* if it is k-ambiguous for some k, and *unambiguous* if it is 1-ambiguous. A NCRA is said to be *deterministic*, and denoted by CRA, if I is a singleton and for all $q \in Q$, $a \in \Sigma$, there exists at most one state $q' \in Q$ and one register update $\sigma \in \mathrm{Subs}(\mathcal{X})$ such that $q \xrightarrow{a|\sigma} q' \in \Delta$.

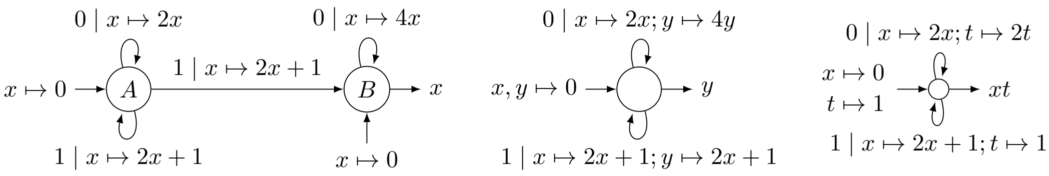


Figure 1 Three equivalent automata: an unambiguous copyless NCRA $\mathcal{A}_n$ (on the left), a $\diamond$-less CRA $\mathcal{A}_d$ (in the middle), a copyless CRA $\mathcal{A}_c$ (on the right).

▶ **Example 4.** Consider the copyless NCRA $\mathcal{A}_n$ depicted on the left of Figure 1 over the alphabet $\{0,1\}$ and the semiring $(\mathbb{N}, +, \times, 0, 1)$. It has a single register x, so that we hereby denote configurations by pairs $(q, \nu(x))$. Every word $w \in \{0,1\}^*$ has two runs, only one being accepting: thus $\mathcal{A}_n$ is unambiguous. For instance, on the word $w = 10100$, the two runs are

$$(A,0) \xrightarrow{1|x\mapsto 2x+1} (A,1) \xrightarrow{0|x\mapsto 2x} (A,2) \xrightarrow{1|x\mapsto 2x+1} (A,5) \xrightarrow{0|x\mapsto 2x} (A,10) \xrightarrow{0|x\mapsto 2x} (A,20)$$
$$(A,0) \xrightarrow{1|x\mapsto 2x+1} (A,1) \xrightarrow{0|x\mapsto 2x} (A,2) \xrightarrow{1|x\mapsto 2x+1} (B,5) \xrightarrow{0|x\mapsto 4x} (B,20) \xrightarrow{0|x\mapsto 4x} (B,80)$$

Therefore, A is a state computing the integer value of the word as binary representation with less significant bits first. To accept, we must jump into the unique final state B while reading the last 1 (or start directly in B if the word contains only 0s), then multiplying by 4 for each remaining 0: it is as if each 0 of the last block of 0s is considered to be duplicated by the CRA. The same function can also be recognised by a deterministic CRA, using two registers x and y, depicted in the middle of Figure 1. Instead of using non-determinism to guess the last 1 of the word, $\mathcal{A}_d$ always computes both the multiplications by 2 and 4 in separate registers x and y when reading 0. On each letter 1 though, the content of register y is reset to the same content as register x: operationally, this means that the content of register x must be duplicated when reading 1.

Flow of registers. A crucial notion in NCRAs is their ability to copy, or not, contents of registers into several registers. A NCRA is therefore called *copyless* if no register updates σ of Δ or terms of φ copy some register (see [3]). A more graphical definition of copyless can be achieved by gathering the *flows* of registers in a notion of *flow graphs*, that we define now, inspired by a close notion of dependency graph introduced in [4]:

▶ **Definition 5.** *A* flow graph *over the set of variables $\mathcal{X}$ is a (finite) directed acyclic (multi)graph (V, E) where $V = (\mathcal{X} \times \{0, 1, \dots, \ell_{max}\}) \uplus \{\Omega\}$ (with $\ell_{max} \geq 0$) is a finite set of vertices (x, ℓ) where ℓ is called the* layer *of the vertex (with ℓ_{max} being the maximal layer of the flow graph), and $E \colon V^2 \to \mathbb{N}$ being a multiset of edges satisfying:*

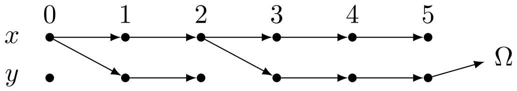
1. *E is consistent with the layers: $E((x_1,\ell_1),(x_2,\ell_2)) \neq 0 \implies \ell_2 = \ell_1 + 1$, and*
2. *Ω has no outgoing edges and all its ingoing edges come from the maximal layer: $E((x,\ell),\Omega) \neq 0 \implies \ell = \ell_{max}$.*

Each run $\rho = q_0 \xrightarrow{a_1|\sigma_1} q_1 \xrightarrow{a_2|\sigma_2} \cdots \xrightarrow{a_k|\sigma_k} q_k$ of a NCRA $\mathcal{A}$ is associated with the flow graph $G_{\mathcal{A}}(\rho) = (V, E)$ defined by: $V = \mathcal{X} \times \{0, \dots, k\} \cup \{\Omega\}$; for $\ell \in \{1, \dots, k\}$, and $x, y \in \mathcal{X}$, $E((x, \ell-1), (y, \ell))$ is the number of occurrences of the variable x in $\sigma_\ell(y)$; $E((x,k),\Omega)$ is the number of occurrences of x in $\varphi(q_k)$, and 0 if $q_k \notin F$.

Copyless restriction of NCRAs can be recovered directly on the flow graphs generated by their runs. For this reason, we say that a vertex of a flow graph is a *copy vertex* if it is the source of at least two edges. We use those to define two other related properties of the flow graphs that will be used in the following.

▶ **Definition 6.** *A flow graph (V, E) is* diamondless *(shorten as* $\diamond$-less *in the following) if there is at most one path linking every pair of vertices. It is called* k-copy *if every vertex can reach at most k copy vertices. We say that it is* copyless *if it is 0-copy.*

▶ **Example 7.** Consider once again the word 10100, and the unique run of the CRA $\mathcal{A}_d$ of Figure 1. Here is a pictural representation of the associated flow graph:



One can observe on this picture that $\mathcal{A}_d$ is not copyless, since register x is copied when reading letter 1. However, there are no diamonds (neither in this particular run, nor in any possible run), which means that $\mathcal{A}_d$ is $\diamond$-less.

By extension, a NCRA is said to be copyless (resp. $\diamond$-less or k-copy) if the flow graphs of all possible accepting runs of the NCRA are copyless (resp. $\diamond$-less or k-copy). It is said to be bounded-copy if it is k-copy for a certain value k. This alternative definition of copyless NCRA is equivalent to one of [3], whenever CRAs are supposed to be trimmed (i.e. all states are reachable from the initial ones and can reach a final state). Note that in the flow graphs of a $\diamond$-less CRAs, the multiset E is indeed a set (all pairs evaluate to 0 or 1). However, this does not imply that the CRA is copyless since a register can appear in the updates of two different registers. The $\diamond$-less property simply ensures that every register will flow at most once in the final output, in every possible execution: there may exist copies, but when it is the case, we are sure that at most one copy will resist up to the end; however, this exact copy can not be known yet as it might depend on the input word.

Contribution. Our main result is Theorem 1 stated already in the introduction. As the class of copyless CRA is obviously included into the class of finitely-ambiguous copyless NCRA, this result implies that these two classes define the same family of functions.

▶ **Example 8.** Our running example can indeed be recognised by the copyless CRA $\mathcal{A}_c$ on the right of Figure 1 which keeps in a register x the binary value of the input, and keeps in another register t the powers of 2 corresponding to the current longest suffix of letter 0. We multiply x by t in the final output function, and reset t when reading a letter 1.

Our construction is split into a cascade of transformations detailed in the next sections:

$$\begin{array}{c}\text{finitely-ambiguous}\\ \text{copyless NCRA}\end{array} \xrightarrow{\text{Prop. 9}} \begin{array}{c}\diamond\text{-less}\\ \text{CRA}\end{array} \xrightarrow{\text{Prop. 11}} \begin{array}{c}\text{bounded-copy}\\ \text{CRA}\end{array} \xrightarrow{\text{Prop. 19}} \begin{array}{c}\text{copyless}\\ \text{CRA}\end{array}$$

The first step in the construction is given in Section 3: a determinisation procedure of the finitely-ambiguous copyless CRA allows us to build a (deterministic) CRA, that may not be copyless, thus trading non-determinism for copies. This new CRA will indeed be $\diamond$-less. The second step is to reduce the number of copies in order to build an equivalent CRA that is bounded-copy: this is not trivial since the $\diamond$-less property does not forbid that unboundedly many copies can be performed. The idea is to delay copies until a moment where we know that enough copies have become useless. This more difficult step is the main contribution of this article and presented in Section 4. Finally, it remains to show in Section 5 how to remove all copies of the bounded-copy CRA, by replicating the registers as many times as needed.

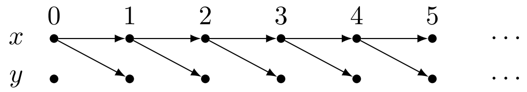
3 From finitely-ambiguous CRAs to diamondless CRAs

As a first step in our construction, we start by transforming every k-ambiguous NCRA into a $\diamond$-less CRA: this is a determinisation step where we maintain replicas of the registers, indexed by the states and a number in $\{1, \dots, k\}$ to distinguish the k possible runs.

▶ **Proposition 9.** *For every finitely-ambiguous copyless* NCRA*, we can compute an equivalent* $\diamond$*-less* CRA.

▶ **Example 10.** Starting from the unambiguous copyless NCRA $\mathcal{A}_n$ of Figure 1, we obtain a $\diamond$-less CRA $\mathcal{A}'$ isomorphic to the CRA $\mathcal{A}_d$: the single state represents the set $\{A, B\}$ of states of $\mathcal{A}$, while registers (x, A) and (x, B) of $\mathcal{A}'$ are the registers x and y on the picture.

The next step aims to limit the number of copies of the $\diamond$-less CRA. Indeed, $\diamond$-less CRAs are not necessarily copyless, and may even copy certain registers an unbounded number of times (in arbitrarily long runs). This is the case for the $\diamond$-less CRA $\mathcal{A}_d$ of Figure 1: for instance, the flow graph associated with the run over the word $11111\cdots$ is of the form



and has thus an unbounded number of copies of register x, that are all reachable from vertex $(x, 0)$; thus, $\mathcal{A}_d$ is not bounded-copy.

4 From diamondless CRAs to bounded-copy CRAs

In this section, we prove the most difficult step of the overall construction:

▶ **Proposition 11.** *For every* $\diamond$*-less* CRA*, we can construct an equivalent bounded-copy* CRA.

As seen in the example of the previous section, this result is not straightforward as $\diamond$-less CRAs may have an unbounded number of copies. Our approach somehow extends the one developed for streaming string transducers in [4]. However, the proof is simpler in the case of transducers where only a single operation exists (the product of a monoid). This allows one to represent in an alternative way the valuation of a register as a product of constants and other registers: instead of storing in registers the value of these products, it is going to be abstracted, storing in fresh registers the constant values separating two consecutive registers in products. In the setting of semirings, with two operations, it is much more intricate to do so, since the content of a register has to be viewed now as a term involving both operations. Hence, it is less clear a priori what could be the constants separating two "consecutive" registers of this term. We start by clarifying this, introducing special terms we call *shapes*, associated with *coefficients* that are these separating constants. Similar kind of shapes are used in [14, Theorem 3] to remove unambiguous non-determinism from bounded-alternation NCRAs, i.e. NCRAs that can only alternate a bounded number of times between sums and products in the register updates. The treatment of shapes is more complex in our case since we have no such limitation on alternations.

Shapes and coefficients. The *shape* of a term t over the set of variables $\mathcal{X}$ is a term with no constants, but with additional variables, obtained as follows. First, all constants are removed from the term to obtain a term $\tilde{t}$ where all leaves are labelled with variables of $\mathcal{X}$: for instance, if $t = (3x_1(5+2)+4) \times (2x_2) + 3$, then $\tilde{t} = x_1 \times x_2$. Doing so, we lose much information on the term, that we then recover by decorating every subterm t' of $\tilde{t}$ with some fresh coefficients α, β, γ, replacing the root of t' by $\alpha t'\beta + \gamma$: α, β represent respectively the left and the right multiplicative coefficient, and γ the additive coefficient. The *shape* of t is the term obtained by decorating each subterm of $\tilde{t}$. It is associated with a valuation χ mapping each coefficient of the shape to its value. On the example, the shape obtained is $\alpha_3[(\alpha_1 x_1 \beta_1 + \gamma_1) \times (\alpha_2 x_2 \beta_2 + \gamma_2)]\beta_3 + \gamma_3$, and the valuation of its coefficients is defined by $\chi(\alpha_1) = 3$, $\chi(\beta_1) = 7$, $\chi(\gamma_1) = 4$, $\chi(\alpha_2) = 2$, $\chi(\beta_2) = \chi(\alpha_3) = \chi(\beta_3) = 1$, $\chi(\gamma_2) = 0$, $\chi(\gamma_3) = 3$.[1] For the special case of a constant term t (without any variables), the shape is reduced to a special coefficient ω.

Shapes are canonical way to store terms. In particular, note that there is only a finite number of shapes of *copyless* terms over $\mathcal{X}$, denoted by Shape($\mathcal{X}$), though there are infinitely many possible coefficient valuations associated with these shapes. This will allow us to store the shapes in states of the bounded-copy CRA, while keeping in registers the valuations of coefficients (that could not fit in a CRA with a finite number of states).

Given a shape τ and an associated coefficient valuation χ, we denote by $\chi(\tau)$ the term obtained by replacing each coefficient of τ by its value: thus, $\chi(\tau)$ is a term over variables $\mathcal{X}$.

▶ **Proposition 12.** *There is a linear-time algorithm that, given a term t, builds a shape τ of t and a coefficient valuation χ, such that t and $\chi(\tau)$ are equivalent terms.*

Unfolding of a CRA. In the rest of the section, we let $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \Delta, I = \{q_{ini}\}, \nu_{ini}, F, \varphi)$ be a $\diamond$-less CRA. We construct another CRA $\mathcal{A}_\infty = (Q', \Sigma, \mathcal{X}', \Delta', I = \{q'_{ini}\}, \nu'_{ini}, F', \varphi')$ equivalent to $\mathcal{A}$, by unfolding: states contain shapes, and registers store the valuations of all corresponding coefficients. At first, $\mathcal{A}_\infty$ will thus have an infinite number of states: we explain afterwards how to reduce it to a finite number.

States of $\mathcal{A}_\infty$ are pairs (q, s), where q is a state of $\mathcal{A}$ and s maps each pair $(x, \ell) \in \mathcal{X} \times \{1, \ldots, \ell_{max}\}$, where $\ell_{max} \in \mathbb{N}$ depends on the state of $\mathcal{A}_\infty$, to a term of Shape($\mathcal{X} \times \{\ell - 1\}$), and each pair $(x, 0)$ to a shape of the form ω: s records the register updates applied so far, keeping only the shapes in memory. We call s the *shape substitution* of the state. Moreover, we enforce all coefficients appearing in all the shapes of s to be different. Notice that such a state (q, s) is associated uniquely with a flow graph $G = (\mathcal{X} \times \{0, \ldots, \ell_{max}\} \uplus \{\Omega\}, E)$ where E is the set[2] defined by $((x, \ell - 1), (y, \ell)) \in E$ iff $(x, \ell - 1)$ appears in $s(y, \ell)$, and $((x, \ell_{max}), \Omega) \in E$ iff $x \in \varphi(q)$. In the following, we use the notions of layers originating from flow graphs directly on s. All vertices of the flow graph that cannot reach a vertex of the maximal layer are useless : their value will not be used in the output of the CRA. Hence, we clean up s by mapping them to constant coefficients ω. In the following, we always consider that shape substitutions s are cleaned up this way.

Registers of $\mathcal{A}_\infty$ are all the possible coefficients appearing in its states (notice that there is an infinite number of them). However, at each point of the execution of $\mathcal{A}_\infty$, only coefficients that appear in the shape substitution of the current state are initialised, other registers being useless at this point. We may call *coefficients* the registers of $\mathcal{A}_\infty$ to emphasise their role.

The initial state is $q'_{ini} = (q_{ini}, s_{ini})$ with s_{ini} mapping $(x, 0)$ to a distinct coefficient ω_x for all x ($\ell_{max} = 0$ for this state); hence, only coefficients ω_x appear in these shapes of q'_{ini} and their value is given by $\nu'_{ini}(q'_{ini})(\omega_x) = \nu_{ini}(q_{ini})(x)$. All other registers can be set to 0 (their value will never be used in the following).

[1] Notice that the use of separate left and right multiplicative is only necessary in non-commutative semirings: in commutative ones, we could merge both of them in a single α coefficient.

[2] E turns out to be not some arbitrary multiset, as we start from a $\diamond$-less CRA $\mathcal{A}$

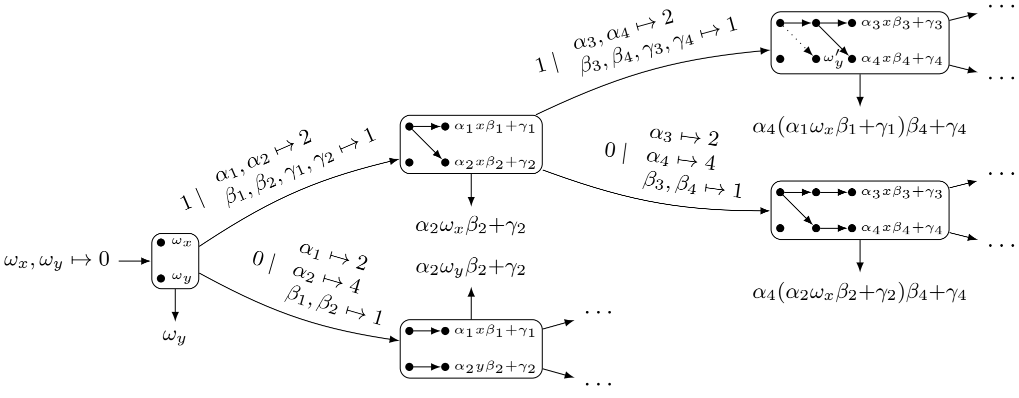

$\omega_x, \omega_y \mapsto 0$
ω_x
ω_y
ω_y
$1 \setminus \alpha_1, \alpha_2 \mapsto 2$
$\beta_1, \beta_2, \gamma_1, \gamma_2 \mapsto 1$
$0 \,/\, \alpha_1 \mapsto 2$
$\alpha_2 \mapsto 4$
$\beta_1, \beta_2 \mapsto 1$
$\alpha_1 x \beta_1 + \gamma_1$
$\alpha_2 x \beta_2 + \gamma_2$
$\alpha_2 \omega_x \beta_2 + \gamma_2$
$\alpha_2 \omega_y \beta_2 + \gamma_2$
$\alpha_1 x \beta_1 + \gamma_1$
$\alpha_2 y \beta_2 + \gamma_2$
$1 \setminus \alpha_3, \alpha_4 \mapsto 2$
$\beta_3, \beta_4, \gamma_3, \gamma_4 \mapsto 1$
$0 \,/\, \alpha_3 \mapsto 2$
$\alpha_4 \mapsto 4$
$\beta_3, \beta_4 \mapsto 1$
$\alpha_3 x \beta_3 + \gamma_3$
ω'_y
$\alpha_4 x \beta_4 + \gamma_4$
$\alpha_4(\alpha_1 \omega_x \beta_1 + \gamma_1)\beta_4 + \gamma_4$
$\alpha_3 x \beta_3 + \gamma_3$
$\alpha_4 x \beta_4 + \gamma_4$
$\alpha_4(\alpha_2 \omega_x \beta_2 + \gamma_2)\beta_4 + \gamma_4$
...

Figure 2 An infinite CRA equivalent to the $\diamond$-less CRA $\mathcal{A}_d$.

Final states of F' are all pairs (q, s) with $q \in F$, and the associated final output φ' is the term using only coefficients obtained by applying the shape substitution as many times as the number of layers from the maximal layer, in order to remove all variables (x, ℓ): $\varphi'(q, s) = \big[x \mapsto s^{\ell_{max}+1}(x, \ell_{max}) \quad \forall x \in \mathcal{X}\big]\,\big(\varphi(q)\big)$ where s^h is the composition of s with itself h times, and ℓ_{max} is the maximal layer of the flow graph associated with s.

We finally describe the transitions of $\mathcal{A}_\infty$. For all states $(q, s) \in Q'$ and transitions $q \xrightarrow{a|\sigma} q'$ in Δ, we add a transition $(q, s) \xrightarrow{a|\sigma'} (q', s')$ in Δ', where s' is the extension of s with an additional layer. If the maximal layer of s is ℓ_{max}, then the maximal layer of s' is $\ell_{max} + 1$. For the additional maximal layer $\ell_{max} + 1$, for all $x \in \mathcal{X}$, the term $\sigma(x)$ associated with the update of the register x in $\mathcal{A}$ can be decomposed, by Prop. 12, into a shape τ and a valuation χ of its fresh coefficients: we have $\chi(\tau) \equiv \sigma(x)$. Then, we let $s'(x, \ell_{max} + 1)$ be the shape τ in which every variable y is replaced by (y, ℓ_{max}). Corresponding (fresh) coefficients κ of τ are set to their value in χ: $\sigma'(\kappa) = \chi(\kappa)$. Layers $0, 1, \ldots, \ell_{max}$ are kept intact, except that vertices that can no longer reach layer $\ell_{max} + 1$ are mapped to a constant shape ω. More precisely, for all $\ell \in \{0, 1, \ldots, \ell_{max}\}$ and $x \in \mathcal{X}$, if (x, ℓ) can reach layer $\ell_{max} + 1$ (in the so-extended flow graph) then $s'(x, \ell) = s(x, \ell)$ and the corresponding coefficients κ remain the same by the update $\sigma'(\kappa) = \kappa$. Otherwise $s'(x, \ell)$ is mapped to a constant shape ω and $\sigma'(\omega) = 0$: coefficients of $s(x, \ell)$ are freed, e.g. by resetting them to 0 by σ'.

▶ **Example 13.** We illustrate this construction on the $\diamond$-less CRA $\mathcal{A}_d$ of Figure 1. A portion of the infinite CRA is shown in Figure 2. We depict in each state the associated flow graph (without the output vertices) as well as the shapes that are different than the corresponding shapes in the predecessor state. In the updates, we only show the values different from 0. Notice the dotted edge in the state reached after having read word 11: this edge disappears from the flow graph since the vertex $(y, 1)$ can no longer reach the maximal layer.

The infinite CRA $\mathcal{A}_\infty$ satisfies the following invariant: each run ρ of $\mathcal{A}$, ending in state q, is bijectively mapped to a run ρ' of $\mathcal{A}_\infty$ that ends in a state (q, s) associated with a flow graph isomorphic to $G_{\mathcal{A}}(\rho)$. Moreover,

▶ **Invariant 14.** *For all words w, if (q, ν) is the (unique) configuration reached by $\mathcal{A}$ over word w, the configuration $((q', s), \nu')$ reached by $\mathcal{A}'$ reading w satisfies: $q = q'$ and for every $x \in \mathcal{X}$, $\nu(x) = [\![\nu' \circ s^{\ell_{max}+1}(x, \ell_{max})]\!]$ where $\ell_{max} + 1$ is the number of layers in s.*

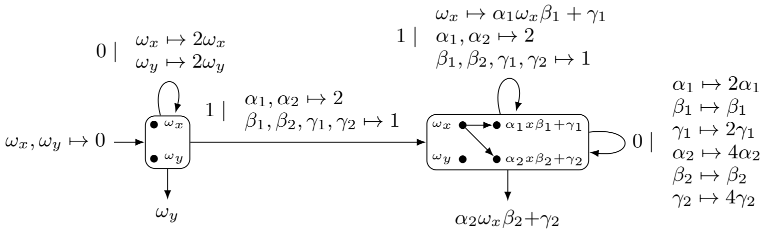

0 | $\omega_x \mapsto 2\omega_x$
$\omega_y \mapsto 2\omega_y$
$\omega_x, \omega_y \mapsto 0$
ω_x
ω_y
ω_y
1 | $\alpha_1, \alpha_2 \mapsto 2$
$\beta_1, \beta_2, \gamma_1, \gamma_2 \mapsto 1$
1 | $\omega_x \mapsto \alpha_1\omega_x\beta_1 + \gamma_1$
$\alpha_1, \alpha_2 \mapsto 2$
$\beta_1, \beta_2, \gamma_1, \gamma_2 \mapsto 1$
ω_x
ω_y
$\alpha_1 x \beta_1 + \gamma_1$
$\alpha_2 x \beta_2 + \gamma_2$
$\alpha_2\omega_x\beta_2 + \gamma_2$
0 | $\alpha_1 \mapsto 2\alpha_1$
$\beta_1 \mapsto \beta_1$
$\gamma_1 \mapsto 2\gamma_1$
$\alpha_2 \mapsto 4\alpha_2$
$\beta_2 \mapsto \beta_2$
$\gamma_2 \mapsto 4\gamma_2$

Figure 3 Finite CRA obtained by merging copyless layers of the CRA in Figure 2.

In particular, both CRAs are equivalent by construction since

$$
\begin{aligned}
[\![\mathcal{A}]\!](w) &= [\![\nu(\varphi(q))]\!] \\
&= [\![\left[x \mapsto \nu' \circ s^{\ell_{max}+1}(x, \ell_{max})\ \forall x \in \mathcal{X}\right](\varphi(q))]\!] = [\![\nu'(\varphi'(q,s))]\!] = [\![\mathcal{A}']\!](w)
\end{aligned}
$$

The goal is now to make $\mathcal{A}_\infty$ finite by contracting the flow graph associated with its states. The main operation is the merging of a layer with the previous one; this merge will require some copies of registers. Starting from the finite $\diamond$-less (but not necessarily bounded-copy) CRA $\mathcal{A}$, the merge operation will turn $\mathcal{A}_\infty$ into a finite CRA, that is moreover bounded-copy.

Merge of two consecutive layers. Let s be the shape substitution in a state of $\mathcal{A}_\infty$ with maximal layer ℓ_{max} and let L be some layer. We now explain how to merge the layers $L-1$ and L of s, leading to a new shape substitution s' with one layer less. This will require an update σ' of the coefficients appearing in the corresponding shapes. s' and σ' are defined as follows for all layers $\ell \in \{0, 1, \ldots, \ell_{max}\}$:

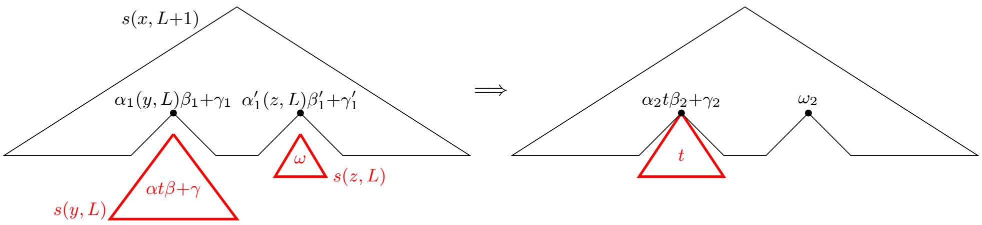
1. if $\ell < L-1$, nothing is changed: for all $x \in \mathcal{X}$, $s'(x,\ell) = s(x,\ell)$ and all the corresponding coefficients κ are left unchanged, i.e. $\sigma'(\kappa) = \kappa$;
2. if $\ell \geq L$, we simply shift down all the layers by one: for all $x \in \mathcal{X}$, $s'(x,\ell) = s(x,\ell+1)$ and the corresponding coefficients κ are left unchanged too;
3. if $\ell = L-1$, we must incorporate the shapes in-between layers $L-1$ and L into layer $L-1$:
   - if $s(x,L)$ is a constant shape ω, then we simply shift it as before: $s'(x,L-1) = \omega$ and coefficient ω is left unchanged by σ';
   - otherwise, consider the term $t = s^2(x,L)$ obtained by replacing all variables y appearing in the shape $s(x,L)$ by the shape $s(y,L-1)$. Unfortunately, t may not be a shape anymore, but Proposition 12 allows us to recover a new shape τ from t with new coefficients whose values are given by χ. We thus have to store $s'(x,L-1) = \tau$ in the state of $\mathcal{A}_\infty$ and update the coefficients accordingly, so that $\sigma'(s'(x,L-1)) = \chi(\tau) \equiv t$.

The most promising merge to perform in the infinite CRA $\mathcal{A}_\infty$ in a state (q,s) is the merge of a *copyless* layer L (such that the substitution $[x \mapsto s(x,L)]$ is copyless) with layer $L-1$.

▶ **Example 15.** Consider the infinite CRA $\mathcal{A}_\infty$ built in Figure 2. Three of its depicted states contain copyless layers: notice in particular that, in the state above right, it becomes possible to merge the first two layers after removing the useless (dotted) edge in the flow graph. After these merges, we obtain the finite copyless (and thus bounded-copy) CRA of Figure 3.

The definition of the new transition function (and coefficient updates) given above is correct, but not precise enough to prove afterwards that we obtain a bounded-copy CRA. Therefore, we need to describe an operational method to build s' and σ' with as few copies of coefficients as possible. There are two difficulties.

First, when glueing together $s(x, L+1)$ with all the shapes $s(y, L)$ (with (y, L) appearing in $s(x, L+1)$), we need to gather the coefficients at the glueing interface. There are two cases:


$s(x,L{+}1)$
$\alpha_1(y,L)\beta_1{+}\gamma_1$
$\alpha_1'(z,L)\beta_1'{+}\gamma_1'$
ω
$s(z,L)$
$\alpha t\beta{+}\gamma$
$s(y,L)$
$\Longrightarrow$
$\alpha_2 t\beta_2{+}\gamma_2$
ω_2
t

Consider first the case where $s(y, L)$ is a (non-constant) shape of the form $\alpha t\beta + \gamma$ with t a term. Let $\alpha_1(y, L)\beta_1 + \gamma_1$ be the subterm of $s(x, L+1)$ where (y, L) appears. The glueing should replace this subterm by $\alpha_1(\alpha t\beta + \gamma)\beta_1 + \gamma_1 \equiv \alpha_1\alpha t\beta\beta_1 + \alpha_1\gamma\beta_1 + \gamma_1$. This is obtained by replacing it by a shape $\alpha_2 t\beta_2 + \gamma_2$ with α_2, β_2 and γ_2 fresh coefficients that we set via $\sigma'(\alpha_2) = \alpha_1\alpha$, $\sigma'(\beta_2) = \beta\beta_1$ and $\sigma'(\gamma_2) = \alpha_1\gamma\beta_1 + \gamma_1$. Coefficients of t are preserved by the update σ'. Notice that σ' uses twice the content of coefficients α_1 and β_1.

The other case is the one where $s(y, L)$ is a constant shape ω. Then, we replace in $s(x, L+1)$ the subterm $\alpha_1'(y, L)\beta_1' + \gamma_1'$ by a fresh constant coefficient ω_2, set via $\sigma'(\omega_2) = \alpha_1'\omega\beta_1' + \gamma_1'$.

After glueing, in case where certain shapes we glued were constant shapes ω, a final step is required. Either the term does not contain variables of $\mathcal{X}$ anymore (all variables have been replaced by constant shapes ω), and we then replace the whole term t by a constant shape ω with $\sigma'(\omega) = t$. Or, there are still variables of $\mathcal{X}$, in which case we need to remove all ω-leaves. They are removed one by one, thus modifying the term and the update of coefficients. If there is a subterm of the form $\alpha_1(\omega_2 \odot \omega_3)\beta_1 + \gamma_1$ with two ω-leaves (and $\odot \in \{+, \times\}$), we replace this subterm by a fresh constant coefficient ω_4 set via $\sigma'(\omega_4) = \alpha_1(\omega_2 \odot \omega_3)\beta_1 + \gamma_1$. Once removed all those terms, there might remain isolated ω-leaves: consider thus a subterm with a single ω-leaf, e.g. of the form $\alpha_1\big((\alpha_2 t\beta_2 + \gamma_2) \odot \omega_3\big)\beta_1 + \gamma_1$ with t a shape without any ω. Notice that we can rewrite this term as:

$$\alpha_1\big((\alpha_2 t\beta_2 + \gamma_2) \odot \omega_3\big)\beta_1 + \gamma_1 \equiv \begin{cases} \alpha_1\alpha_2 t\beta_2\beta_1 + \alpha_1(\gamma_2 + \omega_3)\beta_1 + \gamma_1 & \text{if } \odot = + \\ \alpha_1\alpha_2 t\beta_2\omega_3\beta_1 + \alpha_1\gamma_2\omega_3\beta_1 + \gamma_1 & \text{if } \odot = \times \end{cases}$$

Then, this subterm is replaced by a shape $\alpha_4 t\beta_4 + \gamma_4$ with the coefficient updates:

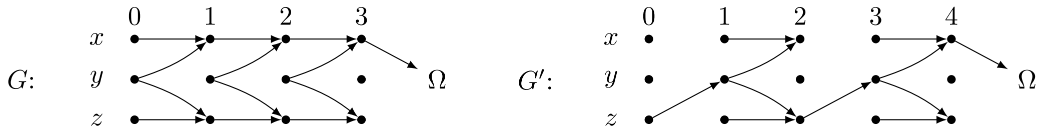
$$\sigma'(\alpha_4) = \alpha_1\alpha_2 \qquad \sigma'(\beta_4) = \begin{cases} \beta_2\beta_1 & \text{if } \odot = + \\ \beta_2\omega_3\beta_1 & \text{if } \odot = \times \end{cases} \qquad \sigma'(\gamma_4) = \alpha_1(\gamma_2 \odot \omega_3)\beta_1 + \gamma_1$$

Notice the copy of ω_3 in the case $\odot = \times$, and the copies of α_1 and β_1 in both cases.

During a whole merge step, we can show that the only coefficients to be copied are the ones of layer $L+1$ and of the constant shapes ω of layer L, and these are copied at most twice.

Primarily-copyless layer. The merge of copyless layers with their predecessor may not be sufficient to obtain a finite CRA, as shown in the following more involved example.

▶ **Example 16.** Consider the $\diamond$-less CRA $\mathcal{A}_3$ over the semiring $(\mathfrak{P}(\{a, b\}), \cup, \cdot, \emptyset, \{\varepsilon\})$ and alphabet $\{a, b\}$, with one state q and three registers x, y, z, that outputs $\varphi(q) = x$ and with transitions $q \xrightarrow{a|\sigma_a} q$ and $q \xrightarrow{a|\sigma_b} q$ using updates $\sigma_a = [x \mapsto xy, y \mapsto a, z \mapsto zya]$ and $\sigma_b = [x \mapsto \varepsilon, y \mapsto zb, z \mapsto \varepsilon]$. Here are the flow graphs G and G' obtained after reading the input words aaa and $baba$ respectively:



The flow graph G' illustrates that $\mathcal{A}_3$ is not bounded-copy, since the flow graphs on input words $(ba)^n$ require a chain of n copies. In G though, no layer is copyless, and thus no merging based on copyless layers can be performed. Notice that the value of register z is not used in the output, but it would be so if the word aaa is extended with b: thus, we cannot simply remove register z to recover some copyless layers. Here, the idea is rather to notice that the second time we see register y being copied (in-between layers 1 and 2), we learn the fact that indeed the previous content of register y is not copied many times (because of the $\diamond$-less hypothesis): if we do not consider this copy anymore, layer 2 becomes copyless and can thus safely be merged with layer 1. It will cost copies, but not too much, as we will see later.

Hence, removing copyless layers is not a sufficient criterion to obtain a *finite* (bounded-copy) CRA from $\mathcal{A}_\infty$. To define the correct criterion, we distinguish some special vertices in a flow graph. A *source* is a vertex (x, ℓ) without predecessors (in the state of $\mathcal{A}_\infty$, this means that $s(x, \ell) = \omega$). The sources of a vertex are all its ancestors that are sources: in the previous flow graph G of Example 16, sources of $(x, 2)$ are $(x, 0)$, $(y, 0)$ and $(y, 1)$. A source is *primary* if it is a source, with lowest layer, of a vertex in the maximal layer: $(x, 0)$, $(y, 0)$, $(z, 0)$ and $(y, 3)$ are all the primary sources of G (note that $(y, 3)$ is source with lowest layer of $(y, 3)$ itself), while $(y, 1)$ and $(y, 2)$ are not. A vertex is *primary* if it is a descendant of a primary source. All other vertices are called *secondary*: in G, $(y, 1)$ and $(y, 2)$ are all the secondary vertices. In particular, all vertices of the maximal layer are primary. On the minimal layer, all vertices that can reach the maximal layer are primary sources. Notice that whenever the flow graph is extended, some primary sources may turn secondary, and new primary sources may appear, but only on the maximal layer; also, vertices may not reach the maximal layer anymore in which case they are removed. A layer L is *primarily-copyless* when only secondary vertices of layer $L - 1$ are possibly copied, i.e. all primary vertices $(x, L - 1)$ are linked to a single vertex of layer L. In G, only layers 2 and 3 are primarily-copyless.

Obviously, we do not build the infinite CRA $\mathcal{A}_\infty$, but instead build a finite CRA $\mathcal{A}'$ step by step by extending it along transitions and directly merging each primarily-copyless layers with the previous one, as much as possible. The idea is that merging such primarily-copyless layers will cost some copies of coefficients, but not too many, yielding the fact that $\mathcal{A}'$ is a bounded-copy CRA with a finite number of states equivalent to the $\diamond$-less CRA $\mathcal{A}$. These properties are proved in the rest of the section, and imply Proposition 11.

$\mathcal{A}'$ is equivalent to $\mathcal{A}$. By Invariant 14, the infinite CRA (without merge of primarily-copyless layers) is equivalent to $\mathcal{A}$. Then, we prove that it remains true when we perform the merge of two consecutive layers. It is a consequence of the following invariant:

▶ **Invariant 17.** *Let s be a shape substitution with $\ell_{max} + 1$ layers, and let s' and σ' be the shape substitution and update obtained by merging layers L and $L - 1$ from s. Then* $\sigma' \circ s'^{\ell_{max}}(x, \ell_{max} - 1) \equiv s^{\ell_{max}+1}(x, \ell_{max})$.

$\mathcal{A}'$ is finite. Given a state $(q, s) \in Q'$, using Lemma 18 (below), s has less than $|\mathcal{X}|^4$ vertices. The substitution is exactly defined by the shapes $s(x, \ell)$ for every vertices (x, ℓ). In each of these shapes, each register in $\mathcal{X}$ appears at most once and there are $|\mathcal{X}|^{\mathcal{O}(|\mathcal{X}|)}$ possible of such shapes. Thus, $|Q'| \leq |Q| \cdot |\mathcal{X}|^{\mathcal{O}(|\mathcal{X}|^5)}$ and $\mathcal{A}'$ is finite.

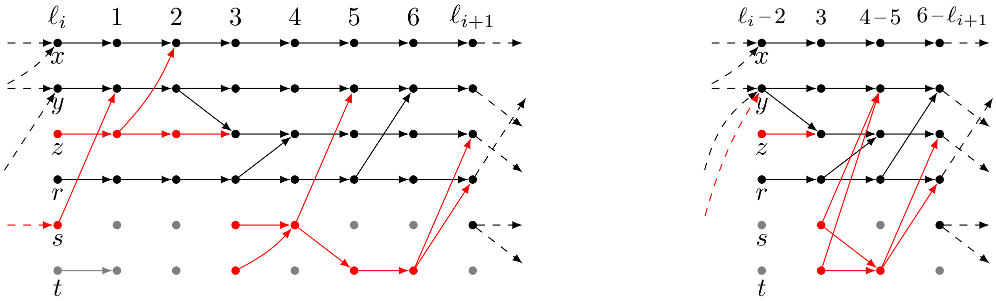


Figure 4 Merging of primarily copyless layers with the previous one.

▶ **Lemma 18.** *For all states (q, s) in $\mathcal{A}'$, the flow graph associated with s has less than $|\mathcal{X}|^2$ primary sources, $|\mathcal{X}|^3$ layers, and $|\mathcal{X}|^4$ vertices.*

Proof. Recall that there are $|\mathcal{X}|$ vertices per layer. In the flow graph, each vertex of the maximal layer defines exactly one minimal layer for its ancestors. All primary sources are on those layers and there are at most $|\mathcal{X}|$ such layers. Each layer contains at most $|\mathcal{X}|$ vertices, thus there are at most $|\mathcal{X}|^2$ primary sources in the flow graph. Also, this implies that the bound on the number of vertices is a direct consequence of the bound on the number of layers. We note $\ell_1, \ldots, \ell_k$ the layers where primary sources are, with $k \leq |\mathcal{X}|$ and $0 = \ell_1 < \ell_2 < \cdots < \ell_k$ (the first layer contains only primary sources). For $1 \leq i < k$, we now bound the number of layers separating ℓ_i from ℓ_{i+1}.

Recall that a copy vertex is a vertex with at least two out-going edges. A *primary copy vertex* is a copy vertex that is primary. A *bad layer* is a layer that is not primarily-copyless layer. Bad layers are exactly layers that remain after the removal of all primarily-copyless layers. Note that by definition, a bad layer contains at least one primary copy vertex. Also, as every primary vertex only reaches primary vertices and as the flow graph is $\diamond$-less, a primary copy vertex reaches primary vertices of the next layer that pairwise cannot flow in the same vertex while they all flow in the maximal layer.

Figure 4 depicts vertices of a flow graph between two layers ℓ_i and ℓ_{i+1} before removing primarily copyless layers on the left and after on the right. Primary vertices are represented in black, secondary vertices are represented in red, and vertices that cannot reach the last layer are represented in gray. Vertices (r, ℓ_i) and (s, ℓ_{i+1}) are primary sources. Between ℓ_i and ℓ_{i+1}, there are 3 primary copy vertices: $(y, 2)$, $(r, 3)$ and $(r, 5)$. Layers ℓ_i, 1 and 2 are merged, as well as layers 4 and 5, and also layers 6 and ℓ_{i+1}.

As the flow graph is $\diamond$-less, every primary vertex (x, ℓ_i) flows at most once in every primary vertex of layer ℓ_{i+1}. We note $n_{x,i}$ the number of vertices of layer ℓ_{i+1} reachable from (x, ℓ_i), we have that $n_{x,i} \leq |\mathcal{X}|$. Then, (x, ℓ_i) reaches at most $n_{x,i} - 1$ copy vertices between layers ℓ_i (included) and ℓ_{i+1} (excluded), all of which are primary. Since this holds for all primary vertices of layer ℓ_i, there are at most $|\mathcal{X}| \left(|\mathcal{X}| - 1\right)$ primary copy vertices between layers ℓ_i and ℓ_{i+1}. This directly implies the same bound for the number of (bad) layers. The argument still holds for (bad) layers between ℓ_k and the maximal layer ℓ_{max} of the flow graph. As a consequence, there are at most $|\mathcal{X}|^2 \left(|\mathcal{X}| - 1\right) \leq |\mathcal{X}|^3$ layers in s. ◀

$\mathcal{A}'$ is bounded-copy. We need some additional intuition on how secondary vertices flow. Consider a secondary vertex v that flows into vertices of the last layer, that are primary. Along the path from v to the last layer, there are secondary vertices, followed by primary vertices. The first primary vertex encountered is called a *target* of v: in the flow graph G of

Example 16, targets of $(y, 1)$ are $(x, 2)$ and $(z, 2)$. Notice that every target is a descendant of some primary source, by definition of primary vertices. A primary vertex is its own target. *Primary sources* of a vertex are all the sources of its targets, that are primary: again in the example G, primary sources of $(y, 2)$ are $(x, 0)$, $(y, 0)$ and $(z, 0)$.

We define a measure on the coefficients of all the shapes in a state of $\mathcal{A}'$ that bounds the number of times it can copy into other coefficients in the future. The measure of a coefficient κ appearing in the shape of a vertex $v = (x, \ell)$ of the shape substitution is defined as the tuple $\|\kappa\| = (\ell, p, c, \ell_\pi, g)$ where p is the number of primary sources in s restricted to the previous layers $\{0, \dots, \ell - 1\}$; $c = 1$ if the coefficient κ is an ω-coefficient, and $c = 0$ otherwise; ℓ_π is the maximal layer of a primary source of vertex v; g is the number of targets of vertex v. We can show that along the update of every transition of $\mathcal{A}'$, if a coefficient κ is used to update the value of another coefficient κ' then $\|\kappa\| \geq \|\kappa'\|$ (where tuples are ordered lexicographically). Moreover, if κ is used in at least two coefficients, then $\|\kappa\| > \|\kappa'\|$. We can bound the length of all decreasing sequences of tuples by $2|\mathcal{X}|^9$. When a coefficient is copied, it flows into at most $2|\mathcal{X}|$ coefficients. Thus, $\mathcal{A}'$ is $(2|\mathcal{X}|)^{2|\mathcal{X}|^9}$-copy.

5 From bounded-copy CRAs to copyless CRAs

The final step of our construction is to remove all copies from a bounded-copy CRA. This is achieved in a purely greedy manner: it suffices to have enough replicas of each register from the beginning, and then split the replicas evenly when the bounded-copy CRA performs copies.

▶ **Proposition 19.** *For every k-copy* CRA*, we can construct an equivalent copyless* CRA*.*

▶ Remark 20. Note that our definition of bounded-copy differs from the one of [4]. There, flow graphs are first trimmed with respect to the output vertex, and k-copy means that there are at most k copy vertices in the whole trimmed flow graph. In this sense, 0-copy implies $\diamond$-less in our setting. Therefore, removing copies is the core of their transformation from functional copyless (non-deterministic) streaming string transducers into copyless deterministic ones.

This ends the proof of Theorem 1. Actually, this last step in the proof extends easily to show that bounded-copy finitely-ambiguous CRAs can be effectively transformed into equivalent finitely-ambiguous CRAs. Applying then Theorem 1, we obtain

▶ **Corollary 21.** *Every bounded-copy finitely-ambiguous* NCRA *can be effectively transformed into an equivalent copyless* CRA*.*

6 Conclusion

We have shown a construction removing the finite-ambiguity of a copyless NCRA, with an application to the removal of regular look-aheads in copyless CRA-RLAs in arbitrary semirings. It can be shown that the result does no longer hold for linear-ambiguous copyless NCRAs using the example of [14, Theorem 2], therefore finite-ambiguity seems the weakest condition for which our result holds. Moreover, existing techniques of regular look-ahead removal for streaming string-to-tree transducers [3] cannot be used directly, as the addition of unary mappings update strictly increases the expressive power of copyless CRAs. Going back to the example of Figure 1, notice that the application of our construction from the finitely-ambiguous NCRA $\mathcal{A}_n$ yields a copyless CRA that is bigger than the alternative solution $\mathcal{A}_c$: it has more states, and more registers. As future works, we thus plan to study minimisation of copyless CRAs in the general setting of semirings. Our results work even for non-commutative semirings. However it heavily relies on the distributivity property of semirings, in order to normalise the terms into shapes. The transformation from a ◇-less CRA to a bounded-copy CRA also relies on the capability to multiply two registers (coefficients) together: this is thus unclear how to extend our approach to the so-called *additive* copyless CRAs where products in register updates must happen between a register and a constant.

References

1 Shaull Almagor, Michaël Cadilhac, Filip Mazowiecki, and Guillermo A. Pérez. Weak Cost Register Automata Are Still Powerful. In *Proceedings of the 22nd International Conference on Developments in Language Theory (DLT 2018)*, volume 11088 of *LNCS*, pages 83–95. Springer, 2018. doi:10.1007/978-3-319-98654-8_7.

2 Rajeev Alur and Loris D'antoni. Streaming Tree Transducers. *Journal of the ACM*, 64(5):1–55, August 2017. doi:10.1145/3092842.

3 Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular Functions and Cost Register Automata. In *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013)*, pages 13–22. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.65.

4 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular Transformations of Infinite Strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS 2012)*, pages 65–74. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.18.

5 Rajeev Alur and Mukund Raghothaman. Decision Problems for Additive Regular Functions. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming, Part II (ICALP 2013)*, volume 7966 of *LNCS*, pages 37–48. Springer, 2013. doi:10.1007/978-3-642-39212-2_7.

6 Marcella Anselmo. Two-Way Automata with Multiplicity. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP 1990)*, volume 443 of *LNCS*, pages 88–102. Springer, 1990.

7 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating Weighted Automata. In *Proceedings of the 17th International Conference on Fundamentals of computation theory (FCT'09)*, volume 5699 of *Lecture Notes in Computer Science*, pages 3–13, 2009.

8 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Transactions on Computational Logic*, 11(4), 2010. doi:10.1145/1805950.1805953.

9 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)*, pages 857–866. ACM, 2016. doi:10.1145/2933575.2934549.

10 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 2009.

11 Kosaburo Hashiguchi, Kenichi Ishiguro, and Shuji Jimbo. Decidability of The Equivalence Problem for Finitely Ambiguous Finance Automata. *International Journal of Algebra and Computation*, 12(3):445, 2002. doi:10.1142/S0218196702000845.

12 Peter Kostolányi and Filip Misún. Alternating weighted automata over commutative semirings. *Theoretical Computer Science*, 740:1–27, 2018. doi:10.1016/j.tcs.2018.05.003.

13 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)*, volume 41 of *LIPIcs*, pages 144–159. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.144.

14 Filip Mazowiecki and Cristian Riveros. Copyless cost-register automata: Structure, expressiveness, and closure properties. *Journal of Computer and System Sciences*, 100:1–29, March 2019. doi:10.1016/j.jcss.2018.07.002.

15 Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. The Design Principles of a Weighted Finite-State Transducer Library. *Theoretical Computer Science*, 231(1):17–32, 2000. doi:10.1016/S0304-3975(99)00014-6.

16 Azaria Paz. Some Aspects of Probabilistic Automata. *Information and Computation*, 9(1):26–60, 1966.

17 Michael O. Rabin. Probabilistic Automata. *Information and Control*, 6(3):230–245, 1963.

18 Marcel Paul Schützenberger. On the Definition of a Family of Automata. *Information and Control*, 4(2-3):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.

Aperiodic Weighted Automata and Weighted First-Order Logic

Manfred Droste

Institut für Informatik, Universität Leipzig, Germany
droste@informatik.uni-leipzig.de

Paul Gastin

LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
paul.gastin@ens-paris-saclay.fr

Abstract

By fundamental results of Schützenberger, McNaughton and Papert from the 1970s, the classes of first-order definable and aperiodic languages coincide. Here, we extend this equivalence to a quantitative setting. For this, weighted automata form a general and widely studied model. We define a suitable notion of a weighted first-order logic. Then we show that this weighted first-order logic and aperiodic polynomially ambiguous weighted automata have the same expressive power. Moreover, we obtain such equivalence results for suitable weighted sublogics and finitely ambiguous or unambiguous aperiodic weighted automata. Our results hold for general weight structures, including all semirings, average computations of costs, bounded lattices, and others.

2012 ACM Subject Classification Theory of computation → Quantitative automata; Theory of computation → Logic and verification

Keywords and phrases Weighted automata, weighted logic, aperiodic automata, first-order logic, unambiguous, finitely ambiguous, polynomially ambiguous

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.76

Related Version A full version of the paper is available at `https://arxiv.org/abs/1902.08149`

Funding *Manfred Droste*: Partly supported by a visiting professorship at ENS Paris-Saclay
Paul Gastin: Partly supported by the DFG Research Training Group QuantLA.

1 Introduction

Fundamental results of Schützenberger, McNaughton and Papert established that aperiodic, star-free and first-order definable languages, respectively, coincide [40, 32]. In this paper, we develop such an equivalence in a quantitative setting, i.e., for suitable notions of aperiodic weighted automata and weighted first-order logic.

Already Schützenberger [39] investigated weighted automata and characterized their behaviors as rational formal power series. Weighted automata can be viewed as classical finite automata in which the transitions are equipped with weights. These weights could model, e.g., the cost, reward or probability of executing a transition. The wide flexibility of this automaton model soon led to a wealth of extensions and applications, cf. [38, 28, 2, 36, 15]. Whereas traditionally weights are taken from a semiring, recently, motivated by practical examples, also average and discounted computations of weights were considered, cf. [8, 7].

In the boolean setting, the seminal Büchi-Elgot-Trakhtenbrot theorem [6, 21, 41] established the expressive equivalence of finite automata and monadic second-order logic (MSO). A weighted monadic second-order logic with the same expressive power as weighted automata was developed in [12, 13]. This led to various extensions to weighted automata and weighted logics on trees [19], infinite words [18], timed words [34], pictures [22], graphs [10], nested words [11], and data words [1], but also for more complicated weight structures including

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 76; pp. 76:1–76:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

average and discounted calculations [16] or multi-weights [17]. Recently, in [23], weighted MSO logic was revisited with a more structured syntax, called core-wMSO, and shown to be expressively equivalent to weighted automata, while permitting a uniform approach to semirings and these more complicated weight structures.

Here, we consider the first-order fragment wFO of this weighted logic. It extends the full classical boolean first-order logic quantitatively by adding weight constants and if-then-else applications, followed by a first-order (universal) product and then further if-then-else applications, finite sums, or first-order (existential) sums. We will show that its expressive power leads to aperiodic weighted automata which, moreover, are polynomially ambiguous. Natural subsets of connectives will correspond to unambiguous or finitely ambiguous aperiodic weighted automata. These various levels of ambiguity are well-known from classical automata theory [24, 42, 26, 25].

Following the approach of [23], we take an *arbitrary* set R of weights. A path in a weighted automaton over R then has the sequence of weights of its transitions as its value. The *abstract semantics* of the weighted automaton is defined as the function mapping each non-empty word to the multiset of weight sequences of the successful paths executing the given word. Correspondingly, we will define the abstract semantics of wFO sentences also as functions mapping non-empty words to multisets of sequences of weights. Our main result will be the following.

▶ **Theorem 1.** *Let Σ be an alphabet and* R *a set of weights. Then the following classes of weighted automata and weighted first-order logics are expressively equivalent:*

1. *Aperiodic polynomially ambiguous weighted automata (*wA*) and* wFO *sentences,*
2. *Aperiodic finitely ambiguous* wA *and* wFO *sentences without first-order sums,*
3. *Aperiodic unambiguous* wA *and* wFO *sentences without binary or first-order sums.*

Note that these characterizations hold without any restrictions on the weights. The above result applies not only to the abstract semantics. As immediate consequence, we obtain corresponding expressive equivalence results for classical weighted automata over arbitrary (even non-commutative) semirings, or with average or discounted calculations of weights, or bounded lattices as in multi-valued logics. All our constructions are effective. In fact, given a wFO sentence and deterministic aperiodic automata for its boolean subformulas, we can construct an equivalent aperiodic weighted automaton of exponential size. We give typical examples for our constructions. The class of arbitrary aperiodic weighted automata and its subclasses of polynomially resp. finitely ambiguous or unambiguous weighted automata form a proper hierarchy for each of the following semirings: natural numbers $\mathbb{N}_{+,\times}$, the max-plus-semiring $\mathbb{N}_{\max,+}$ and the min-plus semiring $\mathbb{N}_{\min,+}$ [14].

It should be noticed that standard constructions used to establish equivalence between automata and MSO logic cannot be applied. Indeed, starting from an automaton $\mathcal{A}$, one usually constructs an *existential* MSO sentence where the existential set quantifications are used to guess an accepting run and the easy first-order kernel is used to check that this guess indeed defines an accepting run. Here, we cannot use quantifications $\exists X$ over set variables X, or their weighted equivalent $\sum_X$. Instead, we take advantage of the fine structure of possible paths of polynomially ambiguous automata, namely the fact that it must be unambiguous on strongly connected components (SCC-unambiguous), as employed for different goals already in [24, 42]. We first give a new construction of a wFO sentence without sums starting from an aperiodic and unambiguous automaton. Then, we extend the construction to polynomially ambiguous aperiodic automata using first-order sums $\sum_x$ to guess positions where the run switches between the unambiguous SCCs. For part 2 of Theorem 1, we also prove that for each *aperiodic* finitely ambiguous weighted automaton we can construct finitely many *aperiodic* unambiguous weighted automata whose disjoint union has the same semantics.

Again, for the implication from weighted formulas to weighted automata, we cannot simply use standard constructions which crucially rely on the fact that functions defined by weighted automata are closed under morphic images. This was used to handle first-oder sums $\sum_x$ and second-order sums $\sum_X$, but also in the more involved proof for the first-order product $\prod_x$ applied to finitely valued weighted automata. But it is well-known that aperiodic languages are not closed under morphic images. Handling the first-order product $\prod_x$ requires a completely new and highly non-trivial proof preserving aperiodicity properties.

Detailed proofs and additional examples are given in the full version [14].

Related work. In [27], polynomially ambiguous, finitely ambiguous and unambiguous weighted automata (without assuming aperiodicity) over commutative semirings were shown to be expressively equivalent to suitable fragments of weighted monadic second order logic. This was further extended in [33] to cover polynomial degrees and weighted tree automata.

A hierarchy of these classes of weighted automata (again without assuming aperiodicity) over the max-plus semiring was described in [26]. As a consequence of pumping lemmas for weighted automata, a similar hierarchy was obtained in [30] for the min-plus semiring.

We note that in [13, 20], an equivalence result for full weighted first-order logic was given, but only for very particular classes of semirings or strong bimonoids as weight structures.

A characterization of the full weighted first-order logic with transitive closure by weighted pebble automata was obtained in [5]. An equivalence result for fragments of weighted first-order logic, weighted LTL and weighted counter-free automata over the max-plus semiring with discounting was given in [29].

2 Preliminaries

A non-deterministic automaton is a tuple $\mathcal{A} = (Q, \Sigma, \Delta)$ where Q is a finite set of states, Σ is a finite alphabet, and $\Delta \subseteq Q \times \Sigma \times Q$ is the set of transitions. The automaton $\mathcal{A}$ is complete if $\Delta(q, a) \neq \emptyset$ for all $q \in Q$ and $a \in \Sigma$. A run ρ of $\mathcal{A}$ is a nonempty sequence of transitions $\delta_1 = (p_1, a_1, q_1)$, $\delta_2 = (p_2, a_2, q_2)$, …, $\delta_n = (p_n, a_n, q_n)$ such that $q_i = p_{i+1}$ for all $1 \leq i < n$. We say that ρ is a run from state p_1 to state q_n and that ρ reads, or has label, the word $a_1 a_2 \cdots a_n \in \Sigma^+$. We denote by $\mathcal{L}(\mathcal{A}_{p,q}) \subseteq \Sigma^*$ the set of labels of runs of $\mathcal{A}$ from p to q. When $p = q$, we include the empty word ε in $\mathcal{L}(\mathcal{A}_{p,q})$ and say that ε labels the empty run from p to p.

An automaton with accepting conditions is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ where (Q, Σ, Δ) is a non-deterministic automaton, $I, F \subseteq Q$ are the sets of initial and final states respectively. The language defined by the automaton is $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_{I,F}) = \bigcup_{p \in I, q \in F} \mathcal{L}(\mathcal{A}_{p,q})$. Subsequently, we also consider automata with several accepting sets $F, G, \ldots$ so that the same automaton may define several languages $\mathcal{L}(\mathcal{A}_{I,F})$, $\mathcal{L}(\mathcal{A}_{I,G})$, …An automaton $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ is *deterministic* if $I = \{\iota\}$ is a singleton and the set Δ of transitions is a partial function: for all $(p, a) \in Q \times \Sigma$ there is at most one state $q \in Q$ such that $(p, a, q) \in \Delta$.

Next, we consider degrees of ambiguity of automata. A run in an automaton is *successful*, if it leads from an initial to a final state. The automaton $\mathcal{A}$ is called *polynomially ambiguous* if there is a polynomial p such that for each $w \in \Sigma^+$ the number of successful paths in $\mathcal{A}$ for w is at most $p(|w|)$. Then, $\mathcal{A}$ is *finitely ambiguous* if p can be taken to be a constant. Further, for an integer $k \geq 1$, $\mathcal{A}$ is k-ambiguous if $p = k$, and *unambiguous* means 1-ambiguous. Notice that k-ambiguous implies $(k+1)$-ambiguous. An automaton $\mathcal{A}$ is at most *exponentially ambiguous*.

A non-deterministic automaton $\mathcal{A} = (Q, \Sigma, \Delta)$ is *aperiodic* if there exists an integer $m \geq 1$, called aperiodicity index, such that for all states $p, q \in Q$ and all words $u \in \Sigma^+$, we have $u^m \in \mathcal{L}(\mathcal{A}_{p,q})$ iff $u^{m+1} \in \mathcal{L}(\mathcal{A}_{p,q})$. In other words, the non-deterministic automaton $\mathcal{A}$ is aperiodic iff its transition monoid $\mathsf{Tr}(\mathcal{A})$ is aperiodic. It is well-known that aperiodic languages coincide with first-order definable languages, cf. [40, 32, 9].

The syntax of first-order logic is given in Section 4 (FO). The semantics is defined by structural induction on the formula and requires an interpretation of the free variables. Let $\mathcal{V} = \{y_1, \dots, y_n\}$ be a finite set of first-order variables. Given a nonempty word $u \in \Sigma^+$, we let $\mathsf{pos}(u) = \{1, \dots, |u|\}$ be the set of positions of u. A valuation or interpretation is a map $\sigma \colon \mathcal{V} \to \mathsf{pos}(u)$ assigning positions of u to variables in $\mathcal{V}$. For a first-order formula φ having free variables contained in $\mathcal{V}$, we write $u, \sigma \models \varphi$ when the word u satisfies φ under the interpretation defined by σ. When φ is a *sentence*, the valuation σ is not needed and we simply write $u \models \varphi$.

We extend the classical semantics by defining when the empty word ε satisfies a sentence. We have $\varepsilon \models \top$ and if $\forall x \psi$ is a sentence then $\varepsilon \models \forall x \psi$. The semantics $\varepsilon \models \varphi$ is extended to all sentences φ since they are boolean combinations of the basic cases above. Notice that if φ has free variables then $\varepsilon \models \varphi$ is not defined. When φ is a sentence we denote by $\mathcal{L}(\varphi) \subseteq \Sigma^*$ the set of words satisfying φ. Notice that $\mathcal{L}(\forall x \bot) = \{\varepsilon\}$ where $\bot = \neg\top$.

▶ **Theorem 2** ([40, 32, 9]). *Let $\mathcal{A}$ be an aperiodic non-deterministic automaton. For all states p, q of $\mathcal{A}$ we can construct a first-order sentence $\varphi_{p,q}$ such that $\mathcal{L}(\mathcal{A}_{p,q}) = \mathcal{L}(\varphi_{p,q})$.*

For the converse of Theorem 2, we need a stronger statement to deal with formulas having free variables. As usual, we encode a pair (u, σ) where $u \in \Sigma^+$ is a nonempty word and $\sigma \colon \mathcal{V} \to \mathsf{pos}(u)$ is a valuation by a word $\overline{u}$ over the extended alphabet $\Sigma_{\mathcal{V}} = \Sigma \times \{0,1\}^{\mathcal{V}}$. A word $\overline{u}$ over $\Sigma_{\mathcal{V}}$ is a *valid* encoding if for each variable $y \in \mathcal{V}$, its projection on the y-component belongs to 0^*10^*. Throughout the paper, we identify a valid word $\overline{u}$ with its encoded pair (u, σ).

▶ **Theorem 3** ([40, 32, 9]). *For each FO-formula φ having free variables contained in $\mathcal{V}$, we can build a deterministic, complete and aperiodic automaton $\mathcal{A}_{\varphi,\mathcal{V}} = (Q, \Sigma_{\mathcal{V}}, \Delta, \iota, F, G)$ over the extended alphabet $\Sigma_{\mathcal{V}}$ such that for all words $\overline{u} \in \Sigma_{\mathcal{V}}^+$ we have:*

- *$\Delta(\iota, \overline{u}) \in F$ iff $\overline{u}$ is a valid encoding of a pair (u, σ) with $(u, \sigma) \models \varphi$,*
- *$\Delta(\iota, \overline{u}) \in G$ iff $\overline{u}$ is a valid encoding of a pair (u, σ) with $(u, \sigma) \models \neg\varphi$,*
- *$\Delta(\iota, \overline{u}) \notin F \cup G$ otherwise, i.e., iff $\overline{u}$ is not a valid encoding of a pair (u, σ).*

Given $u \in \Sigma^+$ and integers k, ℓ, we denote by $u[k, \ell]$ the factor of u between positions k and ℓ. By convention $u[k, \ell] = \varepsilon$ is the empty word when $\ell < k$ or $\ell = 0$ or $k > |u|$.

We will apply the equivalence of Theorem 2 to prefixes, infixes or suffixes of words. Towards this, we use the classical *relativization* of sentences. Let φ be a first-order sentence and let $x, y \in \mathcal{V}$ be first-order variables. We define below the relativizations $\varphi^{<x}$, $\varphi^{(x,y)}$ and $\varphi^{>y}$ so that for all words $u \in \Sigma^+$, and all positions $i, j \in \mathsf{pos}(u) = \{1, \dots, |u|\}$ we have

$$\begin{array}{rcr} u, x \mapsto i \models \varphi^{<x} & \text{iff} & u[1, i-1] \models \varphi \\ u, x \mapsto i, y \mapsto j \models \varphi^{(x,y)} & \text{iff} & u[i+1, j-1] \models \varphi \\ u, x \mapsto j \models \varphi^{>x} & \text{iff} & u[j+1, |u|] \models \varphi \end{array}$$

Notice that, when $i = 1$ or $j \leq i + 1$ or $j = |u|$, the relativization is on the empty word, this is why we had to define when $\varepsilon \models \psi$ for sentences ψ. The relativization is defined by

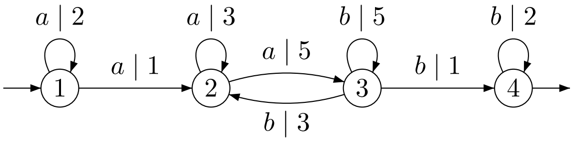


Figure 1 A weighted automaton, which is both aperiodic and polynomially ambiguous.

structural induction on the formulas as follows:

$$\begin{array}{lll} \top^{<x} = \top & (P_a(z))^{<x} = P_a(z) & (y \leq z)^{<x} = (y \leq z) \\ (\neg\psi)^{<x} = \neg(\psi^{<x}) & (\psi_1 \wedge \psi_2)^{<x} = \psi_1^{<x} \wedge \psi_2^{<x} & (\forall z \psi)^{<x} = \forall z (z < x \implies \psi^{<x}) \end{array}$$

The relativizations $\varphi^{(x,y)}$ and $\varphi^{>x}$ are defined similarly. Notice that when φ is a sentence, i.e., a boolean combination of formulas of the form $\top$ or $\forall z \psi$, then the above equivalences hold even when $i = 1$ for $\varphi^{<x}$, or when $i = |u|$ for $\varphi^{>x}$, or when $j \leq i + 1$ for $\varphi^{(x,y)}$.

3 Weighted Automata

Given a set X, we let $\mathbb{N}\langle X \rangle$ be the collection of all finite multisets over X, i.e., all functions $f \colon X \to \mathbb{N}$ such that $f(x) \neq 0$ only for finitely many $x \in X$. The union $f \uplus g$ of two multisets $f, g \in \mathbb{N}\langle X \rangle$ is defined by pointwise addition of functions: $(f \uplus g)(x) = f(x) + g(x)$ for $x \in X$.

For a set R of weights, an R-weighted automaton over Σ is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, \mathsf{wt})$ where (Q, Σ, Δ) is a non-deterministic automaton and $\mathsf{wt} \colon \Delta \to \mathsf{R}$ assigns a weight to every transition. The weight sequence of a run $\rho = \delta_1\delta_2 \cdots \delta_n$ is $\mathsf{wt}(\rho) = \mathsf{wt}(\delta_1)\mathsf{wt}(\delta_2) \cdots \mathsf{wt}(\delta_n) \in \mathsf{R}^+$. The *abstract semantics* of $\mathcal{A}$ from state p to state q is the map $\{\!|\mathcal{A}_{p,q}|\!\} \colon \Sigma^+ \to \mathbb{N}\langle \mathsf{R}^+ \rangle$ which assigns to a word $u \in \Sigma^+$ the multiset of weight sequences of runs from p to q with label u:

$$\{\!|\mathcal{A}_{p,q}|\!\}(u) = \{\!\{\mathsf{wt}(\rho) \mid \rho \text{ is a run from } p \text{ to } q \text{ with label } u\}\!\} \, .$$

Notice that $\{\!|\mathcal{A}_{p,q}|\!\}(u) = \emptyset$ is the empty multiset when there are no runs of $\mathcal{A}$ from p to q with label u, i.e., when $u \notin \mathcal{L}(\mathcal{A}_{p,q})$. When we consider a weighted automaton $\mathcal{A} = (Q, \Sigma, \Delta, \mathsf{wt}, I, F)$ with initial and final sets of states, for all $u \in \Sigma^+$ the semantics $\{\!|\mathcal{A}|\!\}$ is defined as the multiset union: $\{\!|\mathcal{A}|\!\}(u) = \biguplus_{p \in I, q \in F} \{\!|\mathcal{A}_{p,q}|\!\}(u)$. Hence, $\{\!|\mathcal{A}|\!\}$ assigns to every word $u \in \Sigma^+$ the multiset of all weight sequences of accepting runs of $\mathcal{A}$ reading u. The support of $\mathcal{A}$ is the set of words $u \in \Sigma^+$ such that $\{\!|\mathcal{A}|\!\}(u) \neq \emptyset$, i.e., $\mathsf{supp}(\mathcal{A}) = \mathcal{L}(\mathcal{A})$.

For instance, consider the weighted automaton $\mathcal{A}$ of Figure 1. We have $\mathsf{supp}(\mathcal{A}) = a^+a(a+b)^*b^+$. Consider $w = a^m(ba)^nb^p$ with $m > 1$ and $p > 0$. We have $w \in \mathsf{supp}(\mathcal{A})$ and $\{\!|\mathcal{A}|\!\}(w) = \{\!\{2^{k-1} \cdot 1 \cdot 3^{m-k-1} \cdot 5 \cdot (3 \cdot 5)^n \cdot 5^{\ell-1} \cdot 1 \cdot 2^{p-\ell} \mid 1 \leq k < m \text{ and } 1 \leq \ell \leq p\}\!\}$.

A concrete semantics over semirings, or valuation monoids, or valuation structures can be obtained from the abstract semantics defined above by applying the suitable aggregation operator $\mathsf{aggr} \colon \mathbb{N}\langle \mathsf{R}^+ \rangle \to S$ as explained in [23], see also [14]. For the natural semiring $(\mathbb{N}, +, \times, 0, 1)$, the sum-product aggregation operator $\mathsf{aggr}_{\mathsf{sp}}(f)$ gives the sum over all sequences $s_1s_2 \cdots s_k$ in the multiset f of the products $s_1 \times s_2 \times \cdots \times s_k$ in $\mathbb{N}$. We continue the example with the automaton $\mathcal{A}$ of Figure 1 and the word $w = a^m(ba)^nb^p$ with $m > 1$ and $p > 0$. The concrete semantics is given by

$$[\![\mathcal{A}]\!](w) = \mathsf{aggr}_{\mathsf{sp}}(\{\!|\mathcal{A}|\!\}(w)) = \sum_{1 \leq k < m} \sum_{1 \leq \ell \leq p} 2^{k-1+p-\ell} 3^{m-k-1+n} 5^{n+\ell} \, .$$

In further examples, we also use the max-plus semiring $\mathbb{N}_{\max,+} = (\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ and the min-plus semiring $\mathbb{N}_{\min,+} = (\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$.

Now, we investigate finitely ambiguous weighted automata. It was shown in [26] that over the max-plus semiring $\mathbb{N}_{\max,+}$ they are expressively equivalent to finite disjoint unions of unambiguous weighted automata. Moreover, it was proved in [37] that a K-valued rational transducer can be decomposed into K unambiguous transducers. In particular this implies that a K-ambiguous weighted automaton can be decomposed into K unambiguous weighted automata. We can prove that the same holds for aperiodic weighted automata [14].

▶ **Theorem 4.** *Let $K \geq 1$. Given an aperiodic K-ambiguous weighted automaton $\mathcal{A}$, we can construct aperiodic unambiguous weighted automata $\mathcal{B}_1, \dots, \mathcal{B}_K$ such that $\{\!|\mathcal{A}|\!\} = \{\!|\mathcal{B}_1 \uplus \cdots \uplus \mathcal{B}_K|\!\} = \{\!|\mathcal{B}_1|\!\} \uplus \cdots \uplus \{\!|\mathcal{B}_K|\!\}$.*

Our proof is based on lexicographic ordering of runs. The proof of [37] uses lexicographic coverings. It would be interesting to see whether this proof also preserves aperiodicity and to compare the complexity of the constructions.

4 Weighted First-Order Logic

In this section, we define the syntax and semantics of our weighted first-order logic. In [12, 13], weighted MSO used the classical syntax of MSO logic; only the semantics over a semiring was changed to use sums for disjunction and existential quantifications, and products for conjunctions and universal quantifications. The possibility to express boolean properties in wMSO was obtained via so-called unambiguous formulae. To improve readability, a more structured syntax was later used [3, 16, 27], separating a boolean MSO layer with classical boolean semantics from the higher level of weighted formulas using *products* ($\prod_X$, $\prod_x$ corresponding to $\forall X$, $\forall x$) and sums ($\sum_X$, $\sum_x$ corresponding to $\exists X$, $\exists x$) with quantitative semantics. As shown in [12, 13], in general, to retain equivalence with weighted automata, wMSO has to be restricted. Products $\prod_X$ over set variables are disallowed, and first-order products $\prod_x$ must be restricted to finitely valued series where the pre-image of each value is recognizable. This basically means that first-order products cannot be nested or applied after first-order or second-order sums $\sum_x$ or $\sum_X$. This motivated the equivalent and even more structured syntax of core-wMSO introduced in [23].

As in Section 3, we consider a set R of weights. The syntax of wFO is obtained from core-wMSO by removing set variables, set quantifications and set sums. In addition to the classical boolean first-order logic (FO), it has two weighted layers. Step formulas defined in (step-wFO) consist of constants and if-then-else applications, where the conditions are formulated in boolean first-order logic. Finally, wFO builds on this by performing products of step formulas and then applying if-then-else, finite sums, or existential sums.

$$\varphi ::= \top \mid P_a(x) \mid x \leq y \mid \neg\varphi \mid \varphi \wedge \varphi \mid \forall x \varphi \tag{FO}$$

$$\Psi ::= r \mid \varphi \,?\, \Psi : \Psi \tag{step-wFO}$$

$$\Phi ::= \mathbf{0} \mid \textstyle\prod_x \Psi \mid \varphi \,?\, \Phi : \Phi \mid \Phi + \Phi \mid \textstyle\sum_x \Phi \tag{wFO}$$

with $a \in \Sigma$, $r \in \mathsf{R}$ and x, y first-order variables.

The semantics of step-wFO formulas is defined inductively. As above, let $u \in \Sigma^+$ be a nonempty word and $\sigma \colon \mathcal{V} \to \mathsf{pos}(u) = \{1, \dots, |u|\}$ be a valuation. For step-wFO formulas whose free variables are contained in $\mathcal{V}$, we define the $\mathcal{V}$-semantics as

$$[\![r]\!]_{\mathcal{V}}(u,\sigma) = r \qquad [\![\varphi \,?\, \Psi_1 : \Psi_2]\!]_{\mathcal{V}}(u,\sigma) = \begin{cases} [\![\Psi_1]\!]_{\mathcal{V}}(u,\sigma) & \text{if } u,\sigma \models \varphi \\ [\![\Psi_2]\!]_{\mathcal{V}}(u,\sigma) & \text{otherwise.} \end{cases}$$

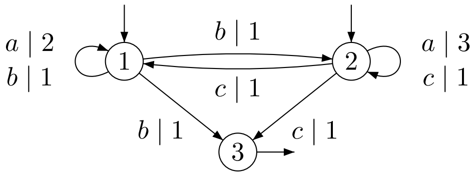


Figure 2 A weighted automaton, which is both aperiodic and unambiguous.

Notice that the semantics of a step-wFO formula is always a single weight from R.

For wFO formulas Φ whose free variables are contained in $\mathcal{V}$, we define the $\mathcal{V}$-semantics $\{\!|\Phi|\!\}_{\mathcal{V}}\colon \Sigma_{\mathcal{V}}^{+} \to \mathbb{N}\langle \mathsf{R}^{+}\rangle$. First, we let $\{\!|\Phi|\!\}_{\mathcal{V}}(\overline{u}) = \emptyset$ be the empty multiset when $\overline{u} \in \Sigma_{\mathcal{V}}^{+}$ is not a valid encoding of a pair (u,σ). Assume now that $\overline{u} = (u,\sigma)$ is a valid encoding of a nonempty word $u \in \Sigma^{+}$ and a valuation $\sigma\colon \mathcal{V} \to \mathsf{pos}(u)$. The semantics of wFO formulas is also defined inductively: $\{\!|\mathbf{0}|\!\}_{\mathcal{V}}(u,\sigma) = \emptyset$ is the empty multiset, and

$$\{\!|\textstyle\prod_x \Psi|\!\}_{\mathcal{V}}(u,\sigma) = \{\!\{r_1 r_2 \cdots r_{|u|}\}\!\} \text{ where } r_i = [\![\Psi]\!]_{\mathcal{V}\cup\{x\}}(u,\sigma[x\mapsto i]) \text{ for } 1 \le i \le |u|$$

$$\{\!|\varphi \mathbin{?} \Phi_1 : \Phi_2|\!\}_{\mathcal{V}}(u,\sigma) = \begin{cases} \{\!|\Phi_1|\!\}_{\mathcal{V}}(u,\sigma) & \text{if } u,\sigma \models \varphi \\ \{\!|\Phi_2|\!\}_{\mathcal{V}}(u,\sigma) & \text{otherwise} \end{cases}$$

$$\{\!|\Phi_1 + \Phi_2|\!\}_{\mathcal{V}}(u,\sigma) = \{\!|\Phi_1|\!\}_{\mathcal{V}}(u,\sigma) \uplus \{\!|\Phi_2|\!\}_{\mathcal{V}}(u,\sigma)$$

$$\{\!|\textstyle\sum_x \Phi|\!\}_{\mathcal{V}}(u,\sigma) = \biguplus_{i\in\mathsf{pos}(u)} \{\!|\Phi|\!\}_{\mathcal{V}\cup\{x\}}(u,\sigma[x\mapsto i])\,.$$

The semantics of the product (first line), is a singleton multiset which consists of a weight sequence whose length is $|u|$. We deduce that all weight sequences in a multiset $\{\!|\Phi|\!\}_{\mathcal{V}}(u,\sigma)$ have the same length and $\{\!|\Phi|\!\}_{\mathcal{V}}(u,\sigma) \in \mathbb{N}\langle \mathsf{R}^{|u|}\rangle$. We simply write $[\![\Psi]\!]$ and $\{\!|\Phi|\!\}$ when the set $\mathcal{V}$ of variables is clear from the context.

As explained in Section 3, applying an aggregation function allows to recover the semantics $[\![\Phi]\!]$ over semirings such as $\mathbb{N}_{+,\times}$, $\mathbb{N}_{\max,+}$, etc. For instance, consider the function $f\colon \{a,b\}^{+} \to \mathbb{N}$ which assign to a word $w \in \{a,b\}^{+}$ the length of the maximal a-block, i.e., $f(w) = n$ if a^n is a factor of w but a^{n+1} is not. Over $\mathbb{N}_{\max,+}$, we have $f = [\![\Phi]\!]$ where

$$\Phi = \textstyle\sum_{y,z} (\forall u\,(y \le u \le z) \to P_a(u)) \mathbin{?} (\prod_x (y \le x \le z) \mathbin{?} 1 : 0) : (\prod_x 0)\,.$$

We refer to [13] for further examples of quantitative specifications in weighted logic.

5 From Weighted Automata to Weighted FO

We say that a non-deterministic automaton $\mathcal{A} = (Q,\Sigma,\Delta)$ is *unambiguous from state p to state q* if for all words $u \in \Sigma^{+}$, there is at most one run of $\mathcal{A}$ from p to q with label u.

▶ **Theorem 5.** *Let $\mathcal{A}$ be an aperiodic weighted automaton which is unambiguous from p to q. We can construct a* wFO *sentence $\Phi_{p,q} = \varphi_{p,q} \mathbin{?} \prod_x \Psi_{p,q} : \mathbf{0}$ where $\varphi_{p,q}$ is a first-order sentence and $\Psi_{p,q}(x)$ is a* step-wFO *formula with a single free variable x such that $\{\!|\mathcal{A}_{p,q}|\!\} = \{\!|\Phi_{p,q}|\!\}$.*

Before proving Theorem 5, we start with an example. The automaton $\mathcal{A}$ of Figure 2 is unambiguous and it accepts the language $\mathcal{L}(\mathcal{A}) = (a^*b + a^*c)^{+} = (a+b+c)^*(b+c)$. We define a wFO sentence $\Phi_{1,3} = \varphi_{1,3} \mathbin{?} \prod_x \Psi_{1,3}(x) : \mathbf{0}$ as follows. The FO sentence $\varphi_{1,3}$ checks that $\mathcal{A}$ has a run from state 1 to state 3 on the input word w, i.e., that $w \in a^*b(a^*b + a^*c)^*$:

$$\varphi_{1,3} = \exists y\,(P_b(y) \wedge \forall z\,(z < y \implies P_a(z))) \wedge \exists y\,(\neg P_a(y) \wedge \forall z\,(z \le y))$$

When this is the case, the step-wFO formula $\Psi_{1,3}(x)$ computes the weight of the transition taken at a position x in the input word:

$$\Psi_{1,3}(x) = (P_b(x) \vee P_c(x))\,?\,1 : \exists y\,(x < y \wedge P_b(y) \wedge \forall z\,(x < z < y \implies P_a(z)))\,?\,2 : 3\,.$$

Notice that the same formula $\Psi = \Psi_{2,3} = \Psi_{1,3}$ also allows to compute the sequence of weights for the accepting runs starting in state 2. Therefore, $\mathcal{A}$ is equivalent to the wFO sentence

$$\Phi = \exists y\,(\neg P_a(y) \wedge \forall z\,(z \leq y))\,?\,\prod_x \Psi(x) : \mathbf{0}\,.$$

Proof of Theorem 5. Let $\mathcal{A} = (Q, \Sigma, \Delta, \mathsf{wt})$ be the aperiodic weighted automaton. By Theorem 2, for every pair of states $r, s \in Q$ there is a first-order sentence $\varphi_{r,s}$ such that $\mathcal{L}(\mathcal{A}_{r,s}) = \mathcal{L}(\varphi_{r,s})$. This gives in particular the first-order sentence $\varphi_{p,q}$ which is used in $\Phi_{p,q}$.

▷ Claim 6. We can construct a step-wFO formula $\Psi_{p,q}(x)$ such that for each word $u \in \mathcal{L}(\mathcal{A}_{p,q})$ and each position $1 \leq i \leq |u|$ in the word u, we have $[\![\Psi_{p,q}]\!](u, x \mapsto i) = \mathsf{wt}(\delta)$ where δ is the ith transition of the unique run ρ of $\mathcal{A}$ from p to q with label u.

Before proving this claim, let us show how we can deduce the statement of Theorem 5. Clearly, if a word $u \in \Sigma^+$ is not in $\mathcal{L}(\mathcal{A}_{p,q})$ then we have $\{\!|\mathcal{A}_{p,q}|\!\}(u) = \emptyset = \{\!|\Phi_{p,q}|\!\}(u)$. Consider now a word $u = a_1 a_2 \cdots a_n \in \mathcal{L}(\mathcal{A}_{p,q})$ and the unique run $\rho = \delta_1 \delta_2 \cdots \delta_n$ of $\mathcal{A}$ from p to q with label u. We have $\{\!|\mathcal{A}_{p,q}|\!\}(u) = \{\{\mathsf{wt}(\delta_1)\mathsf{wt}(\delta_2) \cdots \mathsf{wt}(\delta_n)\}\} = \{\!|\prod_x \Psi_{p,q}|\!\}(u)$ where the second equality follows from Claim 6. We deduce that $\{\!|\mathcal{A}_{p,q}|\!\} = \{\!|\Phi_{p,q}|\!\}$.

We turn now to the proof of Claim 6. Let $\delta = (r, a, s) \in \Delta$ be a transition of $\mathcal{A}$. We define the FO-formula with one free variable $\varphi_\delta(x) = \varphi_{p,r}^{<x} \wedge P_a(x) \wedge \varphi_{s,q}^{>x}$.

▷ Claim 7. For each word $u \in \Sigma^+$ and position $1 \leq i \leq |u|$, we have $u, x \mapsto i \models \varphi_\delta$ iff $u \in \mathcal{L}(\mathcal{A}_{p,q})$ and δ is the ith transition of the unique run of $\mathcal{A}$ from p to q with label u.

Indeed, assume that $u, x \mapsto i \models \varphi_\delta$. Then, $u[1, i-1] \models \varphi_{p,r}$ and there is a run ρ' of $\mathcal{A}$ from p to r with label $u[1, i-1]$. Notice that if $i = 1$ then $p = r$ and ρ' is the empty run. Similarly, from $u[i+1, |u|] \models \varphi_{s,q}$ we deduce that there is a run ρ'' of $\mathcal{A}$ from s to q with label $u[i+1, |u|]$. Finally, $u, x \mapsto i \models P_a(x)$ means that the ith letter of u is a. We deduce that $\rho = \rho'\delta\rho''$ is a run of $\mathcal{A}$ from p to q with label u, hence $u \in \mathcal{L}(\mathcal{A}_{p,q})$. Moreover, ρ is the unique such run since $\mathcal{A}$ is unambiguous from p to q. Now, δ is the ith transition of ρ, which concludes one direction. Conversely, assume that $u \in \mathcal{L}(\mathcal{A}_{p,q})$ and δ is the ith transition of the unique run of $\mathcal{A}$ from p to q with label u. Then, $u[1, i-1] \models \varphi_{p,r}$, $u[i+1, |u|] \models \varphi_{s,q}$, and the ith letter of u is a. Therefore, $u, x \mapsto i \models \varphi_\delta$. This concludes the proof of Claim 7.

Now, choose an arbitrary enumeration $\delta^1, \delta^2, \ldots, \delta^k$ of the transitions in Δ and define the step-wFO formula with one free variable

$$\Psi_{p,q}(x) = \varphi_{\delta^1}(x)\,?\,\mathsf{wt}(\delta^1) : \varphi_{\delta^2}(x)\,?\,\mathsf{wt}(\delta^2) : \;\cdots\; \varphi_{\delta^k}(x)\,?\,\mathsf{wt}(\delta^k) : \mathsf{wt}(\delta^k)\,.$$

We show that this formula satisfies the property of Claim 6. Consider a word $u \in \mathcal{L}(\mathcal{A}_{p,q})$ and a position $1 \leq i \leq |u|$. Let δ be the ith transition of the unique run of $\mathcal{A}$ from p to q with label u. By Claim 7, we have $u, x \mapsto i \models \varphi_{\delta^j}$ iff $\delta^j = \delta$. Therefore, $[\![\Psi_{p,q}]\!](u, x \mapsto i) = \mathsf{wt}(\delta)$, which concludes the proof of Claim 6. ◀

▶ **Corollary 8.**

1. *Let $\mathcal{A}$ be an aperiodic and unambiguous weighted automaton. We can construct a* wFO *sentence Φ which does not use any $\sum_x$ operator or $+$ operator, and such that $\{\!|\mathcal{A}|\!\} = \{\!|\Phi|\!\}$.*
2. *Let $\mathcal{A}$ be an aperiodic and finitely ambiguous weighted automaton. We can construct a* wFO *sentence Φ which does not use any $\sum_x$ operator, and such that $\{\!|\mathcal{A}|\!\} = \{\!|\Phi|\!\}$.*

Let $\mathcal{A} = (Q, \Sigma, \Delta)$ be a non-deterministic automaton. Two states $p, q \in Q$ *are in the same strongly connected component* (SCC), denoted $p \approx q$, if $p = q$ or there exist a run of $\mathcal{A}$ from p to q and also a run of $\mathcal{A}$ from q to p. Note that $\approx$ is an equivalence relation on Q. We denote by $[p]$ the strongly connected component of p, i.e., the equivalence class of p under $\approx$.

The automaton $\mathcal{A}$ is *SCC-unambiguous* if it is unambiguous on each strongly connected component, i.e., $\mathcal{A}$ is unambiguous from p to q for all p, q such that $p \approx q$. A trimmed (all states are reachable and co-reachable) and unambiguous automaton is SCC-unambiguous.

For instance, the automaton $\mathcal{A}$ of Figure 1 has three strongly connected components: $\{1\}$, $\{2, 3\}$ and $\{4\}$. It is not unambiguous from 1 to 4, but it is SCC-unambiguous.

▶ **Proposition 9** ([35, 24] and [42] Thm 4.1)**.** *Let $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ be a trimmed non-deterministic automaton. Then $\mathcal{A}$ is polynomially ambiguous iff $\mathcal{A}$ is SCC-unambiguous.*

▶ **Theorem 10.** *Let $\mathcal{A}$ be an aperiodic weighted automaton which is SCC-unambiguous. For each pair of states p and q, we can construct a* wFO *sentence $\Phi_{p,q}$ such that $\{\!|\mathcal{A}_{p,q}|\!\} = \{\!|\Phi_{p,q}|\!\}$. Moreover, we can construct a* wFO *sentence Φ such that $\{\!|\mathcal{A}|\!\} = \{\!|\Phi|\!\}$.*

First, we give for the weighted automaton $\mathcal{A}$ of Figure 1 the equivalent wFO formula $\Phi_{1,4} = \sum_{y_1}\sum_{y_2}\varphi(y_1, y_2) \,?\, \prod_x \Psi(x, y_1, y_2) : \mathbf{0}$ where φ and Ψ are defined below. When reading a word $w \in \mathsf{supp}(\mathcal{A})$, the automaton makes two non-deterministic choices corresponding to the positions y_1 and y_2 at which the transitions *switching* between the strongly connected components are taken, i.e., transition from state 1 to state 2 is taken at position y_1, and transition from 3 to 4 is taken at position y_2. Since the automaton is SCC-unambiguous, given the input word and these two positions, the run is uniquely determined. Formula $\varphi(y_1, y_2)$ states that it is possible to take the switching transitions at positions y_1 and y_2:

$$\varphi(y_1, y_2) = y_1 < y_2 \wedge \forall z\,(z \le y_1 \to P_a(z)) \wedge P_a(y_1 + 1) \wedge \forall z\,(y_2 \le z \to P_b(z))\,.$$

When this is the case, the step-wFO formula $\Psi(x, y_1, y_2)$ computes the weight of the transition taken at a position x in the input word:

$$\Psi(x, y_1, y_2) = (x < y_1 \vee y_2 < x) \,?\, 2 : (x = y_1 \vee x = y_2) \,?\, 1 : P_a(x + 1) \,?\, 3 : 5\,.$$

With these definitions, we obtain $\{\!|\mathcal{A}|\!\} = \{\!|\Phi_{1,4}|\!\}$. The proof of Theorem 10 is in [14]. Intuitively, a run from p to q reading a word $w \in \Sigma^+$ uses a sequence of transitions switching between connected components of $\mathcal{A}$. The positions where these switches are taken can be described by a sequence of $\sum_y$-operators. Since there are only finitely many sequences of switching transitions, they can be described by a finite sum of wFO sentences.

6 From Weighted FO to Weighted Automata

Let $\mathcal{A} = (Q, \Sigma, \Delta)$ and $\mathcal{A}' = (Q', \Sigma, \Delta')$ be two non-deterministic automata over Σ. Assuming that $Q \cap Q' = \emptyset$, we define their disjoint union as $\mathcal{A} \uplus \mathcal{A}' = (Q \uplus Q', \Sigma, \Delta \uplus \Delta')$ and their product as $\mathcal{A} \times \mathcal{A}' = (Q \times Q', \Sigma, \Delta'')$ where $\Delta'' = \{((p, p'), a, (q, q')) \mid (p, a, p') \in \Delta \wedge (p', a, q') \in \Delta'\}$.

▶ **Lemma 11.** *The following holds.*

1. *If $\mathcal{A}$ and $\mathcal{A}'$ are aperiodic, then $\mathcal{A} \uplus \mathcal{A}'$ and $\mathcal{A} \times \mathcal{A}'$ are also aperiodic.*
2. *If $\mathcal{A}$ and $\mathcal{A}'$ are SCC-unambiguous, then $\mathcal{A} \uplus \mathcal{A}'$ and $\mathcal{A} \times \mathcal{A}'$ are also SCC-unambiguous.*

Now let φ be an FO-formula with free variables contained in the finite set $\mathcal{V}$, and let $\mathcal{A}_{\varphi,\mathcal{V}} = (Q, \Sigma_{\mathcal{V}}, \Delta, \iota, F, G)$ be the deterministic, complete, trim and aperiodic automaton given by Theorem 3. For $i = 1, 2$, let $\mathcal{A}_i = (Q_i, \Sigma_{\mathcal{V}}, \Delta_i, \mathsf{wt}_i, I_i, F_i)$ be two weighted automata over $\Sigma_{\mathcal{V}}$ with $Q_1 \cap Q_2 = \emptyset$. Define the weighted automaton $\mathcal{A}' = (Q', \Sigma_{\mathcal{V}}, \Delta', \mathsf{wt}', I', F')$ by

- $Q' = Q \times Q_1 \uplus Q \times Q_2$, $I' = \{\iota\} \times I_1 \uplus \{\iota\} \times I_2$, $F' = F \times F_1 \uplus G \times F_2$,
- $\Delta' = \{\big((p,p'),a,(q,q')\big) \mid (p,a,q) \in \Delta$ and $(p',a,q') \in \Delta_1 \cup \Delta_2\}$, and $\mathsf{wt}'\big((p,p'),a,(q,q')\big) = \mathsf{wt}_i(p',a,q')$ if $(p',a,q') \in \Delta_i$ for $i = 1,2$.

▶ **Lemma 12.** *For each $\overline{u} \in \Sigma_{\mathcal{V}}^+$, we have*

$$\{\!|\mathcal{A}'|\!\}(\overline{u}) = \begin{cases} \{\!|\mathcal{A}_1|\!\}(\overline{u}), & \text{if } \overline{u} \text{ is valid and } \overline{u} \models \varphi, \\ \{\!|\mathcal{A}_2|\!\}(\overline{u}), & \text{if } \overline{u} \text{ is valid and } \overline{u} \not\models \varphi, \\ \emptyset, & \text{if } \overline{u} \text{ is not valid.} \end{cases}$$

Moreover, if $\mathcal{A}_1$ and $\mathcal{A}_2$ are aperiodic (resp. unambiguous, SCC-unambiguous) then so is $\mathcal{A}'$.

Let $\mathcal{V}$ be a finite set of first-order variables and let $\mathcal{V}' = \mathcal{V} \cup \{y\}$ where $y \notin \mathcal{V}$. Given a word $\overline{w} \in \Sigma_{\mathcal{V}}^+$ and a position $i \in \mathsf{pos}(w)$, we denote by $(\overline{w}, y \mapsto i)$ the word over $\Sigma_{\mathcal{V}'}$ whose projection on $\Sigma_{\mathcal{V}}$ is $\overline{w}$ and projection on the y-component is $0^{i-1}10^{|w|-i}$, i.e., has a unique 1 on position i. Given a function $A\colon \Sigma_{\mathcal{V}'}^+ \to \mathbb{N}\langle X\rangle$, we define the function $\sum_y A\colon \Sigma_{\mathcal{V}}^+ \to \mathbb{N}\langle X\rangle$ for $\overline{w} \in \Sigma_{\mathcal{V}}^+$ by $(\sum_y A)(\overline{w}) = \biguplus_{i \in \mathsf{pos}(w)} A(\overline{w}, y \mapsto i)$.

▶ **Lemma 13.** *Let $\mathcal{A}$ be a weighted automaton over $\Sigma_{\mathcal{V}'}$. We can construct a weighted automaton $\mathcal{A}'$ over $\Sigma_{\mathcal{V}}$ such that $\{\!|\mathcal{A}'|\!\} = \sum_y \{\!|\mathcal{A}|\!\}$. Moreover, if $\mathcal{A}$ is aperiodic then $\mathcal{A}'$ is also aperiodic, and if $\mathcal{A}$ is SCC-unambiguous then $\mathcal{A}'$ is also SCC-unambiguous.*

We turn now to one of our main results: given a $\mathsf{step\text{-}wFO}$ formula Ψ, we can construct a weighted automaton for $\prod_x \Psi$ which is both aperiodic and unambiguous.

When weights are uninterpreted, a weighted automaton $\mathcal{A} = (Q, \Sigma, \Delta, \mathsf{wt}, I, F)$ is a letter-to-letter transducer from its input alphabet Σ to the output alphabet R. If in addition the input automaton is unambiguous, then we have a functional transducer. In the following, we will construct such functional transducers using the boolean output alphabet $\mathbb{B} = \{0,1\}$.

▶ **Lemma 14.** *Let $\mathcal{V} = \{y_1, \ldots, y_m\}$. Given an* FO *formula φ with free variables contained in $\mathcal{V}' = \mathcal{V} \cup \{x\}$, we can construct a transducer $\mathcal{B}_{\varphi,\mathcal{V}}$ from $\Sigma_{\mathcal{V}}$ to $\mathbb{B}$ which is aperiodic and unambiguous and such that for all words $\overline{w} \in \Sigma_{\mathcal{V}}^+$*

1. *there is a (unique) accepting run of $\mathcal{B}_{\varphi,\mathcal{V}}$ on the input word $\overline{w}$ iff it is a valid encoding of a pair (w, σ) where $w \in \Sigma^+$ and $\sigma\colon \mathcal{V} \to \mathsf{pos}(w)$ is a valuation,*
2. *and in this case, for all $1 \le i \le |w|$, the ith bit of the output is 1 iff $w, \sigma[x \mapsto i] \models \varphi$.*

Proof (sketch). Notice that $\Sigma_{\mathcal{V}'} = \Sigma_{\mathcal{V}} \times \mathbb{B}$ so letters in $\Sigma_{\mathcal{V}'}$ are of the form $(\overline{a}, 0)$ or $(\overline{a}, 1)$ where $\overline{a} \in \Sigma_{\mathcal{V}}$. Abusing the notations, when $\overline{v} \in \Sigma_{\mathcal{V}}^*$, we write $(\overline{v}, 0)$ to denote the word over $\Sigma_{\mathcal{V}'}$ whose projection on $\Sigma_{\mathcal{V}}$ is $\overline{v}$ and projection on the x-component consists of 0's only.

Consider the deterministic, complete and aperiodic automaton $\mathcal{A}_{\varphi,\mathcal{V}'} = (Q, \Sigma_{\mathcal{V}'}, \Delta, \iota, F, G)$ associated with φ by Theorem 3. We also denote by Δ the extension of the transition function to subsets of Q. So we see the deterministic and complete transition relation both as a total function $\Delta\colon Q \times \Sigma_{\mathcal{V}'} \to Q$ and $\Delta\colon 2^Q \times \Sigma_{\mathcal{V}'} \to 2^Q$.

We construct now the transducer $\mathcal{B}_{\varphi,\mathcal{V}} = (Q', \Sigma_{\mathcal{V}}, \Delta', \mathsf{wt}, I', F')$. The set of states is $Q' = Q \times 2^Q \times 2^Q \times \mathbb{B}$. The unique initial state is $\iota' = (\iota, \emptyset, \emptyset, 0)$. The set of final states is $F' = (Q \times 2^F \times 2^G \times \mathbb{B}) \setminus \{\iota'\}$. Then, we define the following transitions:

- $\delta = ((p,X,Y,b), \overline{a}, (p',X',Y',1)) \in \Delta'$ is a transition with weight $\mathsf{wt}(\delta) = 1$ if $p' = \Delta(p, (\overline{a}, 0))$, $X' = \Delta(X, (\overline{a}, 0)) \cup \{\Delta(p, (\overline{a}, 1))\}$ and $Y' = \Delta(Y, (\overline{a}, 0))$,
- $\delta = ((p,X,Y,b), \overline{a}, (p',X',Y',0)) \in \Delta'$ is a transition with weight $\mathsf{wt}(\delta) = 0$ if $p' = \Delta(p, (\overline{a}, 0))$, $X' = \Delta(X, (\overline{a}, 0))$ and $Y' = \Delta(Y, (\overline{a}, 0)) \cup \{\Delta(p, (\overline{a}, 1))\}$.

Notice that, whenever we read a new input letter $\overline{a} \in \Sigma_{\mathcal{V}}$, there is a non-deterministic choice. In the first case above, we guess that formula φ will hold on the input word when the valuation is extended by assigning x to the current position, whereas in the second case we guess that φ will not hold. The guess corresponds to the output of the transition, as required by the second condition of Lemma 14. Now, we have to check that the guess is correct. For this, the first component of $\mathcal{B}_{\varphi,\mathcal{V}}$ computes the state $p = \Delta(\iota, (\overline{u}, 0))$ reached by $\mathcal{A}_{\varphi,\mathcal{V}'}$ after reading $(\overline{u}, 0)$ where $\overline{u} \in \Sigma_{\mathcal{V}}^*$ is the current prefix of the input word. When reading the current letter $\overline{a} \in \Sigma_{\mathcal{V}}$, the transducer adds the state $\Delta(p, (\overline{a}, 1)) = \Delta(\iota, (\overline{u}, 0)(\overline{a}, 1))$ either to the "positive" X-component or to the "negative" Y-component of its state, depending on its guess as explained above. Then, the transducer continues reading the suffix $\overline{v} \in \Sigma_{\mathcal{V}}^*$ of the input word. It updates the X (resp. Y)-component so that it contains the state $q = \Delta(\iota, (\overline{u}, 0)(\overline{a}, 1)(\overline{v}, 0))$ at the end of the run. Now, the acceptance condition allows us to check that the guess was correct.

1. If $\overline{w} = \overline{uav}$ is not a valid encoding of a pair (w, σ) with $w \in \Sigma^+$ and $\sigma \colon \mathcal{V} \to \mathsf{pos}(w)$ then $q \notin F \cup G$ and the run of the transducer is not accepting. Otherwise, let $i \in \mathsf{pos}(w)$ be the position where the guess was made.
2. If the guess was positive then q belongs to the X-component and the accepting condition implies $q \in F$, which means by definition of $\mathcal{A}_{\varphi,\mathcal{V}'}$ that $w, \sigma[x \mapsto i] \models \varphi$.
3. If the guess was negative then q belongs to the Y-component and the accepting condition implies $q \in G$, which means by definition of $\mathcal{A}_{\varphi,\mathcal{V}'}$ that $w, \sigma[x \mapsto i] \not\models \varphi$. ◀

▶ **Theorem 15.** *Let $\mathcal{V} = \{y_1, \ldots, y_m\}$. Given a* step-wFO *formula Ψ with free variables contained in $\mathcal{V}' = \mathcal{V} \cup \{x\}$, we can construct a weighted automaton $\mathcal{A}_{\Psi,\mathcal{V}}$ over $\Sigma_{\mathcal{V}}$ which is aperiodic, unambiguous and equivalent to $\prod_x \Psi$, i.e., $\{\!|\mathcal{A}_{\Psi,\mathcal{V}}|\!\}(\overline{w}) = \{\!|\prod_x \Psi|\!\}_{\mathcal{V}}(\overline{w})$ for all words $\overline{w} \in \Sigma_{\mathcal{V}}^+$.*

Proof. In case $\Psi = r$ is an atomic step-wFO formula, we replace it with the equivalent $\top \,?\, r : r$ step-wFO formula. Let $\varphi_1, \ldots, \varphi_k$ be the FO formulas occurring in Ψ. By the above remark, we have $k \geq 1$. Consider the aperiodic and unambiguous transducers $\mathcal{B}_1, \ldots, \mathcal{B}_k$ given by Lemma 14. For $1 \leq i \leq k$, we let $\mathcal{B}_i = (Q_i, \Sigma_{\mathcal{V}}, \Delta_i, \mathsf{wt}_i, I_i, F_i)$. The weighted automaton $\mathcal{A}_{\Psi,\mathcal{V}} = (Q, \Sigma_{\mathcal{V}}, \Delta, \mathsf{wt}, I, F)$ is essentially a cartesian product of the transducers $\mathcal{B}_i$. More precisely, we let $Q = \prod_{i=1}^k Q_i$, $I = \prod_{i=1}^k I_i$, $F = \prod_{i=1}^k F_i$, and

$$\Delta = \{((p_1, \ldots, p_k), \overline{a}, (q_1, \ldots, q_k)) \mid (p_i, \overline{a}, q_i) \in \Delta_i \text{ for all } 1 \leq i \leq k\}\,.$$

Since the transducers $\mathcal{B}_i$ are all aperiodic and unambiguous, we deduce by Lemma 11 that $\mathcal{A}_{\Psi,\mathcal{V}}$ is also aperiodic and unambiguous. It remains to define the weight function wt.

Given a bit vector $\overline{b} = (b_1, \ldots, b_k) \in \mathbb{B}^k$ of size k, we define $\Psi(\overline{b})$ as the weight from R resulting from the step-wFO formula Ψ when the FO conditions $\varphi_1, \ldots, \varphi_k$ evaluate to $\overline{b}$. Formally, the definition is by structural induction on the step-wFO formula:

$$r(\overline{b}) = r \qquad\qquad (\varphi_i \,?\, \Psi_1 : \Psi_2)(\overline{b}) = \begin{cases} \Psi_1(\overline{b}) & \text{if } b_i = 1 \\ \Psi_2(\overline{b}) & \text{if } b_i = 0\,. \end{cases}$$

Consider a transition $\delta = ((p_1, \ldots, p_k), \overline{a}, (q_1, \ldots, q_k)) \in \Delta$ and let $\delta_i = (p_i, \overline{a}, q_i)$ for $1 \leq i \leq k$. Let $\overline{b} = (b_1, \ldots, b_k) \in \mathbb{B}^k$ where $b_i = \mathsf{wt}(\delta_i) \in \mathbb{B}$ for all $1 \leq i \leq k$. We define $\mathsf{wt}(\delta) = \Psi(\overline{b})$.

Let $\overline{w} \in \Sigma_{\mathcal{V}}^+$. If $\overline{w}$ is not a valid encoding of a pair (w, σ) then $\{\!|\prod_x \Psi|\!\}_{\mathcal{V}}(\overline{w}) = \emptyset$ by definition. Moreover, $\{\!|\mathcal{A}_{\Psi,\mathcal{V}}|\!\}(\overline{w}) = \emptyset$ since by Lemma 14, $\overline{w}$ is not in the support of $\mathcal{B}_1$. We assume below that $\overline{w}$ is a valid encoding of a pair (w, σ) where $w \in \Sigma^+$ and $\sigma \colon \mathcal{V} \to \mathsf{pos}(w)$ is a valuation. Then, each transducer $\mathcal{B}_i$ admits a unique accepting run ρ_i reading the input word $\overline{w}$. These result in the unique accepting run ρ of $\mathcal{A}_{\Psi,\mathcal{V}}$ reading $\overline{w}$. The projections of ρ on $\mathcal{B}_1, \ldots, \mathcal{B}_k$ are $\rho_1, \ldots, \rho_k$. Let $j \in \mathsf{pos}(w) = \{1, \ldots, |w|\}$ be a position in $\overline{w}$ and let δ^j be the j-th transition of ρ. For $1 \leq i \leq k$, we denote by δ_i^j the projection of δ^j on $\mathcal{B}_i$ and we let $b_i^j = \mathsf{wt}(\delta_i^j)$. By Lemma 14, we get $b_i^j = 1$ iff $w, \sigma[x \mapsto j] \models \varphi_i$. Finally, let $\overline{b}^j = (b_1^j, \ldots, b_k^j)$. From the above, we deduce that $[\![\Psi]\!]_{\mathcal{V}\cup\{x\}}(w, \sigma[x \mapsto j]) = \Psi(\overline{b}^j) = \mathsf{wt}(\delta^j)$. Putting things together, we have $\{\!|\mathcal{A}_{\Psi,\mathcal{V}}|\!\}(w, \sigma) = \{\{\mathsf{wt}(\rho)\}\} = \{\{\mathsf{wt}(\delta^1) \cdots \mathsf{wt}(\delta^{|w|}\}\} = \{\!|\prod_x \Psi|\!\}_{\mathcal{V}}(w, \sigma)$. ◀

▶ **Theorem 16.** *Let Φ be a* wFO *sentence. We can construct an aperiodic SCC-unambiguous weighted automaton $\mathcal{A}$ such that $\{\!|\mathcal{A}|\!\} = \{\!|\Phi|\!\}$. Moreover, if Φ does not contain the sum operations $+$ and $\sum_x$, then $\mathcal{A}$ can be chosen to be unambiguous. If Φ does not contain the sum operation $\sum_x$, we can construct $\mathcal{A}$ as a finite union of unambiguous weighted automata.*

Proof. We proceed by structural induction on Φ. For $\Phi = \mathbf{0}$ this is trivial. For $\Phi = \prod_x \Psi$ with a step-wFO formula Ψ, we obtain an aperiodic unambiguous weighted automaton $\mathcal{A}$ by Theorem 15. For formulas $\varphi\,?\,\Phi_1 : \Phi_2$, $\Phi_1 + \Phi_2$ and $\sum_x \Phi$, we apply Lemmas 12, 11, 13. ◀

In the proof of Theorem 16, we may obtain the final statement also as a consequence of the preceding one by the following observations which could be of independent interest. Let φ be an FO-formula and Φ_1, Φ_2 two wFO formulas, with free variables contained in $\mathcal{V}$. Then,

$$\{\!|\varphi\,?\,\Phi_1 : \Phi_2|\!\}_{\mathcal{V}} = \{\!|\varphi\,?\,\Phi_1 : \mathbf{0} + \neg\varphi\,?\,\Phi_2 : \mathbf{0}|\!\}_{\mathcal{V}} \,,$$
$$\{\!|\varphi\,?\,\Phi_1 + \Phi_2 : \mathbf{0}|\!\}_{\mathcal{V}} = \{\!|\varphi\,?\,\Phi_1 : \mathbf{0} + \varphi\,?\,\Phi_2 : \mathbf{0}|\!\}_{\mathcal{V}} \,.$$

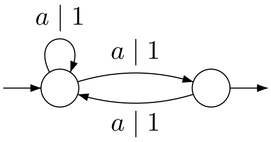
Hence, given a wFO sentence Φ not containing the sum operation $\sum_x$, we can rewrite Φ as a sum of $\mathbf{0}$, $\prod_x \Psi$ and if-then-else sentences of the form $\varphi\,?\,\Phi' : \mathbf{0}$ where Φ' does not contain the sum operations $+$ or $\sum_x$.

Proof of Thm 1. Immediate by Theorem 10, Theorem 4, Corollary 8 and Theorem 16. ◀

7 Concluding remarks

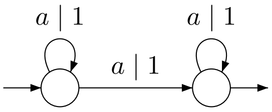
We introduced a model of aperiodic weighted automata and showed that a suitable concept of weighted first order logic and two natural sublogics have the same expressive power as polynomially ambiguous, finitely ambiguous, resp. unambigous aperiodic weighted automata. For the three semirings $\mathbb{N}_{+,\times}$, $\mathbb{N}_{\max,+}$ and $\mathbb{N}_{\min,+}$ the hierarchies of these automata classes and thereby of the corresponding logics are strict. Some separating examples are given below. Proofs and other separating examples can be found in [14].

▶ **Example 17.** Let $\Sigma = \{a\}$ and consider the automaton $\mathcal{A}$ below over the semiring $\mathbb{N}_{+,\times}$ of natural numbers. Note that the weighted automaton computes the sequence $(F_n)_{n\geq 0}$ of Fibonacci numbers $0, 1, 1, 2, 3, 5, \cdots$. More precisely, for any $n \in \mathbb{N}$, we have $[\![\mathcal{A}]\!](a^n) = F_n$.



Clearly, $\mathcal{A}$ is exponentially ambiguous and aperiodic with index 2. But $[\![\mathcal{A}]\!]$ cannot be realized by an aperiodic polynomially ambiguous weighted automaton. In [31], it was shown that the Fibonacci numbers cannot be computed by copyless cost-register automata. ⌟

▶ **Example 18.** Consider the automaton $\mathcal{A}$ below over $\Sigma = \{a\}$ and the semiring $\mathbb{N}_{+,\times}$. Clearly, $[\![\mathcal{A}]\!](a^n) = n$ for each $n > 0$, and $\mathcal{A}$ is aperiodic and polynomially (even linearly) ambiguous. But $\mathcal{A}$ is not equivalent to any finitely ambiguous weighted automaton.



◀

▶ **Example 19.** Let $\Sigma = \{a\}$ and consider the function $f \colon \Sigma^* \to \mathbb{N}$ defined by $f(a^n) = 2^n + 1$. We have $f = [\![\mathcal{A}]\!]$ for some aperiodic and 2-ambiguous weighted automaton: one self-loop computes 2^n and another self-loop computes 1. But f cannot be realized by an unambiguous weighted automaton over $\mathbb{N}_{+,\times}$.

Our main theorem generalizes to the weighted setting a classical result of automata theory. A challenging open problem is to obtain similar results for suitable weighted linear temporal logics. Another interesting problem is to characterize wFO with unrestricted weighted products, possibly using *aperiodic* restrictions of the pebble weighted automata studied in [4, 27, 5].

Decidability problems for wFO or equivalently for weighted aperiodic automata are also open and very interesting. For instance, given a wMSO sentence, is there an equivalent wFO sentence? Decidability may indeed depend on the specific semiring.

References

1 Parvaneh Babari, Manfred Droste, and Vitaly Perevoshchikov. Weighted register automata and weighted logic on data words. *Theor. Comput. Sci.*, 744:3–21, 2018.

2 Jean Berstel and Christophe Reutenauer. *Rational Series and their Languages*. Springer, 1988.

3 Benedikt Bollig and Paul Gastin. Weighted versus Probabilistic Logics. In Volker Diekert and Dirk Nowotka, editors, *International Conference on Developments in Language Theory (DLT'09)*, volume 5583 of *Lecture Notes in Computer Science*, pages 18–38. Springer, 2009.

4 Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. Pebble Weighted Automata and Transitive Closure Logics. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming*, pages 587–598, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

5 Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. Pebble Weighted Automata and Weighted Logics. *ACM Transactions on Computational Logic*, 15(2):1–35, 2014.

6 J. Richard Büchi. Weak Second-Order Arithmetic and Finite Automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

7 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Expressiveness and Closure Properties for Quantitative Languages. *Logical Methods in Computer Science*, 6(3), 2010.

8 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010.

9 Volker Diekert and Paul Gastin. First-order definable languages. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.

10 Manfred Droste and Stefan Dück. Weighted Automata and Logics on Graphs. In *Mathematical Foundations of Computer Science (MFCS'15)*, volume 9234 of *Lecture Notes in Computer Science*, pages 192–204. Springer, 2015.

11 Manfred Droste and Stefan Dück. Weighted automata and logics for infinite nested words. *Inf. Comput.*, 253:448–466, 2017.

12 Manfred Droste and Paul Gastin. Weighted Automata and Weighted Logics. In *International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 2005.

13 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.

14 Manfred Droste and Paul Gastin. Aperiodic Weighted Automata and Weighted First-Order Logic. *CoRR*, abs/1902.08149, 2019.

15 Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata.* Springer Berlin Heidelberg, 2009.

16 Manfred Droste and Ingmar Meinecke. Weighted automata and weighted MSO logics for average and long-time behaviors. *Inf. Comput.*, 220:44–59, 2012.

17 Manfred Droste and Vitaly Perevoshchikov. Multi-weighted Automata and MSO Logic. *Theory Comput. Syst.*, 59(2):231–261, 2016.

18 Manfred Droste and George Rahonis. Weighted automata and weighted logics on infinite words. *Izvestiya VUZ. Matematika*, 54:26–45, 2010.

19 Manfred Droste and Heiko Vogler. Weighted tree automata and weighted logics. *Theor. Comput. Sci.*, 366(3):228–247, 2006.

20 Manfred Droste and Heiko Vogler. Weighted automata and multi-valued logics over arbitrary bounded lattices. *Theor. Comput. Sci.*, 418:14–36, 2012.

21 Calvin C. Elgot. Decision Problems of Finite Automata Design and Related Arithmetics. *Transactions of the American Mathematical Society*, 98:21–52, 1961.

22 Ina Fichtner. Weighted Picture Automata and Weighted Logics. *Theory Comput. Syst.*, 48(1):48–78, 2011.

23 Paul Gastin and Benjamin Monmege. A unifying survey on weighted logics and weighted automata. *Soft Computing*, 22(4):1047–1065, December 2018.

24 Oscar H Ibarra and Bala Ravikumar. On sparseness, ambiguity and other decision problems for acceptors and transducers. In *Symposium on Theoretical Aspects of Computer Science (STACS'86)*, volume 210 of *Lecture Notes in Computer Science*, pages 171–179. Springer, 1986.

25 Daniel Kirsten. A Burnside Approach to the Termination of Mohri's Algorithm for Polynomially Ambiguous Min-Plus-Automata. *RAIRO - Theoretical Informatics and Applications*, 42(3):553–581, June 2008.

26 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, November 2004.

27 Stephan Kreutzer and Cristian Riveros. Quantitative Monadic Second-Order Logic. In *Symposium on Logic in Computer Science (LICS'13)*. IEEE, June 2013.

28 Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages.* Springer Berlin Heidelberg, 1986.

29 Eleni Mandrali and George Rahonis. On weighted first-order logics with discounting. *Acta Informatica*, 51(2):61–106, January 2014.

30 Filip Mazowiecki and Cristian Riveros. Pumping Lemmas for Weighted Automata. In *Symposium on Theoretical Aspects of Computer Science (STACS'18)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

31 Filip Mazowiecki and Cristian Riveros. Copyless cost-register automata: Structure, expressiveness, and closure properties. *Journal of Computer and System Sciences*, 100:1–29, March 2019.

32 Robert McNaughton and Seymour Papert. *Counter-Free Automata.* The MIT Press, Cambridge, Mass., 1971.

33 Erik Paul. On Finite and Polynomial Ambiguity of Weighted Tree Automata. In *International Conference on Developments in Language Theory (DLT'16)*, volume 9840 of *Lecture Notes in Computer Science*, pages 368–379. Springer, 2016.

34 Karin Quaas. MSO logics for weighted timed automata. *Formal Methods in System Design*, 38(3):193–222, 2011.

35 Christophe Reutenauer. *Propriétés arithmétiques et topologiques de séries rationnelles en variables non commutatives*. PhD thesis, Université Paris VI, 1977.

36 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

37 Jacques Sakarovitch and Rodrigo de Souza. Lexicographic Decomposition of k-Valued Transducers. *Theory of Computing Systems*, 47(3):758–785, April 2009.

38 Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer, 1978.

39 Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, September 1961.

40 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

41 Boris A. Trakhtenbrot. Finite Automata and Logic of Monadic Predicates. *Doklady Akademii Nauk SSSR*, 149:326–329, 1961.

42 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, October 1991.

A Congruence-based Perspective on Automata Minimization Algorithms

Pierre Ganty
IMDEA Software Institute, Madrid, Spain
pierre.ganty@imdea.org

Elena Gutiérrez
IMDEA Software Institute, Madrid, Spain
Universidad Politécnica de Madrid, Spain
elena.gutierrez@imdea.org

Pedro Valero
IMDEA Software Institute, Madrid, Spain
Universidad Politécnica de Madrid, Spain
pedro.valero@imdea.org

Abstract

In this work we use a framework of finite-state automata constructions based on equivalences over words to provide new insights on the relation between well-known methods for computing the minimal deterministic automaton of a language.

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

Keywords and phrases Double-Reversal Method, Minimization, Automata, Congruences, Regular Languages

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.77

Related Version An extended version is available at `https://arxiv.org/abs/1906.06194`

Funding *Pierre Ganty*: Supported by the Spanish Ministry of Economy and Competitiveness project No. PGC2018-102210-B-I00, BOSCO - Foundations for the development, analysis and understanding of BlOck chains and Smart COntracts, by the Madrid Regional Government project No. S2018/TCS-4339, BLOQUES - Contratos inteligentes y Blockchains Escalables y Seguros mediante Verificación y Análisis, and by a Ramón y Cajal fellowship RYC-2016-20281.
Elena Gutiérrez: Supported by BES-2016-077136 grant from the Spanish Ministry of Economy, Industry and Competitiveness.

1 Introduction

In this paper we consider the problem of building the minimal deterministic finite-state automaton generating a given regular language. This is a classical issue that arises in many different areas of computer science such as verification, regular expression searching and natural language processing, to name a few.

There exists a number of methods, such as Hopcroft's [10] and Moore's algorithms [14], that receive as input a deterministic finite-state automaton (DFA for short) generating a language and build the minimal DFA for that language. In general, these methods rely on computing a partition of the set of states of the input DFA which is then used as the set of states of the minimal DFA.

On the other hand, Brzozowski [4] proposed the *double-reversal method* for building the minimal DFA for the language generated by an input non-deterministic automaton (NFA for short). This algorithm alternates a reverse operation and a determinization operation twice,

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 77; pp. 77:1–77:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

relying on the fact that, for any given NFA $\mathcal{N}$, if the reverse automaton of $\mathcal{N}$ is deterministic then the determinization operation yields the minimal DFA for the language of $\mathcal{N}$. This method has been recently generalized by Brzozowski and Tamm [5]. They showed the following *necessary and sufficient condition*: the determinization operation yields the minimal DFA for the language of $\mathcal{N}$ if and only if the reverse automaton of $\mathcal{N}$ is *atomic*.

It is well-known that all these approaches to the DFA minimization problem aim to compute Nerode's equivalence relation for the considered language. However, the double-reversal method and its later generalization appear to be quite isolated from other methods such as Hopcroft's and Moore's algorithms. This has led to different attempts to better explain Brzozowski's method [3] and its connection with other minimization algorithms [1, 7, 16]. We use a framework of automata constructions based on equivalence classes over *words* to give new insights on the relation between these algorithms.

In this paper we consider equivalence relations over words on an alphabet Σ that induce finite partitions over Σ^*. Furthermore, we require that these partitions are well-behaved with respect to concatenation, namely, *congruences*. Given a regular language L and an equivalence relation satisfying these conditions, we use well-known automata constructions that yield automata generating the language L [6, 13]. In this work, we consider two types of equivalence relations over words verifying the required conditions.

First, we define a *language-based equivalence*, relative to a regular language, that behaves well with respect to *right* concatenation, also known as the right Nerode's equivalence relation for the language. When applying the automata construction to the right Nerode's equivalence, we obtain the minimal DFA for the given language [6, 13]. In addition, we define an *automata-based equivalence*, relative to an NFA. When applying the automata construction to the automata-based equivalence we obtain a determinized version of the input NFA.

On the other hand, we also obtain counterpart automata constructions for relations that are well-behaved with respect to *left* concatenation. In this case, language-based and automata-based equivalences yield, respectively, the minimal co-deterministic automaton and a co-deterministic NFA for the language.

The relation between the automata constructions resulting from the language-based and the automata-based congruences, together with the the duality between right and left congruences, allows us to relate determinization and minimization operations. As a result, we formulate a sufficient and necessary condition that guarantees that determinizing an automaton yields the minimal DFA. This formulation evidences the relation between the double-reversal and the state partition refinement minimization methods.

We start by giving a simple proof of Brzozowski's double-reversal method [4], to later address the generalization of Brzozowski and Tamm [5]. Furthermore, we relate the iterations of Moore's partition refinement algorithm, which works on the states of the input DFA, to the iterations of the greatest fixpoint algorithm that builds the right Nerode's partition on words. We conclude by relating the automata constructions introduced by Brzozowski and Tamm [5], named the *átomaton* and *the partial átomaton*, to the automata constructions described in this work.

Structure of the paper. After preliminaries in Section 2, we introduce in Section 3 the automata constructions based on congruences on words and establish the duality between these constructions when using right and left congruences. Then, in Section 4, we define language-based and automata-based congruences and analyze the relations between the resulting automata constructions. In Section 5, we study a collection of well-known constructions for the minimal DFA. Finally, we give further details on related work in Section 6. For space reasons, missing proofs are deferred to the extended version of this paper [9].

2 Preliminaries

Languages. Let Σ be a finite nonempty *alphabet* of symbols. Given a word $w \in \Sigma^*$, w^R denotes the *reverse* of w. Given a language $L \subseteq \Sigma^*$, $L^R \stackrel{\text{def}}{=} \{w^R \mid w \in L\}$ denotes the *reverse language* of L. We denote by L^c the *complement* of the language L. The *left (resp. right) quotient* of L by a word u is defined as the language $u^{-1}L \stackrel{\text{def}}{=} \{x \in \Sigma^* \mid ux \in L\}$ (resp. $Lu^{-1} \stackrel{\text{def}}{=} \{x \in \Sigma^* \mid xu \in L\}$).

Automata. A *(nondeterministic) finite-state automaton* (NFA for short), or simply *automaton*, is a 5-tuple $\mathcal{N} = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of *states*, Σ is an alphabet, $I \subseteq Q$ are the *initial* states, $F \subseteq Q$ are the *final* states, and $\delta : Q \times \Sigma \to \wp(Q)$ is the *transition* function. We denote the *extended transition function* from Σ to Σ^* by $\hat{\delta}$. Given $S, T \subseteq Q$, $W^{\mathcal{N}}_{S,T} \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid \exists q \in S, q' \in T : q' \in \hat{\delta}(q, w)\}$. In particular, when $S = \{q\}$ and $T = F$, we define *the right language* of state q as $W^{\mathcal{N}}_{q,F}$. Likewise, when $S = I$ and $T = \{q\}$, we define the *left language* of state q as $W^{\mathcal{N}}_{I,q}$. We define $\mathrm{post}^{\mathcal{N}}_w(S) \stackrel{\text{def}}{=} \{q \in Q \mid w \in W^{\mathcal{N}}_{S,q}\}$ and $\mathrm{pre}^{\mathcal{N}}_w(S) \stackrel{\text{def}}{=} \{q \in Q \mid w \in W^{\mathcal{N}}_{q,S}\}$. In general, we omit the automaton $\mathcal{N}$ from the superscript when it is clear from the context. We say that a state q is *unreachable* iff $W^{\mathcal{N}}_{I,q} = \emptyset$ and we say that q is *empty* iff $W^{\mathcal{N}}_{q,F} = \emptyset$. Finally, note that $\mathcal{L}(\mathcal{N}) = \bigcup_{q \in I} W^{\mathcal{N}}_{q,F} = \bigcup_{q \in F} W^{\mathcal{N}}_{I,q} = W^{\mathcal{N}}_{I,F}$.

Given an NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$, the *reverse NFA* for $\mathcal{N}$, denoted by $\mathcal{N}^R$, is defined as $\mathcal{N}^R = (Q, \Sigma, \delta_r, F, I)$ where $q \in \delta_r(q', a)$ iff $q' \in \delta(q, a)$. Clearly, $\mathcal{L}(\mathcal{N})^R = \mathcal{L}(\mathcal{N}^R)$.

A *deterministic finite-state automaton* (DFA for short) is an NFA such that, $I = \{q_0\}$, and, for every state $q \in Q$ and every symbol $a \in \Sigma$, there exists exactly one $q' \in Q$ such that $\delta(q, a) = q'$. According to this definition, DFAs are always *complete*, i.e., they define a transition for each state and input symbol. In general, we denote NFAs by $\mathcal{N}$, using $\mathcal{D}$ for DFAs when the distinction is important. A *co-deterministic finite-state automata* (co-DFA for short) is an NFA $\mathcal{N}$ such that $\mathcal{N}^R$ is deterministic. In this case, co-DFAs are always *co-complete*, i.e., for each target state q' and each input symbol, there exists a source state q such that $\delta(q, a) = q'$. Recall that, given an NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$, the well-known *subset construction* builds a DFA $\mathcal{D} = (\wp(Q), \Sigma, \delta_d, \{I\}, F_d)$ where $F_d = \{S \in \wp(Q) \mid S \cap F \neq \emptyset\}$ and $\delta_d(S, a) = \{q' \mid \exists q \in S, q' \in \delta(q, a)\}$ for every $a \in \Sigma$, that accepts the same language as $\mathcal{N}$ [11]. Given an NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$, we denote by $\mathcal{N}^D$ the DFA that results from applying the subset construction to $\mathcal{N}$ where only subsets (including the empty subset) that are reachable from the initial subset of $\mathcal{N}^D$ are used. Then, $\mathcal{N}^D$ possibly contains empty states but no state is unreachable. A DFA for the language $\mathcal{L}(\mathcal{N})$ is *minimal*, denoted by $\mathcal{N}^{DM}$, if it has no unreachable states and no two states have the same right language. The minimal DFA for a regular language is unique modulo isomorphism.

Equivalence Relations and Partitions. Recall that an *equivalence relation* on a set X is a binary relation $\sim$ that is reflexive, symmetric and transitive. Every equivalence relation $\sim$ on X induces a *partition* $P_\sim$ of X, i.e., a family $P_\sim = \{B_i\}_{i \in \mathcal{I}} \subseteq \wp(X)$ of subsets of X, with $\mathcal{I} \subseteq \mathbb{N}$, such that:

(i) $B_i \neq \emptyset$ for all $i \in \mathcal{I}$;
(ii) $B_i \cap B_j = \emptyset$, for all $i, j \in \mathcal{I}$ with $i \neq j$; and
(iii) $X = \bigcup_{i \in \mathcal{I}} B_i$.

We say that a partition is *finite* when $\mathcal{I}$ is finite. Each B_i is called a *block* of the partition. Given $u \in X$, then $P_\sim(u)$ denotes the unique block that contains u and corresponds to the *equivalence class* u w.r.t. $\sim$, $P_\sim(u) \stackrel{\text{def}}{=} \{v \in X \mid u \sim v\}$. This definition can be extended in a natural way to a set $S \subseteq X$ as $P_\sim(S) \stackrel{\text{def}}{=} \bigcup_{u \in S} P_\sim(u)$. We say that the partition $P_\sim$

represents precisely S iff $P_{\sim}(S) = S$. An equivalence relation $\sim$ is *of finite index* iff $\sim$ defines a finite number of equivalence classes, i.e., the induced partition $P_{\sim}$ is finite. In the following, we will always consider equivalence relations of finite index, i.e., finite partitions.

Finally, denote $Part(X)$ the set of partitions of X. We use the standard refinement ordering $\preceq$ between partitions: let $P_1, P_2 \in Part(X)$, then $P_1 \preceq P_2$ iff for every $B \in P_1$ there exists $B' \in P_2$ such that $B \subseteq B'$. Then, we say that P_1 is *finer than* P_2 (or equivalently, P_2 is *coarser than* P_1). Given $P_1, P_2 \in Part(X)$, define the *coarsest common refinement*, denoted by $P_1 \curlywedge P_2$, as the coarsest partition $P \in Part(X)$ that is finer than both P_1 and P_2. Likewise, define the *finest common coarsening*, denoted by $P_1 \curlyvee P_2$, as the finest partition P that is coarser than both P_1 and P_2. Recall that $(Part(X), \preceq, \curlyvee, \curlywedge)$ is a complete lattice where the top (coarsest) element is $\{X\}$ and the bottom (finest) element is $\{\{x\} \mid x \in X\}$.

3 Automata Constructions from Congruences

We will consider equivalence relations on Σ^* (and their corresponding partitions) with good properties w.r.t. concatenation. An equivalence relation $\sim$ is a *right (resp. left) congruence* iff for all $u, v \in \Sigma^*$, we have that $u \sim v \Rightarrow ua \sim va$, for all $a \in \Sigma$ (resp. $u \sim v \Rightarrow au \sim av$). We will denote right congruences (resp. left congruences) by $\sim^r$ (resp. $\sim^\ell$). The following lemma gives a characterization of right and left congruences.

▶ **Lemma 1.** *The following properties hold:*

1. *$\sim^r$ is a right congruence iff $P_{\sim^r}(v)u \subseteq P_{\sim^r}(vu)$, for all $u, v \in \Sigma^*$.*
2. *$\sim^\ell$ is a left congruence iff $uP_{\sim^\ell}(v) \subseteq P_{\sim^\ell}(uv)$, for all $u, v \in \Sigma^*$.*

Given a right congruence $\sim^r$ and a regular language $L \subseteq \Sigma^*$ such that $P_{\sim^r}$ represents precisely L, i.e., $P_{\sim^r}(L) = L$, the following automata construction recognizes exactly the language L [13].

▶ **Definition 2** (Automata construction $\mathsf{H}^r(\sim^r, L)$)**.** *Let $\sim^r$ be a right congruence and let $P_{\sim^r}$ be the partition induced by $\sim^r$. Let $L \subseteq \Sigma^*$ be a language. Define the automaton $\mathsf{H}^r(\sim^r, L) = (Q, \Sigma, \delta, I, F)$ where $Q = \{P_{\sim^r}(u) \mid u \in \Sigma^*\}$, $I = \{P_{\sim^r}(\varepsilon)\}$, $F = \{P_{\sim^r}(u) \mid u \in L\}$, and $\delta(P_{\sim^r}(u), a) = P_{\sim^r}(v)$ iff $P_{\sim^r}(u)a \subseteq P_{\sim^r}(v)$, for all $u, v \in \Sigma^*$ and $a \in \Sigma$.*

▶ Remark 3. Note that $\mathsf{H}^r(\sim^r, L)$ is *finite* since we assume $\sim^r$ is of finite index. Note also that $\mathsf{H}^r(\sim^r, L)$ is a complete *deterministic* finite-state automaton since, for each $u \in \Sigma^*$ and $a \in \Sigma$, there exists *exactly one* block $P_{\sim^r}(v)$ such that $P_{\sim^r}(u)a \subseteq P_{\sim^r}(v)$, which is $P_{\sim^r}(ua)$. Finally, observe that $\mathsf{H}^r(\sim^r, L)$ possibly contains empty states but no state is unreachable.

▶ **Lemma 4.** *Let $\sim^r$ be a right congruence and let $L \subseteq \Sigma^*$ be a language such that $P_{\sim^r}(L) = L$. Then $\mathcal{L}(\mathsf{H}^r(\sim^r, L)) = L$.*

Due to the left-right duality between $\sim^\ell$ and $\sim^r$, we can give a similar automata construction such that, given a left congruence $\sim^\ell$ and a language $L \subseteq \Sigma^*$ with $P_{\sim^\ell}(L) = L$, recognizes exactly the language L.

▶ **Definition 5** (Automata construction $\mathsf{H}^\ell(\sim^\ell, L)$)**.** *Let $\sim^\ell$ be a left congruence and let $P_{\sim^\ell}$ be the partition induced by $\sim^\ell$. Let $L \subseteq \Sigma^*$ be a language. Define the automaton $\mathsf{H}^\ell(\sim^\ell, L) = (Q, \Sigma, \delta, I, F)$ where $Q = \{P_{\sim^\ell}(u) \mid u \in \Sigma^*\}$, $I = \{P_{\sim^\ell}(u) \mid u \in L\}$, $F = \{P_{\sim^\ell}(\varepsilon)\}$, and $P_{\sim^\ell}(v) \in \delta(P_{\sim^\ell}(u), a)$ iff $aP_{\sim^\ell}(v) \subseteq P_{\sim^\ell}(u)$, for all $u, v \in \Sigma^*$ and $a \in \Sigma$.*

▶ Remark 6. In this case, $\mathsf{H}^\ell(\sim^\ell, L)$ is a co-complete *co-deterministic* finite-state automaton since, for each $v \in \Sigma^*$ and $a \in \Sigma$, there exists *exactly one* block $P_{\sim^\ell}(u)$ such that $aP_{\sim^\ell}(v) \subseteq P_{\sim^\ell}(u)$, which is $P_{\sim^\ell}(av)$. Finally, observe that $\mathsf{H}^\ell(\sim^\ell, L)$ possibly contains unreachable states but no state is empty.

▶ **Lemma 7.** *Let $\sim^\ell$ be a left congruence and let $L \subseteq \Sigma^*$ be a language such that $P_{\sim^\ell}(L) = L$. Then $\mathcal{L}(\mathsf{H}^\ell(\sim^\ell, L)) = L$.*

Lemma 8 shows that H^ℓ and H^r inherit the left-right duality between $\sim^\ell$ and $\sim^r$.

▶ **Lemma 8.** *Let $\sim^r$ and $\sim^\ell$ be a right and left congruence respectively, and let $L \subseteq \Sigma^*$ be a language. If the following property holds*

$$u \sim^r v \Leftrightarrow u^R \sim^\ell v^R \tag{1}$$

then $\mathsf{H}^r(\sim^r, L)$ is isomorphic to $\left(\mathsf{H}^\ell(\sim^\ell, L^R)\right)^R$.

4 Language-based Congruences and their Approximation using NFAs

Given a language $L \subseteq \Sigma^*$, we recall the following equivalence relations on Σ^*, which are often denoted as *Nerode's equivalence relations* (e.g., see [13]).

▶ **Definition 9** (Language-based Equivalences). *Let $u, v \in \Sigma^*$ and let $L \subseteq \Sigma^*$ be a language. Define:*

$$u \sim^r_L v \Leftrightarrow u^{-1}L = v^{-1}L \qquad \text{Right-}\textit{language-based Equivalence} \tag{2}$$

$$u \sim^\ell_L v \Leftrightarrow Lu^{-1} = Lv^{-1} \qquad \text{Left-}\textit{language-based Equivalence} \tag{3}$$

Note that the right and left language-based equivalences defined above are, respectively, right and left *congruences*. Furthermore, when L is a regular language, $\sim^r_L$ and $\sim^\ell_L$ are of *finite index* [6, 13]. Since we are interested in congruences of finite index (or equivalently, finite partitions), we will always assume that L is a regular language over Σ.

The following result states that, given a language L, the right Nerode's equivalence induces the coarsest partition of Σ^* which is a right congruence and precisely represents L.

▶ **Lemma 10** (de Luca and Varricchio [8]). *Let $L \subseteq \Sigma^*$ be a regular language. Then,*

$$P_{\sim^r_L} = \bigvee\{P_{\sim^r} \mid \sim^r \textit{ is a right congruence and } P_{\sim^r}(L) = L\} \ .$$

In a similar way, one can prove that the same property holds for the left Nerode's equivalence. Therefore, as we shall see, applying the construction H to these equivalences yields minimal automata. However, computing them becomes unpractical since languages are possibly infinite, even if they are regular. Thus, we will consider congruences based on the states of the NFA-representation of the language which induce finer partitions of Σ^* than Nerode's equivalences. In this sense, we say that the automata-based equivalences *approximate* Nerode's equivalences.

▶ **Definition 11** (Automata-based Equivalences). *Let $u, v \in \Sigma^*$ and let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA. Define:*

$$u \sim^r_{\mathcal{N}} v \Leftrightarrow \mathrm{post}^{\mathcal{N}}_u(I) = \mathrm{post}^{\mathcal{N}}_v(I) \qquad \text{Right-}\textit{automata-based Equivalence} \tag{4}$$

$$u \sim^\ell_{\mathcal{N}} v \Leftrightarrow \mathrm{pre}^{\mathcal{N}}_u(F) = \mathrm{pre}^{\mathcal{N}}_v(F) \qquad \text{Left-}\textit{automata-based Equivalence} \tag{5}$$

Note that the right and left automata-based equivalences defined above are, respectively, right and left *congruences*. Furthermore, they are of *finite index* since each equivalence class is represented by a subset of states of $\mathcal{N}$.

The following result gives a sufficient and necessary condition for the language-based (Definition 9) and the automata-based equivalences (Definition 11) to coincide.

▶ **Lemma 12.** *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an automaton with $L = \mathcal{L}(\mathcal{N})$. Then,*

$$\sim^r_L = \sim^r_{\mathcal{N}} \quad \textit{iff} \quad \forall u, v \in \Sigma^*,\ W^{\mathcal{N}}_{\mathrm{post}^{\mathcal{N}}_u(I),F} = W^{\mathcal{N}}_{\mathrm{post}^{\mathcal{N}}_v(I),F} \Leftrightarrow \mathrm{post}^{\mathcal{N}}_u(I) = \mathrm{post}^{\mathcal{N}}_v(I) \ . \tag{6}$$

4.1 Automata Constructions

In what follows, we will use Min and Det to denote the construction H when applied, respectively, to the language-based congruences induced by a regular language and the automata-based congruences induced by an NFA.

▶ **Definition 13.** *Let $\mathcal{N}$ be an NFA generating the language $L = \mathcal{L}(\mathcal{N})$. Define:*

$$\mathsf{Min}^r(L) \stackrel{\text{def}}{=} \mathsf{H}^r(\sim^r_L, L) \qquad \mathsf{Det}^r(\mathcal{N}) \stackrel{\text{def}}{=} \mathsf{H}^r(\sim^r_{\mathcal{N}}, L)$$
$$\mathsf{Min}^\ell(L) \stackrel{\text{def}}{=} \mathsf{H}^\ell(\sim^\ell_L, L) \qquad \mathsf{Det}^\ell(\mathcal{N}) \stackrel{\text{def}}{=} \mathsf{H}^\ell(\sim^\ell_{\mathcal{N}}, L) \ .$$

Given an NFA $\mathcal{N}$ generating the language $L = \mathcal{L}(\mathcal{N})$, all constructions in the above definition yield automata generating L. However, while the constructions using the right congruences result in DFAs, the constructions relying on left congruences result in co-DFAs. Furthermore, since the pairs of relations (2)-(3) and (4)-(5), from Definition 9 and 11 respectively, are dual, i.e., they satisfy the hypothesis of Lemma 8, it follows that $\mathsf{Min}^\ell(L)$ is isomorphic to $(\mathsf{Min}^r(L^R))^R$ and $\mathsf{Det}^\ell(\mathcal{N})$ is isomorphic to $(\mathsf{Det}^r(\mathcal{N}^R))^R$.

On the other hand, since Min^r relies on the language-based congruences, the resulting DFA is minimal, which is not guaranteed to occur with Det^r. This easily follows from the fact that the states of the automata constructions are the equivalence classes of the given congruences and there is no right congruence (representing L precisely) that is coarser than the right Nerode's equivalence (see Lemma 10).

Finally, since every co-deterministic automaton satisfies the right-hand side of Equation (6), it follows that determinizing (Det^r) a co-deterministic automaton ($\mathsf{Det}^\ell(\mathcal{N})$) results in the minimal DFA ($\mathsf{Min}^r(\mathcal{L}(\mathcal{N}))$), as already proved by Sakarovitch [15, Proposition 3.13].

We formalize all these notions in Theorem 14. Finally, Figure 1 summarizes all these well-known connections between the automata constructions given in Definition 13.

▶ **Theorem 14.** *Let $\mathcal{N}$ be an NFA generating language $L = \mathcal{L}(\mathcal{N})$. Then the following properties hold:*

(a) $\mathcal{L}(\mathsf{Min}^r(L)) = \mathcal{L}(\mathsf{Min}^\ell(L)) = L = \mathcal{L}(\mathsf{Det}^r(\mathcal{N})) = \mathcal{L}(\mathsf{Det}^\ell(\mathcal{N}))$.

(b) $\mathsf{Min}^r(L)$ *is isomorphic to the minimal deterministic automaton for L.*

(c) $\mathsf{Det}^r(\mathcal{N})$ *is isomorphic to $\mathcal{N}^D$.*

(d) $\mathsf{Min}^\ell(L)$ *is isomorphic to* $(\mathsf{Min}^r(L^R))^R$.

(e) $\mathsf{Det}^\ell(\mathcal{N})$ *is isomorphic to* $(\mathsf{Det}^r(\mathcal{N}^R))^R$.

(f) $\mathsf{Det}^r(\mathsf{Det}^\ell(\mathcal{N}))$ *is isomorphic to* $\mathsf{Min}^r(L)$.

5 A Congruence-based Perspective on Known Algorithms

We can find in the literature several well-known independent techniques for the construction of minimal DFAs. Some of these techniques are based on refining a state partition of an input DFA, such as Moore's algorithm [14], while others directly manipulate an input NFA, such as the double-reversal method [4]. Now, we establish a connection between these algorithms through Theorem 16, which gives a necessary and sufficient condition on an NFA so that determinizing it yields the minimal DFA.

▶ **Lemma 15.** *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA with $L = \mathcal{L}(\mathcal{N})$ and $\sim^r_L = \sim^r_{\mathcal{N}}$. Then $\forall q \in Q,\ P_{\sim^r_L}(W^{\mathcal{N}}_{I,q}) = W^{\mathcal{N}}_{I,q}$.*

$$\begin{tikzcd}
\mathcal{N} \arrow[r, "\mathsf{Det}^\ell"] \arrow[rr, bend left, "\mathsf{Min}^r"] \arrow[d, leftrightarrow, "R"'] & \mathsf{Det}^\ell(\mathcal{N}) \arrow[r, "\mathsf{Det}^r"] \arrow[d, leftrightarrow, "R"'] & \mathsf{Det}^r(\mathsf{Det}^\ell(\mathcal{N})) \arrow[d, leftrightarrow, "R"'] \\
\mathcal{N}^R \arrow[r, "\mathsf{Det}^r"] \arrow[rr, bend right, "\mathsf{Min}^\ell"'] & \mathsf{Det}^r(\mathcal{N}^R) \arrow[r, "\mathsf{Det}^\ell"] & \mathsf{Det}^\ell(\mathsf{Det}^r(\mathcal{N}^R))
\end{tikzcd}$$

The upper part of the diagram follows from Theorem 14 (f). Both squares of the diagram follow from Theorem 14 (e), which states that $\mathsf{Det}^\ell(\mathcal{N})$ is isomorphic to $(\mathsf{Det}^r(\mathcal{N}^R))^R$. Finally, the bottom curved arc follows from Theorem 14 (d). Incidentally, the diagram shows a new relation which follows from the left-right dualities between $\sim^\ell_L$ and $\sim^r_L$, and $\sim^\ell_\mathcal{N}$ and $\sim^r_\mathcal{N}$: $\mathsf{Min}^\ell(\mathcal{L}(\mathcal{N}^R))$ is isomorphic to $\mathsf{Det}^\ell(\mathsf{Det}^r(\mathcal{N}^R))$.

Figure 1 Relations between the constructions $\mathsf{Det}^\ell, \mathsf{Det}^r, \mathsf{Min}^\ell$ and Min^r. Note that constructions Min^r and Min^ℓ are applied to the language generated by the automaton in the origin of the labeled arrow, while constructions Det^r and Det^ℓ are applied directly to the automaton.

Proof.

$$\begin{aligned}
P_{\sim^r_L}(W^\mathcal{N}_{I,q}) = \quad & [\text{By definition of } P_{\sim^r_L}] \\
\{w \in \Sigma^* \mid \exists u \in W^\mathcal{N}_{I,q},\ w^{-1}L = u^{-1}L\} = \quad & [\text{Since } \sim^r_L = \sim^r_\mathcal{N}] \\
\{w \in \Sigma^* \mid \exists u \in W^\mathcal{N}_{I,q},\ \mathrm{post}^\mathcal{N}_w(I) = \mathrm{post}^\mathcal{N}_u(I)\} \subseteq \quad & [u \in W^\mathcal{N}_{I,q} \iff q \in \mathrm{post}^\mathcal{N}_u(I)] \\
\{w \in \Sigma^* \mid q \in \mathrm{post}^\mathcal{N}_w(I)\} = \quad & [\text{By definition of } W^\mathcal{N}_{I,q}] \\
W^\mathcal{N}_{I,q} \quad . &
\end{aligned}$$

By reflexivity of $\sim^r_L$, we conclude that $P_{\sim^r_L}(W^\mathcal{N}_{I,q}) = W^\mathcal{N}_{I,q}$. ◀

▶ **Theorem 16.** *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA with $L = \mathcal{L}(\mathcal{N})$. Then $\mathsf{Det}^r(\mathcal{N})$ is the minimal DFA for L iff $\forall q \in Q,\ P_{\sim^r_L}(W^\mathcal{N}_{I,q}) = W^\mathcal{N}_{I,q}$.*

Proof. Assume $\mathsf{Det}^r(\mathcal{N})$ is minimal. Then $P_{\sim^r_\mathcal{N}}(u) = P_{\sim^r_L}(u)$ for all $u \in \Sigma^*$, i.e. $\sim^r_L = \sim^r_\mathcal{N}$. It follows from Lemma 15 that $P_{\sim^r_L}(W^\mathcal{N}_{I,q}) = W^\mathcal{N}_{I,q}$.

Now, assume that $P_{\sim^r_L}(W^\mathcal{N}_{I,q}) = W^\mathcal{N}_{I,q}$, for each $q \in Q$. Then, for every $u \in \Sigma^*$,

$$P_{\sim^r_\mathcal{N}}(u) = \bigcap_{q \in \mathrm{post}^\mathcal{N}_u(I)} W^\mathcal{N}_{I,q} \cap \bigcap_{q \notin \mathrm{post}^\mathcal{N}_u(I)} (W^\mathcal{N}_{I,q})^c = \bigcap_{q \in \mathrm{post}^\mathcal{N}_u(I)} P_{\sim^r_L}(W^\mathcal{N}_{I,q}) \cap \bigcap_{q \notin \mathrm{post}^\mathcal{N}_u(I)} (P_{\sim^r_L}(W^\mathcal{N}_{I,q}))^c$$

where the first equality follows by rewriting $P_{\sim^r_\mathcal{N}}(u) = \{v \in \Sigma^* \mid \mathrm{post}^\mathcal{N}_u(I) = \mathrm{post}^\mathcal{N}_v(I)\}$ with universal quantifiers, hence intersections, and the last equality follows from the initial assumption $P_{\sim^r_L}(W^\mathcal{N}_{I,q}) = W^\mathcal{N}_{I,q}$.

It follows that $P_{\sim^r_\mathcal{N}}(u)$ is a *union* of blocks of $P_{\sim^r_L}$. Recall that $\sim^r_L$ induces the coarsest right congruence such that $P_{\sim^r_L}(L) = L$ (Lemma 10). Since $\sim^r_\mathcal{N}$ is a right congruence satisfying $P_{\sim^r_\mathcal{N}}(L) = L$ then $P_{\sim^r_\mathcal{N}} \preccurlyeq P_{\sim^r_L}$. Therefore, $P_{\sim^r_\mathcal{N}}(u)$ necessarily corresponds to one single block of $P_{\sim^r_L}$, namely, $P_{\sim^r_L}(u)$. Since $P_{\sim^r_\mathcal{N}}(u) = P_{\sim^r_L}(u)$ for each $u \in \Sigma^*$, we conclude that $\mathsf{Det}^r(\mathcal{N}) = \mathsf{Min}^r(L)$. ◀

5.1 Double-reversal Method

In this section we give a simple proof of the well-known double-reversal minimization algorithm of Brzozowski [4] using Theorem 16. Note that, since $\mathsf{Det}^r(\mathcal{N})$ is isomorphic to $\mathcal{N}^D$ by Theorem 14 (c), the following result coincides with that of Brzozowski.

▶ **Theorem 17** ([4])**.** *Let $\mathcal{N}$ be an NFA. Then $\mathsf{Det}^r((\mathsf{Det}^r(\mathcal{N}^R))^R)$ is isomorphic to the minimal DFA for $\mathcal{L}(\mathcal{N})$.*

Proof. Let $L = \mathcal{L}(\mathcal{N})$. By definition, $\mathcal{N}' = (\mathsf{Det}^r(\mathcal{N}^R))^R$ is a co-DFA and, therefore, satisfies the condition on the right-hand side of Equation (6). It follows from Lemma 12 that $\sim^r_L = \sim^r_{\mathcal{N}'}$ which, by Lemma 15 and Theorem 16, implies that $\mathsf{Det}^r(\mathcal{N}')$ is minimal. ◀

Note that Theorem 17 can be inferred from Figure 1 by following the path starting at $\mathcal{N}$, labeled with $R - \mathsf{Det}^r - R - \mathsf{Det}^r$ and ending in $\mathsf{Min}^r(\mathcal{L}(\mathcal{N}))$.

5.2 Generalization of the Double-reversal Method

Brzozowski and Tamm [5] generalized the double-reversal algorithm by defining a necessary and sufficient condition on an NFA which guarantees that the determinized automaton is minimal. They introduced the notion of *atomic* NFA and showed that $\mathcal{N}^D$ is minimal iff $\mathcal{N}^R$ is atomic. We shall show that this result is equivalent to Theorem 16 due to the left-right duality between the language-based equivalences (Lemma 8).

▶ **Definition 18** (Atom [5])**.** *Let L be a regular language L. Let $\{K_i \mid 0 \leq i \leq n-1\}$ be the set of left quotients of L. An* atom *is any non-empty intersection of the form $\widetilde{K_0} \cap \widetilde{K_1} \cap \ldots \cap \widetilde{K_{n-1}}$, where each $\widetilde{K_i}$ is either K_i or K_i^c.*

This notion of atom coincides with that of equivalence class for the left language-based congruence $\sim^\ell_L$. This was first noticed by Iván [12].

▶ **Lemma 19.** *Let L be a regular language. Then for every $u \in \Sigma^*$,*

$$P_{\sim^\ell_L}(u) = \bigcap_{\substack{u \in w^{-1}L \\ w \in \Sigma^*}} w^{-1}L \;\cap \bigcap_{\substack{u \notin w^{-1}L \\ w \in \Sigma^*}} (w^{-1}L)^c \ .$$

▶ **Definition 20** (Atomic NFA [5])**.** *An NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ is* atomic *iff for every state $q \in Q$, the right language $W^{\mathcal{N}}_{q,F}$ is a union of atoms of $\mathcal{L}(\mathcal{N})$.*

It follows from Lemma 19 that the set of atoms of a language L corresponds to the partition $P_{\sim^\ell_L}$. Therefore, a set $S \subseteq \Sigma^*$ is a union of atoms iff $P_{\sim^\ell_L}(S) = S$. This property, together with Definition 20, shows that an NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ with $L = \mathcal{L}(\mathcal{N})$ is *atomic* iff

$$\forall q \in Q,\; P_{\sim^\ell_L}(W^{\mathcal{N}}_{q,F}) = W^{\mathcal{N}}_{q,F} \ . \tag{7}$$

We are now in condition to give an alternative proof of the generalization of Brzozowski and Tamm [5] relying on Theorem 16.

▶ **Lemma 21.** *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA with $L = \mathcal{L}(\mathcal{N})$. Then $\mathcal{N}^R$ is atomic iff $\mathsf{Det}^r(\mathcal{N})$ is the minimal DFA for L.*

Proof. Let $\mathcal{N}^R = (Q, \Sigma, \delta_r, F, I)$ and $L^R = \mathcal{L}(\mathcal{N}^R)$. Then,

$$\begin{aligned}
\forall q \in Q,\; P_{\sim^\ell_{L^R}}(W^{\mathcal{N}^R}_{q,I}) = W^{\mathcal{N}^R}_{q,I} &\iff && [\text{By } A = B \Leftrightarrow A^R = B^R] \\
\forall q \in Q,\; \left(P_{\sim^\ell_{L^R}}(W^{\mathcal{N}^R}_{q,I})\right)^R = \left(W^{\mathcal{N}^R}_{q,I}\right)^R &\iff && [\text{By } u \sim^\ell_L v \Leftrightarrow u^R \sim^r_{L^R} v^R] \\
\forall q \in Q,\; P_{\sim^r_L}\left(\left(W^{\mathcal{N}^R}_{q,I}\right)^R\right) = \left(W^{\mathcal{N}^R}_{q,I}\right)^R &\iff && [\text{By } \left(W^{\mathcal{N}^R}_{q,I}\right)^R = W^{\mathcal{N}}_{I,q}] \\
\forall q \in Q,\; P_{\sim^r_L}(W^{\mathcal{N}}_{I,q}) = W^{\mathcal{N}}_{I,q} \ . &&&
\end{aligned}$$

It follows from Theorem 16 that $\mathsf{Det}^r(\mathcal{N})$ is minimal. ◀

We conclude this section by collecting all the conditions described so far that guarantee that determinizing an automaton yields the minimal DFA.

▶ **Corollary 22.** *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA with $L = \mathcal{L}(\mathcal{N})$. The following are equivalent:*

(a) $\mathsf{Det}^r(\mathcal{N})$ *is minimal.*

(b) $\sim^r_{\mathcal{N}} = \sim^r_L$.

(c) $\forall u, v \in \Sigma^*,\ W^{\mathcal{N}}_{\mathrm{post}^{\mathcal{N}}_u(I),F} = W^{\mathcal{N}}_{\mathrm{post}^{\mathcal{N}}_v(I),F} \Leftrightarrow \mathrm{post}^{\mathcal{N}}_u(I) = \mathrm{post}^{\mathcal{N}}_v(I)$.

(d) $\forall q \in Q,\ P_{\sim^r_L}(W^{\mathcal{N}}_{I,q}) = W^{\mathcal{N}}_{I,q}$.

(e) $\mathcal{N}^R$ *is atomic.*

5.3 Moore's Algorithm

Given a DFA $\mathcal{D}$, Moore [14] builds the minimal DFA for the language $L = \mathcal{L}(\mathcal{D})$ by removing unreachable states from $\mathcal{D}$ and then performing a stepwise refinement of an initial partition of the set of reachable states of $\mathcal{D}$. Since we are interested in the refinement step, in what follows we assume that all DFAs have no unreachable states. In this section, we will describe Moore's state-partition $\mathcal{Q}^{\mathcal{D}}$ and the right-language-based partition $P_{\sim^r_L}$ as greatest fixpoint computations and show that there exists an isomorphism between the two at each step of the fixpoint computation. In fact, this isomorphism shows that Moore's DFA M satisfies $P_{\sim^r_L}(W^M_{I,q}) = W^M_{I,q}$ for every state q. Thus, by Theorem 16, M is isomorphic to $\mathsf{Min}^r(\mathcal{L}(\mathcal{D}))$.

First, we give Moore's algorithm which computes the state-partition that is later used to define Moore's DFA.

Moore's Algorithm Algorithm for constructing Moore's partition.

```
Data: DFA $\mathcal{D} = \langle Q, \Sigma, \delta, I, F \rangle$ with $L = \mathcal{L}(\mathcal{D})$.
Result: $\mathcal{Q}^{\mathcal{D}} \in Part(Q)$.
$\mathcal{Q}^{\mathcal{D}} := \{F, F^c\}$, $\mathcal{Q}' := \varnothing$;
while $\mathcal{Q}^{\mathcal{D}} \neq \mathcal{Q}'$ do
    $\mathcal{Q}' := \mathcal{Q}^{\mathcal{D}}$;
    forall $a \in \Sigma$ do
        $\mathcal{Q}_a := \curlywedge_{p \in \mathcal{Q}^{\mathcal{D}}} \{\mathrm{pre}^{\mathcal{D}}_a(p), (\mathrm{pre}^{\mathcal{D}}_a(p))^c\}$;
    $\mathcal{Q}^{\mathcal{D}} := \mathcal{Q}^{\mathcal{D}} \curlywedge \curlywedge_{a \in \Sigma} \mathcal{Q}_a$;
return $\mathcal{Q}^{\mathcal{D}}$;
```

▶ **Definition 23** (Moore's DFA)**.** *Let $\mathcal{D} = (Q, \Sigma, \delta, I, F)$ be a DFA, and let $\mathcal{Q}^{\mathcal{D}}$ be the partition of Q built by using Moore's algorithm.* Moore's DFA *for $\mathcal{L}(\mathcal{D})$ is $M = (Q^M, \Sigma, \delta^M, I^M, F^M)$ where $Q^M = \mathcal{Q}^{\mathcal{D}}$, $I^M = \{\mathcal{Q}^{\mathcal{D}}(q) \mid q \in I\}$, $F^M = \{\mathcal{Q}^{\mathcal{D}}(q) \mid q \in F\}$ and, for each $S, S' \in Q^M$ and $a \in \Sigma$, we have that $\delta^M(S, a) = S'$ iff $\exists q \in S, q' \in S'$ with $\delta(q, a) = q'$.*

Next, we describe Moore's state-partition $\mathcal{Q}^{\mathcal{D}}$ and the right-language-based partition $P_{\sim^r_L}$ as greatest fixpoint computations and show that there exists an isomorphism between the two at each step of the fixpoint computation.

▶ **Definition 24** (Moore's state-partition)**.** *Let $\mathcal{D} = (Q, \Sigma, \delta, I, F)$ be a DFA. Define* Moore's state-partition w.r.t. *$\mathcal{D}$, denoted by $\mathcal{Q}^{\mathcal{D}}$, as follows.*

$$\mathcal{Q}^{\mathcal{D}} \stackrel{\mathrm{def}}{=} \mathrm{gfp}(\lambda X. \curlywedge_{a \in \Sigma, S \in X} \{\mathrm{pre}_a(S), (\mathrm{pre}_a(S))^c\} \curlywedge \{F, F^c\}) \ .$$

On the other hand, by Theorem 14 (b), each state of the minimal DFA for L corresponds to an equivalence class of $\sim^r_L$. These equivalence classes can be defined in terms of non-empty intersections of complemented or uncomplemented right quotients of L.

▶ **Lemma 25.** *Let L be a regular language. Then, for every $u \in \Sigma^*$,*

$$P_{\sim^r_L}(u) = \bigcap_{\substack{u \in Lw^{-1} \\ w \in \Sigma^*}} Lw^{-1} \cap \bigcap_{\substack{u \notin Lw^{-1} \\ w \in \Sigma^*}} (Lw^{-1})^c \ .$$

It follows from Lemma 25 that $P_{\sim^r_L} = \curlywedge_{w \in \Sigma^*} \{Lw^{-1}, (Lw^{-1})^c\}$, for every regular language L . Thus, $P_{\sim^r_L}$ can also be obtained as a greatest fixpoint computation as follows.

▶ **Lemma 26.** *Let L be a regular language. Then*

$$P_{\sim^r_L} = \mathrm{gfp}(\lambda X. \bigcurlywedge_{a \in \Sigma, B \in X} \{Ba^{-1}, (Ba^{-1})^c\} \curlywedge \{L, L^c\}) \ . \tag{8}$$

The following result shows that, given a DFA $\mathcal{D}$ with $L = \mathcal{L}(\mathcal{D})$, there exists a partition isomorphism between $\mathcal{Q}^{\mathcal{D}}$ and $P_{\sim^r_L}$ at each step of the fixpoint computations given in Definition 24 and Lemma 26 respectively.

▶ **Theorem 27.** *Let $\mathcal{D} = (Q, \Sigma, \delta, I, F)$ be a DFA with $L = \mathcal{L}(\mathcal{D})$ and let $\varphi : \wp(Q) \to \wp(\Sigma^*)$ be a function defined by $\varphi(S) \overset{\text{def}}{=} W^{\mathcal{D}}_{I,S}$. Let $\mathcal{Q}^{\mathcal{D}(n)}$ and $P^{(n)}_{\sim^r_L}$ be the n-th step of the fixpoint computation of $\mathcal{Q}^{\mathcal{D}}$ (Definition 24) and $P_{\sim^r_L}$ (Lemma 26), respectively. Then, φ is an isomorphism between $\mathcal{Q}^{\mathcal{D}(n)}$ and $P^{(n)}_{\sim^r_L}$ for each $n \geq 0$.*

Proof. In order to show that φ is a partition isomorphism, it suffices to prove that φ is a bijective mapping between the partitions. We first show that $\varphi(\mathcal{Q}^{\mathcal{D}(n)}) = P^{(n)}_{\sim^r_L}$, for every $n \geq 0$. Thus, the mapping φ is surjective. Secondly, we show that φ is an injective mapping from $\mathcal{Q}^{\mathcal{D}(n)}$ to $P^{(n)}_{\sim^r_L}$. Therefore, we conclude that φ is a bijection.

To show that $\varphi(\mathcal{Q}^{\mathcal{D}(n)}) = P^{(n)}_{\sim^r_L}$, for each $n \geq 0$, we proceed by induction.

- *Base case:* By definition, $\mathcal{Q}^{\mathcal{D}(0)} = \{F, F^c\}$ and $P^{(0)}_{\sim^r_L} = \{L, L^c\}$. Since $\mathcal{D}$ is deterministic (and complete), it follows that $\varphi(F) = W^{\mathcal{D}}_{I,F} = L$ and $\varphi(F^c) = W^{\mathcal{D}}_{I,F^c} = L^c$.
- *Inductive step:* Before proceeding with the inductive step, we show that the following equations hold for each $a, b \in \Sigma$ and $S, S_i, S_j \in \mathcal{Q}^{\mathcal{D}(n)}$ with $n \geq 0$:

$$\varphi(\mathrm{pre}_a(S)^c) = ((W^{\mathcal{D}}_{I,S})a^{-1})^c \tag{9}$$

$$\varphi(\mathrm{pre}_a(S_i) \cap \mathrm{pre}_b(S_j)) = (W^{\mathcal{D}}_{I,S_i})a^{-1} \cap (W^{\mathcal{D}}_{I,S_j})b^{-1} \ . \tag{10}$$

For each $S \in \mathcal{Q}^{\mathcal{D}(n)}$ and $a \in \Sigma$ we have that:

$$\begin{aligned}
\varphi(\mathrm{pre}_a(S)^c) = {} & \quad [\text{By definition of } \varphi] \\
W^{\mathcal{D}}_{I,\mathrm{pre}_a(S)^c} = {} & \quad [I = \{q_0\} \text{ and def. of } W^{\mathcal{D}}_{I,\mathrm{pre}_a(S)^c}] \\
\{w \in \Sigma^* \mid \exists q \in \mathrm{pre}_a(S)^c,\ q = \hat{\delta}(q_0, w)\} = {} & \quad [\mathcal{D} \text{ is deterministic and complete}] \\
\{w \in \Sigma^* \mid \exists q \in \mathrm{pre}_a(S),\ q = \hat{\delta}(q_0, w)\}^c = {} & \quad [\text{By definition of } \mathrm{pre}_a(S)] \\
\{w \in \Sigma^* \mid \exists q \in S,\ q = \hat{\delta}(q_0, wa)\}^c = {} & \quad [\text{By definition of } (W^{\mathcal{D}}_{I,S})a^{-1}] \\
((W^{\mathcal{D}}_{I,S})a^{-1})^c \ . &
\end{aligned}$$

Therefore Equation (9) holds at each step of the fixpoint computation. Consider now Equation (10). Let $S_i, S_j \in \mathcal{Q}^{\mathcal{D}(n)}$. Then,

$$
\begin{aligned}
\varphi(\mathrm{pre}_a(S_i) \cap \mathrm{pre}_b(S_j)) = \quad & [\text{By Def. } \varphi] \\
W^{\mathcal{D}}_{I,(\mathrm{pre}_a(S_i)\cap\mathrm{pre}_b(S_j))} = \quad & [I = \{q_0\} \text{ and def. } W_{I,S}] \\
\{w \in \Sigma^* \mid \exists q \in \mathrm{pre}_a(S_i) \cap \mathrm{pre}_b(S_j), q = \hat{\delta}(q_0, w)\} = \quad & [\text{By Def. of } \cap] \\
\{w \in \Sigma^* \mid \exists q \in \mathrm{pre}_a(S_i),\ q \in \mathrm{pre}_b(S_j), q = \hat{\delta}(q_0, w)\} = \quad & [\mathcal{D} \text{ is deterministic}] \\
W^{\mathcal{D}}_{I,\mathrm{pre}_a(S_i)} \cap W^{\mathcal{D}}_{I,\mathrm{pre}_b(S_j)} = \quad & [\text{By Def. of } (W^{\mathcal{D}}_{I,S})a^{-1}] \\
(W^{\mathcal{D}}_{I,S_i})a^{-1} \cap (W_{I,S_j})b^{-1} \quad . &
\end{aligned}
$$

Therefore Equation (10) holds at each step of the fixpoint computation.
Let us assume that $\varphi\left(\mathcal{Q}^{\mathcal{D}(n)}\right) = P^{(n)}_{\sim^r_L}$ for every $n \leq k$ with $k > 0$. Then,

$$
\begin{aligned}
\varphi\big(\mathcal{Q}^{\mathcal{D}(k+1)}\big) = & \ [\text{By Def. 24 with } X = \mathcal{Q}^{\mathcal{D}(k)}] \\
\varphi\Big(\bigcurlywedge_{a\in\Sigma, S\in X} \{\mathrm{pre}_a(S), \mathrm{pre}_a(S)^c\} \curlywedge \{F, F^c\}\Big) = & \ [\text{By Eqs. (9), (10) and def. of } \bigcurlywedge] \\
\bigcurlywedge_{\substack{a\in\Sigma \\ \varphi(S)\in\varphi(X)}} \{(W^{\mathcal{D}}_{I,S})a^{-1}, ((W^{\mathcal{D}}_{I,S})a^{-1})^c\} \curlywedge \{L, L^c\} = & \ [\text{By induction hypothesis, } \varphi(X) = P^{(k)}_{\sim^r_L}] \\
\bigcurlywedge_{a\in\Sigma, B\in X'} \{Ba^{-1}, (Ba^{-1})^c\} \curlywedge \{L, L^c\} = & \ [\text{By Lemma 26 with } X' = P^{(k)}_{\sim^r_L}] \\
P^{(k+1)}_{\sim^r_L} \quad . &
\end{aligned}
$$

Finally, since $\mathcal{D}$ is a DFA then, for each $S_i, S_j \in \mathcal{Q}^{\mathcal{D}(n)} (n \geq 0)$ with $S_i \neq S_j$ we have that $W^{\mathcal{D}}_{I,S_i} \neq W^{\mathcal{D}}_{I,S_j}$, i.e., $\varphi(S_i) \neq \varphi(S_j)$. Therefore, φ is an injective mapping. ◀

▶ **Corollary 28.** *Let $\mathcal{D}$ be a DFA with $L = \mathcal{L}(\mathcal{D})$. Let $\mathcal{Q}^{\mathcal{D}(n)}$ and $P^{(n)}_{\sim^r_L}$ be the n-th step of the fixpoint computation of $\mathcal{Q}^{\mathcal{D}}$ and $P_{\sim^r_L}$ respectively. Then, for each $n \geq 0$,*

$$P^{(n)}_{\sim^r_L}(W^{\mathcal{D}}_{I,S}) = W^{\mathcal{D}}_{I,S} \ \ , \textit{ for each } S \in \mathcal{Q}^{\mathcal{D}(n)} \ \ .$$

It follows that Moore's DFA M, whose set of states corresponds to the state-partition at the end of the execution of Moore's algorithm, satisfies that $\forall q \in Q^M$, $P_{\sim^r_L}(W^M_{I,q}) = W^M_{I,q}$ with $L = \mathcal{L}(M)$. By Theorem 16, we have that $\mathsf{Det}^r(M)(= M$, since M is a DFA) is minimal.

▶ **Theorem 29.** *Let $\mathcal{D}$ be a DFA and M be Moore's DFA for $\mathcal{L}(\mathcal{D})$ as in Definition 23. Then, M is isomorphic to $\mathsf{Min}^r(\mathcal{L}(\mathcal{D}))$.*

Finally, recall that Hopcroft [10] defined a DFA minimization algorithm which offers better performance than Moore's. The ideas used by Hopcroft can be adapted to our framework to devise a new algorithm for computing $P_{\sim^r_L}$. However, by doing so, we could not derive a better explanation than the one provided by Berstel et al. [2].

6 Related Work and Conclusions

Brzozowski and Tamm [5] showed that every regular language defines a unique NFA, which they call *átomaton*. The átomaton is built upon the minimal DFA $\mathcal{N}^{DM}$ for the language, defining its states as non-empty intersections of complemented or uncomplemented right languages of $\mathcal{N}^{DM}$, i.e., the atoms of the language. They also observed that the atoms correspond to intersections of complemented or uncomplemented left quotients of the language. Then they proved that the átomaton is isomorphic to the reverse automaton of the minimal deterministic DFA for the reverse language.

Intuitively, the construction of the átomaton based on the right languages of the minimal DFA corresponds to $\mathsf{Det}^{\ell}(\mathcal{N}^{DM})$, while its construction based on left quotients of the language corresponds to $\mathsf{Min}^{\ell}(\mathcal{L}(\mathcal{N}))$.

▶ **Corollary 30.** *Let $\mathcal{N}^{DM}$ be the minimal DFA for a regular language L. Then,*
(a) $\mathsf{Det}^{\ell}(\mathcal{N}^{DM})$ *is isomorphic to the átomaton of L.*
(b) $\mathsf{Min}^{\ell}(L)$ *is isomorphic to the átomaton of L.*

In the same paper, they also defined the notion of *partial átomaton* which is built upon an NFA $\mathcal{N}$. Each state of the partial atomaton is a non-empty intersection of complemented or uncomplemented right languages of $\mathcal{N}$, i.e., union of atoms of the language. Intuitively, the construction of the partial átomaton corresponds to $\mathsf{Det}^{\ell}(\mathcal{N})$.

▶ **Corollary 31.** *Let $\mathcal{N}$ be an NFA. Then,* $\mathsf{Det}^{\ell}(\mathcal{N})$ *is isomorphic to the partial átomaton of $\mathcal{N}$.*

Finally, they also presented a number of results [5, Theorem 3] related to the átomaton $\mathcal{A}$ of a minimal DFA $\mathcal{D}$ with $L = \mathcal{L}(\mathcal{D})$:

1. $\mathcal{A}$ is isomorphic to $\mathcal{D}^{RDR}$.
2. $\mathcal{A}^{R}$ is the minimal DFA for L^{R}
3. $\mathcal{A}^{D}$ is the minimal DFA for L.
4. $\mathcal{A}$ is isomorphic to $\mathcal{N}^{RDMR}$ for every NFA $\mathcal{N}$ accepting L.

All these relations can be inferred from Figure 2 which connects all the automata constructions described in this paper together with the constructions introduced by Brzozowski and Tamm. For instance, property 1 corresponds to the path starting at $\mathcal{N}^{DM}$ (the minimal DFA for $\mathcal{L}(\mathcal{N})$), labeled with $R - \mathsf{Det}^{r} - R$, and ending in the átomaton of $\mathcal{L}(\mathcal{N})$. On the other hand, property 4 corresponds to the path starting at $\mathcal{N}$, labeled with $R - \mathsf{Min}^{r} - R$ and ending in the átomaton of $\mathcal{L}(\mathcal{N})$. Finally, the path starting at $\mathcal{N}$, labeled with $R - \mathsf{Det}^{r} - R$ and ending in the partial átomaton of $\mathcal{N}$ shows that the later is isomorphic to $\mathcal{N}^{RDR}$.

In conclusion, we establish a connection between well-known independent minimization methods through Theorem 16. Given a DFA, the left languages of its states form a partition on words, P, and thus, each left language is identified by a state. Intuitively, Moore's algorithm merges states to enforce the condition of Theorem 16, which results in merging blocks of P that belong to the same Nerode's equivalence class. Note that Hopcroft's partition refinement method [10] achieves the same goal at the end of its execution though, stepwise, the partition computed may differ from Moore's. On the other hand, any co-deterministic NFA satisfies the right-hand side of Equation (6) hence, by Lemma 15, satisfies the condition of Theorem 16. Therefore, the double-reversal method, which essentially determinizes a co-determinized NFA, yields the minimal DFA. Finally, the left-right duality (Lemma 8) of the language-based equivalences shows that the condition of Theorem 16 is equivalent to that of Brzozowski and Tamm [5].

$$\begin{tikzcd}
\mathcal{N} \arrow[r, "\mathsf{Det}^\ell;\, C.31"] \arrow[rr, bend left, "\mathsf{Min}^r;\, T.14(b)"] \arrow[rrr, bend left, "\mathsf{Min}^\ell;\, T.14(d)"] \arrow[d, leftrightarrow, "R"] & \text{Partial átomaton of } \mathcal{N} \arrow[r, "\mathsf{Det}^r;\, T.14(c)"] \arrow[d, leftrightarrow, "R"] & \mathcal{N}^{DM} \arrow[r, "\mathsf{Det}^\ell;\, C.30(a)"] \arrow[d, leftrightarrow, "R"] & \text{Átomaton of } \mathcal{L}(\mathcal{N}) \arrow[l, bend right, "\mathsf{Det}^r;\, T.14(f)"'] \arrow[d, leftrightarrow, "R"] \\
\mathcal{N}^R \arrow[r, "\mathsf{Det}^r;\, T.14(c)"] \arrow[rr, bend right, "\mathsf{Min}^\ell;\, C.30(b)"'] \arrow[rrr, bend right, "\mathsf{Min}^r;\, T.14(b)"'] & \mathcal{N}^{RD} \arrow[r, "\mathsf{Det}^\ell;\, T.14(e)"] & \text{Átomaton of } \mathcal{L}(\mathcal{N}^R) \arrow[r, "\mathsf{Det}^r;\, T.14(c)"] & \mathcal{N}^{RDM} \arrow[l, bend right, "\mathsf{Det}^\ell;\, C.30(a)"']
\end{tikzcd}$$

Figure 2 Extension of the diagram of Figure 1 including the átomaton and the partial átomaton. Recall that $\mathcal{N}^{DM}$ is the minimal DFA for $\mathcal{L}(\mathcal{N})$. The results referenced in the labels are those justifying the output of the operation.

Some of these connections have already been studied in order to offer a better understanding of Brzozowski's double-reversal method [1, 3, 7, 16]. In particular, Adámek et al. [1] and Bonchi et al. [3] offer an alternative view of minimization and determinization methods in a uniform way from a category-theoretical perspective. In contrast, our work revisits these well-known minimization techniques relying on simple language-theoretical notions.

References

1 Jirí Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius, and Alexandra Silva. A Coalgebraic Perspective on Minimization and Determinization. In *FoSSaCS*, volume 7213 of *Lecture Notes in Computer Science*, pages 58–73. Springer, 2012.

2 Jean Berstel, Luc Boasson, Olivier Carton, and Isabelle Fagnot. Minimization of automata, 2010. `arXiv:1010.5318`.

3 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.

4 Janusz A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. *Mathematical Theory of Automata*, 12(6):529–561, 1962.

5 Janusz A. Brzozowski and Hellis Tamm. Theory of átomata. *Theor. Comput. Sci.*, 539:13–27, 2014.

6 Julius R. Büchi. *Finite Automata, their Algebras and Grammars - Towards a Theory of Formal Expressions*. Springer, 1989.

7 Jean-Marc Champarnaud, Ahmed Khorsi, and Thomas Paranthoën. Split and join for minimizing: Brzozowski's algorithm. In *Stringology*, pages 96–104. Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, 2002.

8 Aldo de Luca and Stefano Varricchio. *Finiteness and Regularity in Semigroups and Formal Languages*. Springer Publishing Company, Incorporated, 1st edition, 2011.

9 Pierre Ganty, Elena Gutiérrez, and Pedro Valero. A Congruence-based Perspective on Automata Minimization Algorithms (extended version), 2019. `arXiv:1906.06194`.

10 John E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.

11 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation - (2. ed.)*. Addison-Wesley-Longman, 2001.

12 Szabolcs Iván. Complexity of atoms, combinatorially. *Inf. Process. Lett.*, 116(5):356–360, 2016.

13 Bakhadyr Khoussainov and Anil Nerode. *Automata Theory and Its Applications*. Birkhauser Boston, Inc., Secaucus, NJ, USA, 2001.

14 Edward F. Moore. Gedanken-experiments on sequential machines. *Automata studies*, 23(1):60–60, 1956.

15 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

16 Manuel Vázquez de Parga, Pedro García, and Damián López. A polynomial double reversal minimization algorithm for deterministic finite automata. *Theor. Comput. Sci.*, 487:17–22, 2013.

Finding Optimal Solutions With Neighborly Help

Elisabet Burjons
Department of Computer Science, ETH Zürich, Universitätstrasse 6, CH-8092 Zürich, Switzerland
eburjons@inf.ethz.ch

Fabian Frei
Department of Computer Science, ETH Zürich, Universitätstrasse 6, CH-8092 Zürich, Switzerland
fabian.frei@inf.ethz.ch

Edith Hemaspaandra
Department of Computer Science, Rochester Institute of Technology, Rochester, NY 14623, USA
eh@cs.rit.edu

Dennis Komm
Department of Computer Science, ETH Zürich, Universitätstrasse 6, CH-8092 Zürich, Switzerland
dennis.komm@inf.ethz.ch

David Wehner
Department of Computer Science, ETH Zürich, Universitätstrasse 6, CH-8092 Zürich, Switzerland
david.wehner@inf.ethz.ch

Abstract

Can we efficiently compute optimal solutions to instances of a hard problem from optimal solutions to neighboring (i.e., locally modified) instances? For example, can we efficiently compute an optimal coloring for a graph from optimal colorings for all one-edge-deleted subgraphs? Studying such questions not only gives detailed insight into the structure of the problem itself, but also into the complexity of related problems; most notably graph theory's core notion of critical graphs (e.g., graphs whose chromatic number decreases under deletion of an arbitrary edge) and the complexity-theoretic notion of minimality problems (also called criticality problems, e.g., recognizing graphs that become 3-colorable when an arbitrary edge is deleted).

We focus on two prototypical graph problems, Colorability and Vertex Cover. For example, we show that it is NP-hard to compute an optimal coloring for a graph from optimal colorings for *all* its one-vertex-deleted subgraphs, and that this remains true even when optimal solutions for *all* one-edge-deleted subgraphs are given. In contrast, computing an optimal coloring from all (or even just two) one-edge-added supergraphs is in P. We observe that Vertex Cover exhibits a remarkably different behavior, demonstrating the power of our model to delineate problems from each other more precisely on a structural level.

Moreover, we provide a number of new complexity results for minimality and criticality problems. For example, we prove that Minimal-3-UnColorability is complete for DP (differences of NP sets), which was previously known only for the more amenable case of deleting vertices rather than edges. For Vertex Cover, we show that recognizing β-vertex-critical graphs is complete for Θ_2^{p} (parallel access to NP), obtaining the first completeness result for a criticality problem for this class.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Complexity classes

Keywords and phrases Critical Graphs, Computational Complexity, Structural Self-Reducibility, Minimality Problems, Colorability, Vertex Cover, Satisfiability, Reoptimization, Advice

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.78

Related Version A full version of this paper, which includes the appendices, is published on arXiv [3] and available at `https://arxiv.org/abs/1906.10078`.

Funding *Edith Hemaspaandra*: Research done in part while on sabbatical at ETH Zürich.

Acknowledgements We thank the anonymous referees and Hans-Joachim Böckenhauer, Rodrigo R. Gumucio Escobar, Lane Hemaspaandra, Juraj Hromkovič, Rastislav Královič, Richard Královič, Xavier Muñoz, Martin Raszyk, Peter Rossmanith, Walter Unger, and Koichi Wada for helpful comments and discussions.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 78; pp. 78:1–78:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction and Related Work

In Subsection 1.1, we introduce and motivate our new model, which we then compare and contrast to related notions in Subsection 1.2. Finally, we present in Subsection 1.3 an overview of our most interesting results and place them into the context of the wider literature.

1.1 Our Model

In view of the almost complete absence of progress in the question of P versus NP, it is natural to wonder just how far and in what way these sets may differ. For example, how much additional information enables us to design an algorithm that solves an otherwise NP-hard problem in polynomial time? We are specifically interested in the case where this additional information takes the form of optimal solutions to neighboring (i.e., locally modified) instances. This models situations such as that of a newcomer who may ask experienced peers for advice on how to solve a difficult problem, for instance finding an optimal work route. Similar circumstances arise when new servers join a computer network. Formally, we consider the following oracle model: An algorithm may, on any given input, repeatedly select an arbitrary instance neighboring the given one and query the oracle for an optimal solution to it. Occasionally, it will be interesting to limit the number of queries that we grant the algorithm. In general, we do not impose such a restriction, however.

What precisely constitutes a local modification and thus a neighbor depends on the specific problem, of course. We examine the prototypical graph problems Colorability and Vertex Cover, considering the following four local modifications, which are arguably the most natural choices: deleting an edge, adding an edge, deleting a vertex (including adjacent edges), and adding a vertex (including an arbitrary, possibly empty, set of edges from the added vertex to the existing ones). For example, we ask whether there is a polynomial-time algorithm that computes a minimum vertex cover for an input graph G if it has access to minimum vertex covers for all one-edge-deleted subgraphs of G. We will show that questions of this sort are closely connected to and yet clearly distinct from research in other areas, in particular the study of critical graphs, minimality problems, self-reducibility, and reoptimization.

1.2 Related Concepts

Criticality. The notion of criticality was introduced into the field of graph theory by Dirac [7] in 1952 in the context of Colorability with respect to vertex deletion. Thirty years later, Wessel [19] generalized the concept to arbitrary graph properties and modification operations. Nevertheless, Colorability has remained a central focus of the extensive research on critical graphs. Indeed, a graph G is called *critical* without any further specification if it is χ-critical under edge deletion, that is, if its chromatic number $\chi(G)$ (the number of colors used in an optimal coloring of G) changes when an arbitrary edge is deleted. Besides Colorability, one other problem has received a comparable amount of attention and thorough analysis in three different manifestations: Independent Set, Vertex Cover, and Clique. The corresponding notions are α-criticality, β-criticality, and ω-criticality, where α is the *independence number* (size of a maximum independent set), β is the *vertex cover number* (size of a minimum vertex cover), and ω is the *clique number* (size of a maximum clique). Note that these graph numbers are all monotone – either nondecreasing or nonincreasing – with respect to each of the local modifications examined in this paper.

Minimality. Another strongly related notion is that of minimality problems. An instance is called *minimal* with respect to a property if only the instance itself but none of its neighbors has this property; that is, it inevitably loses the property under the considered local modification. The corresponding minimality problem is to decide whether an instance is minimal in the described sense. For example, a graph G is minimally 3-uncolorable (with respect to edge deletion) if it is not 3-colorable, yet all its one-edge-deleted neighbors are. The minimality problem MINIMAL-3-UNCOLORABILITY is the set of all minimally 3-uncolorable graphs. Note that a graph is critical exactly if it is minimally k-uncolorable for some k.

While minimality problems tend to be in DP (i.e., differences of two NP sets, the second level of the Boolean hierarchy), DP-hardness is so difficult to prove for them that only a few have been shown to be DP-complete so far; see for instance Papadimitriou and Wolfe [14]. Note that the notion of minimality is not restricted to graph problems. Indeed, minimally unsatisfiable formulas figure prominently in many of our proofs.

Auto-Reducibility. Our model provides a refinement of the notion of functional auto-reducibility; see Faliszewski and Ogihara [8]. An algorithm solves a function problem $R \subseteq \Sigma_1^* \times \Sigma_2^*$ if on input $x \in \Sigma_1^*$ it outputs some $y \in \Sigma_2^*$ with $(x, y) \in R$. The problem R is *auto-reducible* if there is a polynomial-time algorithm with unrestricted access to an oracle that provides solutions to all instances except x itself. The task of finding an optimal solution to a given instance is a special kind of function problem. Defining all instances to be neighbors (local modifications) of each other lets the two concepts coincide.

Self-Reducibility. Self-reducibility is auto-reducibility with the additional restriction that the algorithm may query the oracle only on instances that are smaller in a certain way. There are a multitude of definitions of self-reducibility that differ in what exactly is considered to be "smaller," the two seminal ones stemming from Schnorr [15] and from Meyer and Paterson [13]. For Schnorr, an instance is smaller than another one if its encoding input string is strictly shorter. While his definition does allow for functional problems (i.e., more than mere decision problems, in particular the problem of finding an optimal solution), it is too restrictive for self-reducibility to encompass our model since not all neighboring graphs have shorter strings under natural encodings.

Meyer and Paterson are less rigid and allow instead any partial order having short downward chains to determine which instances are considered smaller than the given one.[1] The partial orders induced by deleting vertices, by deleting edges, and by adding edges all have short downward chains. The definition by Meyer and Paterson is thus sufficient for our model to become part of functional self-reducibility for all local modifications considered in this paper but one, namely, the case of adding a vertex, which is too generous a modification to display any particularly interesting behavior.

As an example, consider the graph decision problem COLORABILITY $= \{(G, k) \mid \chi(G) \leq k\}$, which is self-reducible by the following observation. Any graph G with at least two vertices that is not a clique is k-colorable exactly if at least one of the polynomially many graphs that result from merging two non-adjacent vertices in G is k-colorable. This works for the optimization variant of the problem as well. Any optimal coloring of G assigns at

[1] Formally, a partial order is said to *have short downward chains* if the following condition is satisfied: There is a polynomial p such that every chain decreasing with respect to the considered partial order and starting with some string x is shorter than $p(|x|)$ and such that all strings preceding x in that order are bounded in length by $p(|x|)$.

least two vertices the same color, except in the trivial case of G being a clique. An optimal coloring for the graph that has two such vertices merged then yields an optimal coloring for G. This contrasts well with the findings for Colorability's behavior under our new model discussed below.

Reoptimization. Reoptimization examines optimization problems under a model that is tightly connected to ours. The notion of reoptimization was coined by Schäffter [16] and first applied by Archetti et al. [1]. The reoptimization model sets the following task for an optimization problem:

> *Given an instance, an optimal solution to it, and a local modification of this instance, compute an optimal solution to the modified instance.*

The proximity to our model becomes clearer after a change of perspective. We reformulate the reoptimization task by reversing the roles of the given and the modified instance.

> *Given an instance, a local modification of it, and an optimal solution to the modified instance, compute an optimal solution to the original instance.*

Note that this perspective switch flips the definition of local modification; for example, edge deletion turns into edge addition. Aside from this, the task now reads almost identical to that demanded in our model. The sole but crucial difference is that in reoptimization, the neighboring instance and the optimal solution to it are given as part of the input, whereas in our model, the algorithm may select any number of neighboring instances and query the oracle for optimal solutions to them. Even if we limit the number of queries to just one, our model is still more generous since the algorithm is choosing (instead of being given) the neighboring instance to which the oracle will supply an optimal solution. Thus, hardness in our model always implies hardness for reoptimization, but not vice versa. In fact, all problems examined under the reoptimization model so far remain NP-hard. Only for some of them could an improvement of the approximation ratio be achieved after extensive studies, the first discovered examples being TSP under edge-weight changes [4] and addition or deletion of vertices [2]. This stands in stark contrast to the results for our model, as outlined in the next section.

1.3 Results

We shed a new light on two of the most prominent and well-examined graph problems, Colorability and Vertex Cover. Our results come in two different types.

The first type concerns the hardness of the two problems in our model for the most common local modifications; Table 1 summarizes the main results of this type. In addition, Corollaries 2 and 14 show that Satisfiability and Vertex Cover remain NP-hard for any number of queries if the local modification is the deletion of a clause or a triangle, respectively. The results for the vertex-addition columns are trivial since we can just query an optimal solution for the graph with an added isolated vertex; see Theorem 16 in Appendix A [3]. The hardness results for the one-query case all follow from the same simple Theorem 17, variations of which appear in the study of self-reducibility and many other fields; see Appendix B [3]. The findings of Theorems 10, 12, and 19 in Appendix F [3] clearly delineate our model from that in reoptimization, where the NP-hard problems examined in the literature remain NP-hard despite the significant amount of advice in form of the provided optimal solution; see Böckenhauer et al. [5].

Table 1 An overview of our results regarding the hardness of Colorability and Vertex Cover in our model for the most common definitions of a local modification. The v stands for a vertex and the e stands for an edge. The question mark indicates an interesting open problem. The results in the vertex-addition columns are trivial; see Theorem 16 in Appendix A [3]. The NP-hardness results for the 1-query case all follow from rather simple Turing reductions; see Theorem 17 in Appendix B [3].

No. of Queries	Colorability				Vertex Cover			
	Add v	Delete v	Add e	Delete e	Add v	Delete v	Add e	Delete e
1	P	NP-hard	NP-hard	NP-hard	P	NP-hard	NP-hard	NP-hard
2 or more	P [Thm. 16]	NP-hard [Thm. 4]	P [Thm. 10]	NP-hard [Thm. 6]	P [Thm. 16]	P [Thm. 12]	P [Thm. 19]	?

The results of the second type locate criticality problems in relation to the complexity classes DP and Θ_2^{p}. The class Θ_2^{p} was introduced by Wagner [17] and represents the languages that can be decided in polynomial time by an algorithm that has access to an NP oracle under the restriction that all queries are submitted at the same time. The definitions of the classes immediately yield the inclusions $\mathrm{NP} \cup \mathrm{coNP} \subseteq \mathrm{DP} \subseteq \Theta_2^{\mathrm{p}}$.

Papadimitriou and Wolfe [14] have shown that Minimal-UnSat (the set of unsatisfiable formulas that become satisfiable when an arbitrary clause is deleted) is DP-complete. Cai and Meyer [6] built upon this to prove DP-completeness of VertexMinimal-k-UnColorability (the set of graphs that are not k-colorable but become k-colorable when an arbitrary vertex is deleted), for all $k \geq 3$. With Theorems 7 and 8, we were able to extend this result to classes that are analogously defined for the much smaller local modification of edge deletion, which is considered the default setting; namely, we prove DP-completeness of Minimal-k-UnColorability, for all $k \geq 3$.

In Theorem 9, we show that recognizing criticality and vertex-criticality are in Θ_2^{p} and DP-hard. As Joret [12] points out, a construction by Papadimitriou and Wolfe [14] proves the DP-hardness of recognizing β-critical graphs. This problem also lies in Θ_2^{p}, but no finer classification has been achieved so far. In Theorem 15, we show that this problem is in fact Θ_2^{p}-hard, yielding the first known Θ_2^{p}-completeness result for a criticality problem.

2 Preprocessing 3-Sat

Our main technique for proving the nontrivial hardness results in our model is the following: We build in polynomial-time computable solutions for each locally modified problem instance. That way, the solutions to the locally modified problem instances do not give away any information about the instance to be solved. A similar approach is taken in some proofs of DP-completeness for minimality problems. Indeed, we can occasionally combine the proof of DP-hardness with that of the NP-hardness of computing an optimal solution from optimal solutions to locally modified instances. Denote by 3-CNF the set of nonempty CNF-formulas with exactly three distinct literals per clause.[2] We begin by showing in Theorem 1 that there is a reduction from 3-Sat (the set of satisfiable 3-CNF-formulas) to 3-Sat that builds in polynomial-time computable solutions for all one-clause-deleted subformulas of the resulting 3-CNF-formula. At first glance, this very surprising result may seem dangerously close to

[2] This set is often denoted E3-CNF in the literature.

proving P = NP; Corollary 2 will make explicit where the hardness remains. We will then use the reduction of Theorem 1 as a preprocessing step in reductions from 3-Sat to other problems.

▶ **Theorem 1.** *There is a polynomial-time many-one reduction f from* 3-Sat *to* 3-Sat *and a polynomial-time computable function s such that, for every* 3-CNF*-formula Φ and for every clause C in $f(\Phi)$, $s(f(\Phi) - C)$ is a satisfying assignment for $f(\Phi) - C$.*

Proof. Papadimitriou and Wolfe [14, Lemma 1] give a reduction from 3-UnSat to Minimal-UnSat (the set of CNF-formulas that are unsatisfiable but that become satisfiable with the removal of an arbitrary clause). In Appendix C [3], we show how to enhance this reduction such that it has all properties of our theorem. First, we carefully prove that there is a function s that together with the original reduction satisfies all properties of our theorem, except that we may output a formula that is not in 3-CNF. In order to rectify this, we show that the standard reduction from Sat to Sat that decreases the number of literals per clause to at most three maintains all the required properties. The same is then shown for the standard reduction that transforms CNF-formulas with at most three literals per clause into 3-CNF-formulas that have exactly three distinct literals per clause. Combining these three reductions, we can therefore satisfy all requirements of our theorem. ◀

▶ **Corollary 2.** *Computing a satisfying assignment for a* 3-CNF*-formula whose one-clause-deleted subformulas all have a satisfying assignment from these assignments is* NP*-hard.*

Proof. Given a 3-CNF-formula Φ, compute $f(\Phi)$, where f is the reduction from Theorem 1. Now compute $s(f(\Phi) - C)$ for every clause C in $f(\Phi)$ and compute a satisfying assignment for $f(\Phi)$ from these solutions. Use this assignment to determine whether Φ is satisfiable. ◀

3 Colorability

As mentioned in the previous section, the constructions of some DP-completeness results for minimality problems can be adapted to obtain NP-hardness for computing optimal solutions from optimal solutions to locally modified instances. There are remarkably few complexity results for minimality problems; fortunately, however, VertexMinimal-3-UnColorability (the graphs that are not 3-colorable but that are 3-colorable after deleting any vertex)[3] is DP-complete by reduction from Minimal-3-UnSat [6]. We will show how to extract from said reduction a proof of the fact that computing an optimal coloring for a graph from optimal colorings for its one-vertex-deleted subgraphs is NP-hard (Theorem 4). However, the standard notion of criticality is χ-criticality under edge deletion, and the construction by Cai and Meyer [6] does unfortunately not yield the analogous result for deleting edges. This was to be expected, since working with edge deletion is much harder. Surprisingly, however, a targeted modification of the constructed graph allows us to establish, through a far more elaborate case distinction, that computing an optimal coloring for a graph from optimal colorings for its one-edge-deleted subgraphs is NP-hard (Theorem 6) as well as that the related minimality problem Minimal-3-UnColorability is DP-complete (Theorem 7).

[3] It should be noted that VertexMinimal-3-UnColorability is denoted by Minimal-3-UnColorability by Cai and Meyer [6] despite the fact that minimality problems usually refer to the case of edge deletion.

▶ **Lemma 3.** *There is a polynomial-time many-one reduction g from* 3-Sat *to* 3-Colorability *and a polynomial-time computable function* opt *such that, for every* 3-CNF*-formula* Φ *and for every vertex v in $g(\Phi)$,* $\text{opt}(g(\Phi) - v)$ *is an optimal coloring for $g(\Phi) - v$.*

Proof. Given a 3-CNF-formula Φ, let $g(\Phi) = h(f(\Phi))$, where f is the reduction from Theorem 1 and h is the reduction from Minimal-3-UnSat to VertexMinimal-3-UnColorability by Cai and Meyer [6]. Since h also reduces 3-Sat to 3-Colorability [6, Lemma 2.2], so does g. A careful inspection of the reduction g reveals that there is a polynomial-time computable function opt such that, for every vertex v in $g(\Phi)$, $\text{opt}(g(\Phi) - v)$ is a 3-coloring of $g(\Phi) - v$. We can also verify that $g(\Phi) - v$ does not have a 2-coloring, hence $\text{opt}(g(\Phi) - v)$ is an optimal coloring. We do not dive into the details as this lemma immediately follows from the proof of the analogous result for edge deletion (Lemma 5), as explained in Appendix D [3]. ◀

▶ **Theorem 4.** *Computing an optimal coloring for a graph from optimal colorings for its one-vertex-deleted subgraphs is* NP*-hard.*

Proof. Given a 3-CNF-formula Φ, compute $g(\Phi)$, where g is the reduction from Lemma 3, compute $\text{opt}(g(\Phi) - v)$ for every vertex v in $g(\Phi)$, and from these optimal solutions compute one for $g(\Phi)$. This determines whether $g(\Phi)$ is 3-colorable and thus whether Φ is satisfiable. ◀

▶ **Lemma 5.** *There is a polynomial-time many-one reduction g from* 3-Sat *to* 3-Colorability *and a polynomial-time computable function* opt *such that, for every* 3-CNF*-formula* Φ *and for every edge e in $g(\Phi)$,* $\text{opt}(g(\Phi) - e)$ *is an optimal coloring of $g(\Phi) - e$.*

Proof. Given a 3-CNF-formula Φ, let $g(\Phi) = h(f(\Phi)) - e$, where f is the reduction from Theorem 1, h is the reduction to VertexMinimal-3-UnColorability by Cai and Meyer [6], and e is the edge $\{v_c, v_s\}$, with v_c being the unique vertex adjacent to all variable-setting vertices and v_s being the only remaining neighbor vertex of v_c. We prove in detail that g has all the desired properties in Appendix D [3]. See Figure 1 in Appendix D [3] for an example of the construction. ◀

▶ **Theorem 6.** *Computing an optimal coloring for a graph from optimal colorings for its one-edge-deleted subgraphs is* NP*-hard.*

Proof. The same argument as for Theorem 4 can be applied here. ◀

▶ **Theorem 7.** Minimal-3-UnColorability *is* DP*-complete.*

Proof. Membership in DP is immediate, since given a graph $G = (V, E)$, determining whether $G - e$ is 3-colorable for every $e \in E$ is in NP and so is determining whether G is 3-colorable. As for DP-hardness, the argument from the proof of Lemma 5 shows that mapping Φ to $h(\Phi) - \{v_c, v_s\}$, where h is the reduction from Minimal-3-UnSat to VertexMinimal-3-UnColorability by Cai and Meyer [6], gives a reduction from Minimal-3-UnSat to Minimal-3-UnColorability (and to VertexMinimal-3-UnColorability as well). Recall that Minimal-3-UnSat is DP-hard [14]. ◀

Cai and Meyer [6] show DP-completeness for VertexMinimal-k-UnColorability, for all $k \geq 3$. We now prove that the analogous result for deletion of edges holds as well.

▶ **Theorem 8.** Minimal-k-UnColorability *is* DP*-complete, for every $k \geq 3$.*

Proof. Membership in DP is again immediate. To show hardness for $k \geq 4$, we reduce Minimal-3-UnColorability to Minimal-k-UnColorability. We use the construction for deleting vertices [6, Theorem 3.1] and map graph G to $G + K_{k-3}$.[4] Note that $\chi(K_{k-3}) = k-3$ and $\chi(H + H') = \chi(H) + \chi(H')$ for any two graphs H and H'. First suppose $G + K_{k-3}$ is in Minimal-k-UnColorability. Then $G + K_{k-3}$ is not k-colorable, and so G is not 3-colorable. Let e be an edge in G. Then $(G - e) + K_{k-3} = (G + K_{k-3}) - e$ is k-colorable, and thus $G - e$ is 3-colorable. It follows that G is in Minimal-3-UnColorability.

Now suppose G is in Minimal-3-UnColorability. Then $G + K_{k-3}$ is not k-colorable. Let e be an edge in $G + K_{k-3}$. If e is an edge in G, then $G - e$ is 3-colorable and so $(G + K_{k-3}) - e = (G - e) + K_{k-3}$ is k-colorable. If e is an edge in K_{k-3}, then $K_{k-3} - e$ is $(k-4)$-colorable and G is 4-colorable (let $\hat{e}$ be any edge in G, take a 3-coloring of $G - \hat{e}$, and change the color of one of the vertices incident to $\hat{e}$ to the remaining color), so $(G + K_{k-3}) - e = G + (K_{k-3} - e)$ is k-colorable. Finally, if $e = \{v, w\}$ for a vertex v in G and a vertex w in K_{k-3}, let $\hat{e}$ be an edge in G incident to v, take a 3-coloring of $G - \hat{e}$, take a disjoint $(k-3)$-coloring of K_{k-3}, and change the color of v to the color of w. As a result, for all edges e in $G + K_{k-3}$, $(G + K_{k-3}) - e$ is k-colorable. It follows that $G + K_{k-3}$ is in Minimal-k-UnColorability. ◀

The construction above does not prove the analogues of Lemmas 3 and 5: Note that G is 3-colorable if and only if $(G + K_{k-3}) - v$ and $(G + K_{k-3}) - e$ are both $(k-1)$-colorable for every vertex v in K_{k-3} and for every edge e in K_{k-3}, and so we can certainly determine whether a graph is 3-colorable from the optimal solutions to the one-vertex-deleted subgraphs and one-edge-deleted subgraphs of $G + K_{k-3}$ in polynomial time. Turning to criticality and vertex-criticality, we can bound their complexity as follows.

▶ **Theorem 9.** *The two problems of determining whether a graph is critical and whether it is vertex-critical are both in* Θ_2^{p} *and* DP*-hard.*

Proof. For the Θ_2^{p}-membership of the two problems, we observe that the relevant chromatic numbers of a graph $G = (V, E)$ and its neighbors can be computed by querying the NP oracle Colorability $= \{(G, k) \mid \chi(G) \leq k\}$ for every (G, k), $(G - e, k)$, and $(G - v, k)$ for every $e \in E$, $v \in V$, and $k \leq \|V(G)\|$ in parallel.

For the DP-hardness of the two problems, we prove that $h(\Phi) - \{v_{\mathrm{c}}, v_{\mathrm{s}}\}$ is a reduction from Minimal-3-UnSat to both of them. We have already seen that it reduces Minimal-3-UnSat to Minimal-3-UnColorability. Hence, for every $\Phi \in$ Minimal-3-UnSat, the graph $h(\Phi) - \{v_{\mathrm{c}}, v_{\mathrm{s}}\}$ is in Minimal-3-UnColorability $\subseteq$ VertexMinimal-3-UnColorability and thus both critical and vertex-critical. For the converse it suffices to note that, for every $\Phi \in$ CNF with clauses of size at most 3, $h(\Phi) - \{v_{\mathrm{c}}, v_{\mathrm{s}}\}$ is 4-colorable and thus in Minimal-3-UnColorability (in VertexMinimal-3-UnColorability, respectively) if and only if it is critical (vertex-critical, respectively). ◀

The exact complexity of these problems remains open, however. In particular, it is unknown whether they are Θ_2^{p}-hard. This contrasts with the case of Vertex Cover, for which we prove in Theorem 15 that recognizing β-vertex-criticality is indeed Θ_2^{p}-complete.

Before that, however, we return to our model and consider Colorability under the local modification of adding an edge. If we allow only one query, the problem stays NP-hard via a simple Turing reduction: Iteratively adding edges to the given instance eventually leads to

[4] For two graphs G_1 and G_2, the *graph join* $G_1 + G_2$ is the disjoint union $G_1 \cup G_2$ plus a *join edge* added from every vertex of G_1 to every vertex of G_2; see, e.g., Harary's textbook on graph theory [10, p. 21].

a clique as a trivial instance, see Theorem 17 in Appendix B [3]. Note that the restriction to one query is crucial for this reduction to work; without it, the branching may lead to an exponential blowup in the number of instances that need to be considered. The following theorem shows that this breakdown of the hardness proof is inevitable unless P = NP since the problem becomes in fact polynomial-time solvable if just one more oracle call is granted.

▶ **Theorem 10.** *There is a polynomial-time algorithm that computes an optimal coloring for a graph from optimal colorings of all its one-edge-added supergraphs; in fact, two optimal colorings, one for each of two specific one-edge-added supergraphs, suffice.*

For the proof of this theorem, we naturally extend the notion of universal vertices as follows.

▶ **Definition 11.** *An edge $\{u, v\} \in E$ of a graph $G = (V, E)$ is called* universal *if, for every vertex $x \in V - \{u, v\}$, we have $\{x, u\} \in E$ or $\{x, v\} \in E$. A graph is called* universal-edged *if all its edges are universal.*

Additionally, we denote, for any given graph $G = (V, E)$ and any vertex $x \in V$, the *open neighborhood* of x in G by $N(x) \coloneqq \{y \mid \{x, y\} \in E\}$ and the *closed neighborhood* of x in G by $N[x] \coloneqq N(x) \cup \{x\}$. We are now ready to give the proof of Theorem 10.

Proof of Theorem 10. We show that COLORER (Algorithm 1), which uses the oracle of our model and SUBCOL (Algorithm 2) as subroutines, has the desired properties.

Algorithm 1 COLORER.

Input: An undirected graph $G = (V, E)$.
Output: An optimal coloring for G.
Description: Optimizes universal-edged graphs with two queries to ORACLE, which provides optimal solutions to one-edge-added supergraphs; other graphs are optimized via SUBCOL.

```
 1: for every edge $\{u, v\} \in E$ do
 2:     for every vertex $x \in V - \{u, v\}$ do
 3:         if $\{u, x\} \notin E \wedge \{v, x\} \notin E$ then
 4:             $f_1 \leftarrow$ ORACLE($G \cup \{u, x\}$)
 5:             $f_2 \leftarrow$ ORACLE($G \cup \{v, x\}$)
 6:             if $f_1$ uses fewer colors on $G$ than $f_2$ then
 7:                 return $f_1$
 8:             else
 9:                 return $f_2$
10: $k \leftarrow 1$
11: while SUBCOL($G, k$) = NO do
12:     $k \leftarrow k + 1$
13: return SUBCOL($G, k$)
```

We begin by proving that COLORER is correct. Assume first that the input graph $G = (V, E)$ is not universal-edged. Then COLORER can find an edge $\{u, v\} \in E$ with a non-neighboring vertex $x \in V$ and query the oracle on $G \cup \{u, x\}$ and $G \cup \{v, x\}$ for optimal colorings f_1 and f_2. We argue that at least one of them is also optimal for G. Let f be any optimal coloring of G. Since u and v are connected by an edge, we have $f(u) \neq f(v)$ and hence $f(x) \neq f(u)$ or $f(x) \neq f(v)$; see Figure 2in Appendix E [3]. Thus, f is also an optimal coloring of $G \cup \{x, u\}$ or $G \cup \{x, v\}$, and so we have $\chi(G) = \chi(G \cup \{x, u\})$ or $\chi(G) = \chi(G \cup \{x, v\})$. Therefore, f_1 or f_2 is an optimal coloring for G as well and returned on line 7 or 9, respectively.

Algorithm 2 SUBCOL.

Input: An undirected, universal-edged graph $G = (V, E)$ and a positive integer k.
Output: A k-coloring f for G if there is one; NO if there is none.
Description: Works by recursion on k, with $k = 1$ and $k = 2$ serving as the base cases.

```
 1: if $G$ has no edge then
 2:     return the constant 1-coloring with $f(x) = 1$ for all $x \in V$.
 3: else if $k = 1$ then
 4:     return NO.
 5: if $G$ has bipartition $\{A, B\}$ then
 6:     return the 2-coloring $f(x) = \begin{cases} 1 & \text{for } x \in A \text{ and} \\ 2 & \text{for } x \in B. \end{cases}$
 7: else if k=2 then
 8:     return NO.
 9: Choose an arbitrary edge $\{\ell, r\} \in E$.
10: $L \leftarrow N(\ell) - N[r]$;   $R \leftarrow N(r) - N[\ell]$;   $M \leftarrow N(\ell) \cap N(r)$
11: $g \leftarrow \textsc{Subcol}(G[M], k - 2)$
12: if $g =$ NO then
13:     return NO
14: return the $k$-coloring $f(x) = \begin{cases} g(x) & \text{for } x \in M, \\ k - 1 & \text{for } x \in L \cup \{r\}, \text{ and} \\ k & \text{for } x \in R \cup \{\ell\}. \end{cases}$
```

The while loop can be entered only if the graph G is universal-edged. This allows us to compute an optimal solution to G with no queries at all by using SUBCOL (Algorithm 2). We will show that SUBCOL is a polynomial-time algorithm that computes, for any universal-edged graph G and any positive integer k, a k-coloring of G if there is one, and outputs NO otherwise. The while loop of COLORER thus searches the smallest integer k such that G has a k-coloring, that is, $k = \chi(G)$. Hence, an optimal coloring of G is returned on line 13. Due to $k = \chi(G) \leq \|V\|$, COLORER has polynomial time complexity.

It remains to prove the correctness and polynomial time complexity of SUBCOL. This can be done by bounding its recursion depth and verifying the correctness for each of the six return statements; this is hardest for the last two. The proof relies on the properties of the partition $M \cup L \cup R \cup \{\ell\} \cup \{r\}$ as illustrated in Figure 3 in Appendix E [3]; see Appendix E [3] for all details. ◀

In this section, we have proven that MINIMAL-k-UNCOLORABILITY is complete for DP for every $k \geq 3$ and demonstrated that Colorability remains NP-hard in our model for deletion of vertices or edges, whereas it becomes polynomial-time solvable when the local modification is the addition of an edge. In the next section, we turn our attention to Vertex Cover.

4 Vertex Cover

This section will show that the behavior of Vertex Cover in our model is distinctly different from the one that we demonstrated for Colorability in the previous section. In particular, Theorem 12 proves that computing an optimal vertex cover from optimal solutions of one-vertex-deleted subgraphs can be done in polynomial time, which is impossible for optimal colorings according to Theorem 4 unless P = NP.

First, we note that the NP-hardness proof for our most restricted case with only one query still works (i.e., Theorem 17 in Appendix B [3] is applicable): Deleting vertices, adding edges, or deleting edges repeatedly will always lead to the null graph, an edgeless graph, or a clique through polynomially many instances. As we have seen for Colorability in the previous section, hardness proofs of this type may fail due to exponential branching as soon as multiple queries are allowed. We can show that, unless P = NP, this is necessarily the case for edge addition and vertex deletion since two granted queries suffice to obtain a polynomial-time algorithm.

▶ **Theorem 12.** *There is a polynomial-time algorithm that computes an optimal vertex cover for a graph from two optimal vertex covers for some one-vertex-deleted subgraphs.*

Proof. Observe what can happen when a vertex v is removed from a graph G with an optimal vertex cover of size k. If v is part of any optimal vertex cover of G, then the size of an optimal vertex cover for $G-v$ is $k-1$. Given any graph G, pick any two adjacent vertices v_1 and v_2. Since there is an edge between them, one of them is always part of an optimal vertex cover, thus either $G-v_1$ or $G-v_2$ or both will have an optimal vertex cover of size $k-1$. Two queries to the oracle return optimal vertex covers for $G-v_1$ and $G-v_2$. The algorithm chooses the smaller of these two covers (or any, if they are the same size) and adds the corresponding v_i. The resulting vertex cover has size k and is thus optimal for G. ◀

Theorem 19 in Appendix F [3] proves that the analogous result for adding an edge holds as well.

At this point, we would like to prove either an analogue to Theorem 6, showing that computing an optimal vertex cover is NP-hard even if we get access to a solution for every one-edge-deleted subgraph, or an analogue to Theorem 19 in Appendix F [3], showing that the problem is in P if we have access to a solution for more than one one-edge-deleted subgraph. We were unable to prove either, however. The latter is easy to do for many restricted graph classes (e.g., graphs with bridges), yet we suspect that the problem is NP-hard in general. We will detail a few reasons for the apparent difficulty of proving this statement after the following theorem and corollary, which look at deleting a triangle as the local modification.

▶ **Theorem 13.** *There is a reduction g from* 3-SAT *to* VERTEXCOVER *such that, for every* 3-CNF*-formula Φ and for every triangle T in $g(\Phi)$, there is a polynomial-time computable optimal vertex cover of $g(\Phi)-T$.*

The proof of Theorem 13 relies on the standard reduction from 3-SAT to VERTEXCOVER; see [9], where clauses correspond to triangles; see Appendix G [3] for the details. Applying the same argument as in the proof of Theorem 4 yields the following corollary.

▶ **Corollary 14.** *Computing an optimal vertex cover for a graph from optimal vertex covers for the one-triangle-deleted subgraphs is* NP*-hard.*

What can we say about optimal vertex covers for one-edge-deleted graphs? Papadimitriou and Wolfe show [14, Theorem 4] that there is a reduction g from MINIMAL-3-UNSAT to MINIMAL-k-NOVERTEXCOVER (called CRITICAL-VERTEXCOVER in [14]; asking, given a graph G and an integer k, whether G does not have a vertex cover of size k but all one-edge-deleted subgraphs do). The reduction builds in a polynomial-time computable vertex cover of size k for every one-edge-deleted subgraph. And so g is a reduction from 3-SAT to VERTEXCOVER such that there exists a polynomial-time computable function opt such that for every 3-CNF-formula Φ and $g(\Phi)=(G,k)$, it holds, for every edge e in G, that

$\text{opt}(G-e)$ is a vertex cover of size k. Unfortunately, it may happen that an optimal vertex cover of $G-e$ has size $k-1$; namely, if e is an edge connecting two triangles, an edge between two variable-setting vertices, or any edge of the clause triangles. The function opt does thus not give us an optimal vertex cover, thwarting the proof attempt. This shows that we cannot always get results for our model from the constructions for criticality problems.

The following would be one approach to design a polynomial-time algorithm that computes an optimal vertex cover from optimal vertex covers for all one-edge-deleted subgraphs: It is clear that deleting an edge does not increase the size of an optimal vertex cover and decreases it by at most one. If, for any two neighbor graphs, the provided vertex covers differ in size, then we can take the smaller one, restore the deleted edge, and add any one of the two incident vertices to the vertex cover; this gives us the desired optimal vertex cover. If the optimal vertex cover size decreases for all deletions of a single edge, we can do the same with any of them. Thus, it is sufficient to design a polynomial-time algorithm that solves the problem on graphs whose one-edge-deleted subgraphs all have optimal vertex covers of the same size as an optimal vertex cover of the original graph. One might suspect that only very few and simple graphs can be of this kind. However, we obtain infinitely many such graphs by the removal of any edge from different cliques, as already mentioned in the introduction. In fact, there is a far larger class of graphs with this property and no apparent communality to be exploited for the efficient construction of an optimal vertex cover.

We now turn to our complexity results of β-(vertex-)criticality. The reduction from MINIMAL-3-UNSAT to MINIMAL-k-NOVERTEXCOVER by Papadimitriou and Wolfe [14] establishes the DP-hardness of deciding whether a graph is β-critical. However, it seems unlikely that β-criticality is in DP. The obvious upper bound is Θ_2^{p}, since a polynomial number of queries to a VERTEXCOVER oracle, namely (G,k) and $(G-e,k)$ for all edges e in G and all $k \leq \|V(G)\|$, in parallel allows us to determine $\beta(G)$ and $\beta(G-e)$ for all edges e in polynomial time, and thus allows us to determine whether G is β-critical. While we have not succeeded in proving a matching lower bound, or even any lower bound beyond DP-hardness, we do get this lower bound for β-vertex-criticality, thereby obtaining the first Θ_2^{p}-completeness result for a criticality problem.

▶ **Theorem 15.** *Determining whether a graph is β-vertex-critical is Θ_2^{p}-complete.*

Proof. Membership follows with the same argument as above, this time querying the oracle VERTEXCOVER in parallel for all (G,k) and $(G-v,k)$ for all vertices v in G and all $k \leq \|V(G)\|$. To show that this problem is Θ_2^{p}-hard, we use a similar reduction as the one by Hemaspaandra et al. [11, Lemma 4.12] to prove that it is Θ_2^{p}-hard to determine whether a given vertex is a member of a minimum vertex cover. We reduce from the Θ_2^{p}-complete problem $\text{VC}_= = \{(G,H) \mid \beta(G) = \beta(H)\}$ [18]. Let $n = \max(\|V(G)\|, \|V(H)\|)$, let G' consist of $n+1-\|V(G)\|$ isolated vertices, let H' consist of $n+1-\|V(H)\|$ isolated vertices, and let $F = (G \cup G') + (H \cup H')$. Note that $\beta(F) = (n+1) + \min(\beta(G), \beta(H))$. If $\beta(G) = \beta(H)$, then $\beta(F) = (n+1) + \beta(G) = (n+1) + \beta(H)$ and for every vertex v in F, $\beta(F-v) = n + \beta(G)$. Thus, F is critical. If $\beta(G) \neq \beta(H)$, assume without loss of generality that $\beta(G) < \beta(H)$. Then $\beta(F) = n+1+\beta(G)$. Let v be a vertex in G'. Then $\beta(F-v) = \min(n+1+\beta(G), n+\beta(H)) = n+1+\beta(G)$, and therefore F is not critical. ◀

5 Conclusion and Future Research

We defined a natural model that provides new insights into the structural properties of NP-hard problems. Specifically, we revealed interesting differences in the behavior of Colorability and Vertex Cover under different types of local modifications. While Colorability remains

NP-hard when the local modification is the deletion of either a vertex or an edge, there is an algorithm that finds an optimal coloring by querying the oracle on at most two edge-added supergraphs. Vertex Cover, in contrast, becomes easy in our model for both deleting vertices and adding edges, as soon as two queries are granted. The question of what happens for the local modification of deleting an edge remains as an intriguing open problem that defies any simple approach, as briefly outlined above. Moreover, examples of problems where one can prove a jump from membership in P to NP-hardness at a given number of queries greater than 2 might be especially instructive.

With its close connections to many distinct research areas, most notably the study of self-reducibility and critical graphs, our model can serve as a tool for new discoveries. In particular, we were able to exploit the tight relations to criticality in the proof that recognizing β-vertex-critical graphs is Θ_2^{P}-hard, yielding the first completeness result for Θ_2^{P} in the field.

References

1 Claudia Archetti, Luca Bertazzi, and Maria Grazia Speranza. Reoptimizing the Traveling Salesman Problem. *Networks*, 42(3):154–159, 2003.

2 Giorgio Ausiello, Bruno Escoffier, Jérôme Monnot, and Vangelis Paschos. Reoptimization of Minimum and Maximum Traveling Salesman's Tours. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006)*, volume 4059 of *Lecture Notes in Computer Science*, pages 196–207. Springer-Verlag, 2006.

3 Elisabet Burjons, Fabian Frei, Edith Hemaspaandra, Dennis Komm, and David Wehner. Finding Optimal Solutions With Neighborly Help. Technical Report arXiv:1906.10078 [cs.CC], arXiv.org, June 2019. URL: `https://arxiv.org/abs/1906.10078`.

4 Hans-Joachim Böckenhauer, Luca Forlizzi, Juraj Hromkovič, Joachim Kneis, Joachim Kupke, Guido Proietti, and Peter Widmayer. Reusing Optimal TSP Solutions for Locally Modified Input Instances. In *Proceedings of the 4th IFIP International Conference on Theoretical Computer Science (IFIP TCS 2006)*, pages 251–270. Springer-Verlag, 2006.

5 Hans-Joachim Böckenhauer, Juraj Hromkovič, and Dennis Komm. Reoptimization of Hard Optimization Problems. In Teofilo F. Gonzalez, editor, *AAM Handbook of Approximation Algorithms and Metaheuristics*, volume 1, chapter 25, pages 427–454. CRC Press 2018, 2nd edition, 2018.

6 Jin-Yi Cai and Gabriele E. Meyer. Graph Minimal Uncolorability is D^{P}-Complete. *SIAM Journal on Computing*, 16(2):259–277, 1987.

7 Gabriel A. Dirac. Some Theorems on Abstract Graphs. *Proceedings of the London Mathematical Society*, s3-2(1):69–81, 1952.

8 Piotr Faliszewski and Mitsunori Ogihara. On the Autoreducibility of Functions. *Theory of Computing Systems*, 46(2):222–245, 2010.

9 Michael Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

10 Frank Harary. *Graph Theory.* Addison-Wesley, 1991.

11 Edith Hemaspaandra, Holger Spakowski, and Jörg Vogel. The complexity of Kemeny elections. *Theoretical Computer Science*, 349(3):382–391, 2005.

12 Gwenaël Joret. *Entropy and Stability in Graphs.* PhD thesis, Université Libre de Bruxelles, Faculté des Sciences, 2008.

13 Albert R. Meyer and Mike Paterson. With What Frequency Are Apparently Intractable Problems Difficult? Technical Report MIT/LCS/TM-126, Laboratory for Computer Science, MIT, Cambridge, MA, 1979.

14 Christos H. Papadimitriou and David Wolfe. The Complexity of Facets Resolved. *Journal of Computer and System Sciences*, 37(1):2–13, 1988.

15 Claus-Peter Schnorr. Optimal Algorithms for Self-Reducible Problems. In *Proceedings of the 3rd International Colloquium on Automata, Languages, and Programming*, pages 322–337. Edinburgh University Press, 1976.

16 Markus W. Schäffter. Scheduling with forbidden sets. *Discrete Applied Mathematics*, 72(1–2):155–166, 1997.

17 Karl Wagner. Bounded Query Classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.

18 Klaus W. Wagner. More Complicated Questions About Maxima and Minima, and some Closures of NP. *Theoretical Computer Science*, 51(1–2):53–80, 1987.

19 Walter Wessel. Criticity with respect to properties and operations in graph theory. In László Lovász András Hajnal and Vera T. Sós, editors, *Finite and Infinite Sets. (6th Hungarian Combinatorial Colloquium, Eger, 1981)*, volume 2 of *Colloquia Mathematica Societatis Janos Bolyai*, pages 829–837. North-Holland, 1984.

Reconfiguration of Minimum Steiner Trees via Vertex Exchanges

Haruka Mizuta
Graduate School of Information Sciences, Tohoku University, Sendai, Japan
haruka.mizuta.s4@dc.tohoku.ac.jp

Tatsuhiko Hatanaka
Graduate School of Information Sciences, Tohoku University, Sendai, Japan
hatanaka@ecei.tohoku.ac.jp

Takehiro Ito
Graduate School of Information Sciences, Tohoku University, Sendai, Japan
takehiro@ecei.tohoku.ac.jp

Xiao Zhou
Graduate School of Information Sciences, Tohoku University, Sendai, Japan
zhou@ecei.tohoku.ac.jp

Abstract

In this paper, we study the problem of deciding if there is a transformation between two given minimum Steiner trees of an unweighted graph such that each transformation step respects a prescribed reconfiguration rule and results in another minimum Steiner tree of the graph. We consider two reconfiguration rules, both of which exchange a single vertex at a time, and generalize the known reconfiguration problem for shortest paths in an unweighted graph. This generalization implies that our problems under both reconfiguration rules are PSPACE-complete for bipartite graphs. We thus study the problems with respect to graph classes, and give some boundaries between the polynomial-time solvable and PSPACE-complete cases.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Combinatorial reconfiguration, Graph algorithms, Steiner tree

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.79

Funding *Haruka Mizuta*: Partially supported by JSPS KAKENHI Grant Number JP19J10042, Japan.
Takehiro Ito: Partially supported by JST CREST Grant Number JPMJCR1402, and JSPS KAKENHI Grant Numbers JP18H04091 and JP19K11814, Japan.
Xiao Zhou: Partially supported by JSPS KAKENHI Grant Number JP19K11813, Japan.

1 Introduction

Recently, *combinatorial reconfiguration* [7] has been extensively studied in the field of theoretical computer science. In combinatorial reconfiguration, we are given two feasible solutions of a combinatorial search problem, and are asked to determine whether we can transform one into the other by repeatedly applying a specified reconfiguration rule so that all intermediate results are also feasible. Such problems are called *reconfiguration problems* and have been studied intensively for several combinatorial search problems. (See, e.g., surveys [6, 11].) For example, the SHORTEST PATH RECONFIGURATION problem (SPR, for short) is defined as follows [1, 2, 8, 13]: We are given two shortest paths between two vertices s and t in an unweighted graph, and are asked to determine whether or not we can transform one into the other by exchanging a single vertex in a shortest path at a time so that all intermediate results remain shortest paths between s and t. The problem is known to be

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 79; pp. 79:1–79:11
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

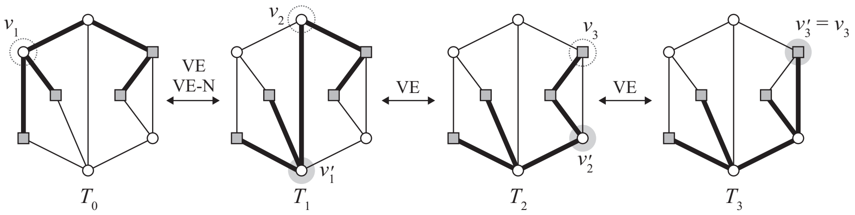


Figure 1 A sequence of minimum Steiner trees, where the terminals are depicted by gray squares, non-terminals by white circles, the edges in Steiner trees by thick lines.

PSPACE-complete [1, 13], and solvable in polynomial time for several graph classes [1, 2, 13]. Interestingly, the polynomial-time solvable cases include planar graphs [2], although many reconfiguration problems remain PSPACE-complete for planar graphs.

In this paper, we introduce and study reconfiguration problems for minimum Steiner trees in a graph, as generalizations of SPR.

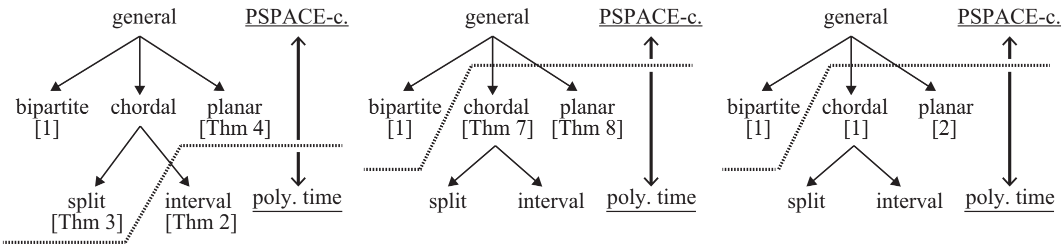
1.1 Our problems

For an unweighted graph G and a vertex subset $S \subseteq V(G)$, called a *terminal set*, a *Steiner tree* of G for S is a subtree of G which contains all vertices in S. A Steiner tree of G for S is *minimum* if it has the minimum number of edges among all Steiner trees of G for S. For example, Figure 1 illustrates four minimum Steiner trees of the same graph G for the same terminal set S. Note that minimum Steiner trees can be seen as a generalization of shortest paths, because any shortest path in G between two vertices s and t forms a minimum Steiner tree of G for $S = \{s, t\}$. We use the terms *node* for Steiner trees and *vertex* for input graphs.

In this paper, we introduce following two reconfiguration rules, which define slightly different adjacency relations on minimum Steiner trees of a graph G for a terminal set S. Both rules exchange a single node v in a minimum Steiner tree T for a single vertex in G (possibly v itself) so that it results in another minimum Steiner tree T' of G for S. (Formal definitions will be given in Section 2.)

- **Vertex Exchange** (VE, for short):
  We say that two minimum Steiner trees T and T' of G for S are *adjacent under* VE if there exist two vertices $v \in V(T)$ and $v' \in V(T')$ such that their removal results in the common subgraph of T and T'. For example, any two consecutive minimum Steiner trees in Figure 1 are adjacent under VE. It should be noted that v and v' can be the same vertex; in such a case, only edges incident to $v = v'$ may be changed between T and T'. (See T_2 and T_3 in Figure 1 as an example.)
- **Vertex Exchange without changing Neighbors** (VE-N, for short):
  We say that two minimum Steiner trees T and T' of G for S are *adjacent under* VE-N if there exist two vertices $v \in V(T)$ and $v' \in V(T')$ such that (a) their removal results in the common subgraph of T and T', and (b) the neighborhood of v in T is equal to that of v' in T'. In Figure 1, only two Steiner trees T_0 and T_1 are adjacent under VE-N. It can hold also under this rule that $v = v'$, but then $T = T'$ holds.

We now define our problems. Given two minimum Steiner trees T_0 and T_r of a graph G for a terminal set S, *the* Minimum Steiner Tree Reconfiguration *problem* (MSTR, for short) *under* VE (resp., VE-N) asks to determine whether or not we can transform one into the other via adjacent minimum Steiner trees under VE (resp., VE-N). For example, when we



(a) MSTR under VE (b) MSTR under VE-N (c) SPR (MSTR with $|S| = 2$)

Figure 2 Our results for MSTR and some known results for SPR, where each arrow between graph classes represents their inclusion relationship: $A \to B$ represents that the graph class B is properly included in the graph class A.

are given two minimum Steiner trees T_0 and T_3 in Figure 1, the answer is yes under VE as illustrated in the figure; while it is a no-instance under VE-N. We note that if $|S| = 2$, then any minimum Steiner tree for S forms a shortest path between the two terminals, and both the rules VE and VE-N are equivalent to the reconfiguration rule of SPR. Therefore, MSTR under both rules VE and VE-N are generalizations of SPR.

1.2 Known and related work

There are several reconfiguration problems for subtrees in an unweighted graph [1, 2, 4, 8, 10, 12, 13]. However, to the best of our knowledge, there is no direct relationship between our problems and these known problems, except for the following two reconfiguration problems.

It is known that SPR is PSPACE-complete even for bipartite graphs [1], and for bounded bandwidth (and hence bounded pathwidth) graphs [13]. Since MSTR under VE and VE-N are generalizations of SPR, they are also PSPACE-complete for bipartite graphs and for bounded bandwidth graphs. As positive results, there are polynomial-time algorithms to solve SPR for planar graphs [2], for chordal graphs [1], and for claw-free graphs [1]. (Figure 2(c) shows a part of these results.)

There is another reconfiguration problem for Steiner trees [10], but it is not a generalization of SPR; its reconfiguration rule is exchanging edges (not vertices). As we have seen in the example of Figure 1, the existence of a transformation often changes according to the choice of reconfiguration rules. However, we will show in Section 3 that some known results for this edge-variant [10] can be converted to our MSTR under VE.

1.3 Our contribution

In this paper, we study the computational complexity of MSTR under VE and VE-N with respect to graph classes. (Figure 2 shows all our results for MSTR.)

We first show that MSTR under VE is solvable in polynomial time for interval graphs, while is PSPACE-complete even for split graphs and for planar graphs. Recall that SPR is solvable in polynomial time for planar graphs and for chordal graphs (and hence split graphs). We next show that MSTR under VE-N is solvable in polynomial time for chordal graphs and for planar graphs; these results generalize the known results for SPR [1, 2].

Notice that there are interesting contrasts between the reconfiguration rules VE and VE-N when we focus on planar graphs, chordal graphs, and split graphs: MSTR is PSPACE-complete under VE, while is solvable in polynomial time under VE-N.

We omit proofs for the claims marked with (∗) from this extended abstract.

2 Preliminaries

In this paper, we consider only simple and unweighted graphs. For a graph G, we denote by $V(G)$ and $E(G)$ the vertex set and the edge set of G, respectively. For a graph G and its vertex subset $S \subseteq V(G)$, we denote by $G[S]$ the subgraph of G induced by S. For a vertex $v \in V(G)$, let $N_G(v)$ be the set of all neighbors of v in G, that is, $N_G(v) = \{w \in V(G) \mid vw \in E(G)\}$.

We now formally define our reconfiguration rules. Let T and T' be two minimum Steiner trees of a graph G for a terminal set S. Then, T and T' are *adjacent under* VE if there exist two vertices $v \in V(T)$ and $v' \in V(T')$ such that

- $T[V(T) \setminus \{v\}] = T'[V(T') \setminus \{v'\}]$.

Recall that v and v' can be the same vertex; in such a case, only edges incident to $v = v'$ may be changed between T and T'. On the other hand, T and T' are *adjacent under* VE-N if there exist two vertices $v \in V(T)$ and $v' \in V(T')$ such that

- $T[V(T) \setminus \{v\}] = T'[V(T') \setminus \{v'\}]$; and
- $N_T(v) = N_{T'}(v')$.

Note that the first condition for VE-N is the same as the condition for VE. It can hold also under VE-N that $v = v'$, but then $T = T'$ holds.

For two minimum Steiner trees T and T', a *reconfiguration sequence* between T and T' under VE (resp., VE-N) is a sequence $\langle T = T_0, T_1, \ldots, T_\ell = T' \rangle$ of minimum Steiner trees such that T_i and T_{i+1} are adjacent under VE (resp., VE-N) for each $i \in \{0, 1, \ldots, \ell - 1\}$. We write $T \overset{\mathsf{VE}}{\leftrightsquigarrow} T'$ (resp., $T \overset{\mathsf{VE\text{-}N}}{\leftrightsquigarrow} T'$) if there exists a reconfiguration sequence between T and T' under VE (resp., VE-N). Then, we formally define the Minimum Steiner Tree Reconfiguration problem (MSTR, for short) under VE (resp., VE-N) as follows:

MSTR under VE (resp., VE-N)
Input: A graph G, a terminal set $S \subseteq V(G)$, and two minimum Steiner trees T_0 and T_r of G for S.
Task: Determine whether $T_0 \overset{\mathsf{VE}}{\leftrightsquigarrow} T_\mathrm{r}$ (resp., $T_0 \overset{\mathsf{VE\text{-}N}}{\leftrightsquigarrow} T_\mathrm{r}$) or not.

We denote by a 4-tuple $(G, S, T_0, T_\mathrm{r})$ an instance of the problems. Throughout the paper, we assume without loss of generality that $|V(T_0)| = |V(T_\mathrm{r})|$ holds; otherwise it is clearly a no-instance.

3 Minimum Steiner Tree Reconfiguration under VE

In this section, we show that MSTR under VE is solvable in polynomial time for interval graphs, while it is PSPACE-complete for split graphs and for planar graphs. To this end, we use the concept of "Steiner sets" and their reconfiguration, which was introduced by [10].

3.1 Steiner sets and their reconfiguration

For a graph G and a terminal set S, a *Steiner set* of G for S is a vertex subset $F \subseteq V(G)$ such that $S \subseteq F$ and $G[F]$ is connected. Notice that if a subtree T of G is a Steiner tree for S, then $V(T)$ is a Steiner set of G for S. Conversely, if F is a Steiner set of G for S, then any spanning tree of $G[F]$ is a Steiner tree for S. A Steiner set F of G for S is *minimum* if the cardinality of F is minimum among all Steiner sets of G for S.

For two Steiner sets F and F' of G for S, a sequence $\langle F = F_0, F_1, \ldots, F_\ell = F' \rangle$ of Steiner sets of G for S is called a *Steiner set sequence* between F and F' if $|F_i \setminus F_{i+1}| = |F_{i+1} \setminus F_i| = 1$ holds for each $i \in \{0, 1, \ldots, \ell - 1\}$. Note that all Steiner sets in the sequence have the same cardinality. The following lemma shows that, in some sense, we do not need to care the tree structure property (but need to care only a connectivity) when we want to check the existence of a reconfiguration sequence under VE.

▶ **Lemma 1** (∗). *Let T and T' be Steiner trees of a graph G for a terminal set S. Then, $T \overset{\mathsf{VE}}{\leftrightsquigarrow} T'$ holds if and only if there exists a Steiner set sequence between two Steiner sets $V(T)$ and $V(T')$ of G for S.*

The concept of Steiner sets was introduced for the reconfiguration of Steiner trees via edge exchanges [10]. Lemma 1 allows us to convert two known results for this edge-exchange variant [10] to our MSTR under VE.

We first consider interval graphs. A graph G with $V(G) = \{v_1, v_2, \dots, v_n\}$ is an *interval graph* if there exists a set $\mathcal{I}$ of (closed) intervals $I_1, I_2, \dots, I_n$ such that $v_i v_j \in E(G)$ if and only if $I_i \cap I_j \neq \emptyset$ for each $i, j \in \{1, 2, \dots, n\}$. For a given graph G, it can be determined in linear time whether G is an interval graph or not [9].

▶ **Theorem 2.** *Let $(G, S, T_0, T_\mathrm{r})$ be a given instance of* MSTR *under* VE *such that G is an interval graph. Then, $T_0 \overset{\mathsf{VE}}{\leftrightsquigarrow} T_\mathrm{r}$ holds.*

Proof. It is known that if a given graph is an interval graph, then there always exists a Steiner set sequence between any pair of Steiner sets of the same cardinality [10]. Thus, the theorem follows from Lemma 1. ◀

We then consider split graphs. A graph is a *split graph* if its vertex set can be partitioned into a clique and an independent set. The following theorem can be obtained by a polynomial-time reduction which is similar to that for Theorem 3 of [10].

▶ **Theorem 3** (∗). MSTR *under* VE *is PSPACE-complete for split graphs.*

3.2 PSPACE-completeness for planar graphs

In this subsection, we consider planar graphs, and give the following theorem.

▶ **Theorem 4.** MSTR *under* VE *is PSPACE-complete for planar graphs.*

We note that MSTR under VE is in PSPACE. Therefore, in the remainder of this subsection, we will prove that the problem is PSPACE-hard for planar graphs. To this end, we construct a polynomial-time reduction from the Minimum Vertex Cover Reconfiguration problem.

Recall that a *vertex cover* C of a graph G is a vertex subset of G which contains at least one of the two endpoints of every edge in G. A vertex cover C of G is *minimum* if the cardinality of C is minimum among all vertex covers of G. Given a graph G and two minimum vertex covers C_0 and C_r of G, the Minimum Vertex Cover Reconfiguration problem (MVCR, for short) asks to determine whether or not there exists a sequence $\langle C_0, C_1, \dots, C_\ell = C_\mathrm{r} \rangle$ of minimum vertex covers of G such that $|C_i \setminus C_{i+1}| = |C_{i+1} \setminus C_i| = 1$ holds for any $i \in \{0, 1, \dots, \ell - 1\}$; we call such a sequence a *vertex cover sequence* between C_0 and C_r. We denote by a triple (G, C_0, C_r) an instance of MVCR. This problem is known to be PSPACE-complete for planar graphs [5].[1]

[1] Precisely, Hearn and Demaine [5] showed the PSPACE-completeness for the reconfiguration of maximum independent sets. However, it immediately yields the PSPACE-completeness of MVCR.

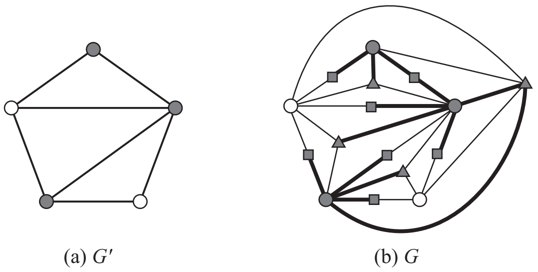


Figure 3 (a) An input graph G' of MVCR with a minimum vertex cover C', where the vertices in C' are depicted by gray vertices, and (b) its corresponding planar graph G of MSTR together with the minimum Steiner tree corresponding to C', where face-terminals are depicted by triangles, edge-terminals by squares, and the minimum Steiner tree by thick lines.

Reduction

Let (G', C'_0, C'_r) be a given instance of MVCR such that G' is a planar graph. We fix a planar embedding of G' arbitrarily, and denote by $F(G')$ the set of all faces (including the outer face) of G'. We construct the corresponding instance (G, S, T_0, T_r) of MSTR under VE, as follows. (See Figure 3 as an example.)

We first construct the corresponding graph G from G'. For each face $f \in F(G')$, we add a new vertex w_f, and join w_f and all vertices on the boundary of f by adding new edges. Then, we subdivide each original edge $e = uv \in E(G')$ by adding a new vertex w_e. Let G be the resulting graph. Then, G is a planar graph.

We then define the corresponding terminal set S as the set of all newly added vertices, that is, $S = \{w_f \mid f \in F(G')\} \cup \{w_e \mid e \in E(G')\}$; each w_f is called a *face-terminal*, while each w_e is called an *edge-terminal*.

We finally define the corresponding minimum Steiner trees T_0 and T_r. We will prove later in Lemma 5 that both $C'_0 \cup S$ and $C'_r \cup S$ form minimum Steiner sets of G for S. Then, we choose arbitrary spanning trees of $G[C'_0 \cup S]$ and $G[C'_r \cup S]$ as T_0 and T_r, respectively.

This completes the construction of (G, S, T_0, T_r). The construction can be done in polynomial time.

Correctness

We start with showing that both $C'_0 \cup S$ and $C'_r \cup S$ form minimum Steiner sets of G for S, and hence T_0 and T_r are indeed minimum Steiner trees of G for S.

▶ **Lemma 5** ($*$). *Let C' be a vertex subset of $V(G')$. Then, C' is a minimum vertex cover of G' if and only if $C' \cup S$ is a minimum Steiner set of G for S.*

The following lemma completes the proof of Theorem 4.

▶ **Lemma 6.** *(G', C'_0, C'_r) is a* yes*-instance of* MVCR *if and only if (G, S, T_0, T_r) is a* yes*-instance of* MSTR.

Proof. By Lemma 1, it suffices to show that there exists a vertex cover sequence on G' between C'_0 and C'_r if and only if there exists a Steiner set sequence on G between $V(T_0)$ and $V(T_r)$.

First, suppose that there exists a Steiner set sequence $\langle V(T_0) = F_0, F_1, \ldots, F_\ell = V(T_r)\rangle$ between $V(T_0)$ and $V(T_r)$. Then, Lemma 5 implies that the sequence $\langle C'_0 = F_0 \setminus S, F_1 \setminus S, \ldots, F_\ell \setminus S = C'_r\rangle$ is a vertex cover sequence on G' between C'_0 and C'_r.

Second, suppose that there exists a vertex cover sequence $\langle C'_0, C'_1, \ldots, C'_{\ell'} = C'_{\mathrm{r}} \rangle$ between C'_0 and C'_{r}. Then, Lemma 5 implies that the sequence $\langle V(T_0) = C'_0 \cup S, C'_1 \cup S, \ldots, C'_{\ell'} \cup S = V(T_{\mathrm{r}}) \rangle$ is a Steiner set sequence on G between $V(T_0)$ and $V(T_{\mathrm{r}})$. ◀

4 Minimum Steiner Tree Reconfiguration under VE-N

In this section, we show that MSTR under VE-N is solvable in polynomial time for chordal graphs and for planar graphs.

We first consider chordal graphs. A graph is *chordal* if G has no induced cycle of length at least four [3]. We use a well-known characterization of chordal graphs, called perfect elimination orderings [3], and give the following theorem.

▶ **Theorem 7** (∗). MSTR *under* VE-N *is solvable in polynomial time for chordal graphs.*

We then consider planar graphs. Recall that MSTR under VE is PSPACE-complete for planar graphs (as shown in Theorem 4). In contrast, we give the following theorem.

▶ **Theorem 8.** MSTR *under* VE-N *is solvable in polynomial time for planar graphs.*

As a proof of the Theorem 8, we construct a polynomial-time algorithm to solve MSTR under VE-N for planar graphs. Roughly speaking, our idea is to decompose a given instance of MSTR under VE-N into several SPR instances for planar graphs. Then, we can solve each SPR instance by using the polynomial-time algorithm for SPR on planar graphs [2]. Finally, we combine the answers to SPR instances, and output the answer to the original MSTR instance under VE-N. To this ends, we introduce the concept of Steiner tree embeddings and their reconfiguration, which gives a necessary condition for the existence of a reconfiguration sequence under VE-N.

4.1 Steiner tree embeddings and their reconfiguration

We first introduce the concept of Steiner tree embeddings. Let T be a Steiner tree of a graph G for a terminal set S. An injection $\varphi \colon V(T) \to V(G)$ is called a T*-embedding* into G if the following two conditions hold:

- $\varphi(x)\varphi(y) \in E(G)$ if $xy \in E(T)$; and
- $\varphi(s) = s$ holds for each $s \in S \subseteq V(T)$.

Thus, a T-embedding φ *defines* a Steiner tree T_φ of G for S. Observe that no two distinct T-embeddings define the same Steiner tree. A Steiner tree T' is said to be T*-embeddable* if there exists a T-embedding φ which defines T'. Note that T itself is T-embeddable. We now give the following lemma.

▶ **Lemma 9** (∗). *Let T_a and T_b be any two minimum Steiner trees of a graph G for a terminal set S. If $T_a \overset{\textsf{VE-N}}{\rightsquigarrow} T_b$, then T_b is T_a-embeddable.*

By taking a contrapositive of Lemma 9, we can conclude that a given instance $(G, S, T_0, T_{\mathrm{r}})$ is a no-instance if T_{r} is not T_0-embeddable; this can be checked in polynomial time. Thus, in the remainder of this section, we assume without loss of generality that T_{r} is T_0-embeddable.

We then introduce the reconfiguration of Steiner tree embeddings. Let T be a Steiner tree of a graph G for a terminal set S. We say that two T-embeddings φ and φ' are *adjacent* if exactly one node in T is mapped into different vertices between φ and φ', that is, $|\{x \in V(T) \mid \varphi(x) \neq \varphi'(x)\}| = 1$ holds. For two T-embeddings φ and φ', an *embedding sequence* between φ and φ' is a sequence $\langle \varphi = \varphi_0, \varphi_1, \ldots, \varphi_\ell = \varphi' \rangle$ of T-embeddings such that φ_i and φ_{i+1} are adjacent for each $i \in \{0, 1, \ldots, \ell - 1\}$. We write $\varphi \overset{\textsf{emb}}{\rightsquigarrow} \varphi'$ if there exists an embedding sequence between φ and φ'. Then, we have the following lemma.

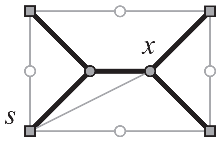


Figure 4 The subtree T depicted by thick lines is a minimum Steiner tree. However, the subpath on T between s and x is not a shortest path in the underlying graph.

▶ **Lemma 10** (∗). *Suppose that T_a and T_b are any two minimum Steiner trees of a graph G for a terminal set S such that T_b is T_a-embeddable. Let φ_a and φ_b be T_a-embeddings which define T_a and T_b, respectively. Then, $T_a \overset{\text{VE-N}}{\leftrightsquigarrow} T_b$ if and only if $\varphi_a \overset{\text{emb}}{\leftrightsquigarrow} \varphi_b$.*

By Lemmas 9 and 10, MSTR under VE-N can be rephrased to the following problem: Given a graph G, a terminal set S, a minimum Steiner tree T (actually T_0), and two T-embeddings φ_0 and φ_{r} into G, we are asked to determine whether or not there exists an embedding sequence between φ_0 and φ_{r}. Therefore, we also denote by $(G, S, T, \varphi_0, \varphi_{\mathrm{r}})$ an instance of MSTR under VE-N.

4.2 Layers for Steiner trees

We here introduce one more important concept, called layers, which was originally introduced by Bonsma [1] for SPR. We generalize the concept to Steiner trees.

Let T be a minimum Steiner tree of a graph G for a terminal set S. For each $x \in V(T)$, let $L_T(x) = \{\varphi(x) \in V(G) \mid \varphi \text{ is a } T\text{-embedding into } G\}$; we call $L_T(x)$ the *layer* of x. Notice that $L_T(s) = \{s\}$ holds for each $s \in S$. We write $L_T(V') = \bigcup_{x \in V'} L_T(x)$ for any node subset $V' \subseteq V(T)$. Then, we have the following property, which says that the layers are disjoint.

▶ **Lemma 11** (∗). *Let T be any minimum Steiner tree of a graph G for a terminal set S. Then, $L_T(x) \cap L_T(y) = \emptyset$ holds for any two distinct nodes $x, y \in V(T)$.*

We call a node $x \in V(T)$ a *branching node* of T if $|N_T(x)| \geq 3$. Let $B(T)$ be the set of all branching nodes of T. Then, we show that a layer of each node in $B(T)$ contains at most two vertices if a given graph is planar.

▶ **Lemma 12** (∗). *Let T be any minimum Steiner tree of a graph G for a terminal set S. If G is planar, then $|L_T(x)| \leq 2$ holds for every branching node $x \in B(T)$.*

We now explain how to compute the layers for a Steiner tree. In SPR [1], we can easily find the layers for a shortest path by computing the distances from the two terminals to each vertex in the underlying graph. This is because the subpath between each node and each terminal is always a shortest path in the underlying graph. On the other hand, this property does not always hold if $|S| \geq 3$, and hence it is difficult to find the layers simply by computing the distances. (For example, see Figure 4.)

Our idea is to compute the "refined" layers for a Steiner tree, instead of computing the layers completely. Let $(G, S, T, \varphi_0, \varphi_{\mathrm{r}})$ be a given instance of MSTR under VE-N. Then, for all nodes $x \in V(T)$, it suffices to find vertex subsets $L'_T(x)$ such that

(a) $L'_T(x) \subseteq L_T(x)$; and

(b) $\varphi(x) \in L'_T(x)$ holds for any T-embedding φ satisfying $\varphi_0 \overset{\text{emb}}{\leftrightsquigarrow} \varphi$ or $\varphi_{\mathrm{r}} \overset{\text{emb}}{\leftrightsquigarrow} \varphi$.

To avoid a confusion, we call such a vertex subset $L'_T(x)$ the *refined-layer* of x, while call the (original) layer $L_T(x)$ the *complete-layer* of x. We know that the vertices in $L_T(x) \setminus L'_T(x)$

are useless when we want to check if $\varphi_0 \overset{\mathsf{emb}}{\leftrightsquigarrow} \varphi_{\mathrm{r}}$ or not. The following lemma says that the refined-layers can be found in polynomial time.

▶ **Lemma 13** (∗). *Let* $(G, S, T, \varphi_0, \varphi_{\mathrm{r}})$ *be a given instance of* MSTR *under* VE-N *such that* G *is a planar graph. Then, there exists a polynomial-time algorithm to compute the refined-layers for all nodes in* T*.*

Given an instance $(G, S, T, \varphi_0, \varphi_{\mathrm{r}})$ of MSTR under VE-N, we can compute refined-layers in polynomial time by Lemma 13. Since the vertices in $L_T(x) \setminus L'_T(x)$, $x \in V(T)$, are useless, we can remove such useless vertices from G. In this way, we can assume without loss of generality that each vertex in G belongs to exactly one (complete-)layer, and we indeed know the layer $L_T(x)$ for each node $x \in V(T)$.

4.3 Decomposition of an MSTR instance into SPR instances

Suppose that $(G, S, T, \varphi_0, \varphi_{\mathrm{r}})$ is an instance of MSTR under VE-N, and that we have the layer $L_T(x)$ for each node $x \in V(T)$. We say that two nodes $x, y \in B(T) \cup S$ are *close* if the unique path on T between x and y contains no vertex in $(B(T) \cup S) \setminus \{x, y\}$. To avoid the duplication of $\{x, y\}$ and $\{y, x\}$, we choose one of the ordered pairs (x, y) and (y, x) arbitrarily for each pair of close nodes, and define the set $C(T)$ of all ordered pairs (x, y) of close nodes x, y in $B(T) \cup S$; we call each pair in $C(T)$ a *close pair.*

For each close pair (x, y) in $C(T)$, we now construct the corresponding instance SPR(x, y) $= (G', S', T', \varphi'_0, \varphi'_{\mathrm{r}})$ such that $|S'| = 2$, as follows. Let P be the unique path on T between x and y. Note that by the definition of close pairs, P is a shortest path on G between x and y. Consider the subgraph of G induced by the vertices in $L_T(V(P))$. We add two new vertices s_x and t_y to the subgraph so that s_x is joined to all vertices in $L_T(x)$ and t_y is joined to all vertices in $L_T(y)$; note that each of s_x and t_y is indeed adjacent to one or two vertices. Let G' be the resulting graph, and let $S' = \{s_x, t_y\}$. We then define T' as the path on G' between s_x and t_y obtained by adding s_x and t_y to P. Note that T' is a shortest path on G' between s_x and t_y. We finally define φ'_0 as a T'-embedding into G' such that $\varphi'_0(s_x) = s_x$, $\varphi'_0(t_y) = t_y$, and $\varphi'_0(x) = \varphi_0(x)$ for each $x \in V(P)$. Similarly, we define φ'_{r} as a T'-embedding into G' such that $\varphi'_{\mathrm{r}}(s_x) = s_x$, $\varphi'_{\mathrm{r}}(t_y) = t_y$, and $\varphi'_{\mathrm{r}}(x) = \varphi_{\mathrm{r}}(x)$ for each $x \in V(P)$. This completes the construction of SPR(x, y). The corresponding instance SPR(x, y) can be obtained in polynomial time, and satisfies the following property.

▶ **Lemma 14** (∗). *If* G *is planar, then* G' *is also planar.*

By Lemma 14 we can solve the instance SPR(x, y) for each close pair $(x, y) \in C(T)$ by the polynomial-time algorithm for SPR on planar graphs [2]. We can immediately conclude that the given instance $(G, S, T, \varphi_0, \varphi_{\mathrm{r}})$ of MSTR under VE-N is a no-instance if there exists at least one instance SPR(x, y) whose answer is no. However, even if the answers are yes to all instances SPR(x, y), $(x, y) \in C(T)$, it is not always possible to extend their embedding sequences to a whole embedding sequence between φ_0 and φ_{r} for the original instance $(G, S, T, \varphi_0, \varphi_{\mathrm{r}})$. To check this, we introduce further notion.

Consider an embedding sequence $\mathcal{R} = \langle \varphi = \varphi_0, \varphi_1, \ldots, \varphi_\ell = \varphi' \rangle$ between two T-embeddings φ and φ'. For each node $x \in V(T)$, we say that $\mathcal{R}$ is *x-touching* if the assignment of x is changed by $\mathcal{R}$ at least once; otherwise it is *x-untouching.* Note that if $\varphi(x) \neq \varphi'(x)$ for a node $x \in V(T)$, then any embedding sequence between φ and φ' must be x-touching. On the other hand, if $|L_T(x)| = 1$, then any embedding sequence must be x-untouching. For each close pair $(x, y) \in C(T)$ and its corresponding instance SPR$(x, y) =$ $(G', S', T', \varphi'_0, \varphi'_{\mathrm{r}})$, we define the set $\mathsf{Touch}(x, y) \subseteq \{(\mathsf{u}, \mathsf{u}), (\mathsf{u}, \mathsf{t}), (\mathsf{t}, \mathsf{u}), (\mathsf{t}, \mathsf{t})\}$, as follows:

- $(\mathsf{u},\mathsf{u}) \in \mathsf{Touch}(x,y)$ if and only if there exists an embedding sequence between φ'_0 and φ'_{r} which is x-untouching and y-untouching;
- $(\mathsf{u},\mathsf{t}) \in \mathsf{Touch}(x,y)$ if and only if there exists an embedding sequence between φ'_0 and φ'_{r} which is x-untouching and y-touching;
- $(\mathsf{t},\mathsf{u}) \in \mathsf{Touch}(x,y)$ if and only if there exists an embedding sequence between φ'_0 and φ'_{r} which is x-touching and y-untouching; and
- $(\mathsf{t},\mathsf{t}) \in \mathsf{Touch}(x,y)$ if and only if there exists an embedding sequence between φ'_0 and φ'_{r} which is x-touching and y-touching.

Note that $\mathsf{Touch}(x,y) = \emptyset$ if there is no embedding sequence between φ'_0 and φ'_{r}. Then, we have the following lemma.

▶ **Lemma 15** (∗). *For each close pair $(x,y) \in C(T)$, $\mathsf{Touch}(x,y)$ can be computed in polynomial time.*

We finally solve the given instance $(G,S,T,\varphi_0,\varphi_{\mathrm{r}})$ of MSTR under VE-N. Assume that $\mathrm{SPR}(x,y)$ are yes-instances for all close pairs $(x,y) \in C(T)$, and hence $\mathsf{Touch}(x,y) \neq \emptyset$; otherwise $(G,S,T,\varphi_0,\varphi_{\mathrm{r}})$ is a no-instance. Consider an assignment $\alpha : B(T) \cup S \to \{\mathsf{u},\mathsf{t}\}$. Then, we say that α is *synchronizing* if $(\alpha(x),\alpha(y)) \in \mathsf{Touch}(x,y)$ holds for every close pair $(x,y) \in C(T)$. The following lemma completes the proof of Theorem 8.

▶ **Lemma 16** (∗). *Suppose that $(G,S,T,\varphi_0,\varphi_{\mathrm{r}})$ is an instance of* MSTR *under* VE-N *such that G is a planar graph. Then, it is a* yes*-instance if and only if there exists a synchronizing assignment α. Furthermore, the existence of a synchronizing assignment can be checked in polynomial time.*

5 Conclusion

In this paper, we have introduced the Minimum Steiner Tree Reconfiguration (MSTR) problems under two reconfiguration rules VE and VE-N. As summarized in Figure 2, we have studied the polynomial-time solvability of the problems with respect to graph classes, and shown several interesting contrasts. In particular, when we focus on planar graphs, chordal graphs, and split graphs, MSTR is PSPACE-complete under VE, while is solvable in polynomial time under VE-N. It would give us a deeper understanding of the problems if there is a graph class such that MSTR is solvable in polynomial time under VE but is intractable under VE-N.

References

1 P. Bonsma. The complexity of rerouting shortest paths. *Theoretical Computer Science*, 510:1–12, 2013.

2 P. Bonsma. Rerouting shortest paths in planar graphs. *Discrete Applied Mathematics*, 231:95–112, 2017.

3 A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey*. SIAM, Philadelphia, PA, 1999.

4 T. Hanaka, T. Ito, H. Mizuta, B. Moore, N. Nishimura, V. Subramanya, A. Suzuki, and K. Vaidyanathan. Reconfiguring spanning and induced subgraphs. In *Proceedings of the 24th International Computing and Combinatorics Conference (COCOON 2018)*, volume 10976 of *Lecture Notes in Computer Science*, pages 428–440, 2018.

5 R.A. Hearn and E.D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1–2):72–96, 2005.

6 J. van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013.

7 T. Ito, E.D. Demaine, N.J.A. Harvey, C.H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011.

8 M. Kamiński, P. Medvedev, and M. Milanič. Shortest paths between shortest paths. *Theoretical Computer Science*, 412(39):5205–5210, 2011.

9 N. Korte and R.H. Möhring. An Incremental Linear-time Algorithm for Recognizing Interval Graphs. *SIAM Journal on Computing*, 18(1):68–81, 1989.

10 H. Mizuta, T. Ito, and X. Zhou. Reconfiguration of Steiner trees in an unweighted graph. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E100-A(7):1532–1540, 2017.

11 N. Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

12 K. Wasa, K. Yamanaka, and H. Arimura. The Complexity of Induced Tree Reconfiguration Problems. *IEICE Transactions on Information and Systems*, 102-D(3):464–469, 2019.

13 M. Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *Journal of Computer and System Sciences*, 93:1–10, 2018.

The Perfect Matching Reconfiguration Problem

Marthe Bonamy
CNRS, LaBRI, Université de Bordeaux,
Talence, France
marthe.bonamy@u-bordeaux.fr

Nicolas Bousquet
CNRS, Laboratoire G-SCOP, Grenoble-INP,
Univ. Grenoble-Alpes, Grenoble, France
nicolas.bousquet@grenoble-inp.fr

Marc Heinrich
LIRIS, Université Claude Bernard Lyon 1,
Lyon, France
marc.heinrich@univ-lyon1.fr

Takehiro Ito
Graduate School of Information Sciences,
Tohoku University, Sendai, Japan
takehiro@ecei.tohoku.ac.jp

Yusuke Kobayashi
Research Institute for Mathematical Sciences,
Kyoto University, Kyoto, Japan
yusuke@kurims.kyoto-u.ac.jp

Arnaud Mary
LBBE, Université Claude Bernard Lyon 1,
Lyon, France
arnaud.mary@univ-lyon1.fr

Moritz Mühlenthaler
Fakultät für Mathematik,
TU Dortmund University, Dortmund, Germany
moritz.muehlenthaler@math.tu-dortmund.de

Kunihiro Wasa
Principles of Informatics Research Division,
National Institute of Informatics, Tokyo, Japan
wasa@nii.ac.jp

Abstract

We study the PERFECT MATCHING RECONFIGURATION problem: Given two perfect matchings of a graph, is there a sequence of flip operations that transforms one into the other? Here, a flip operation exchanges the edges in an alternating cycle of length four. We are interested in the complexity of this decision problem from the viewpoint of graph classes. We first prove that the problem is PSPACE-complete even for split graphs and for bipartite graphs of bounded bandwidth with maximum degree five. We then investigate polynomial-time solvable cases. Specifically, we prove that the problem is solvable in polynomial time for strongly orderable graphs (that include interval graphs and strongly chordal graphs), for outerplanar graphs, and for cographs (also known as P_4-free graphs). Furthermore, for each yes-instance from these graph classes, we show that a linear number of flip operations is sufficient and we can exhibit a corresponding sequence of flip operations in polynomial time.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Combinatorial Reconfiguration, Graph Algorithms, Perfect Matching

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.80

Funding *Marthe Bonamy*: Partially supported by ANR project GrR (ANR-18-CE40-0032).
Nicolas Bousquet: Partially supported by ANR project GrR (ANR-18-CE40-0032).
Marc Heinrich: Partially supported by ANR project GrR (ANR-18-CE40-0032).
Takehiro Ito: Partially supported by JST CREST Grant Number JPMJCR1402, and JSPS KAKENHI Grant Numbers JP18H04091 and JP19K11814, Japan.
Yusuke Kobayashi: Supported by JST ACT-I Grant Number JPMJPR17UB, and JSPS KAKENHI Grant Numbers JP16K16010, JP16H03118, JP17K19960 and JP18H05291, Japan.
Arnaud Mary: Partially supported by ANR project GrR (ANR-18-CE40-0032).
Kunihiro Wasa: Partially supported by JST CREST Grant Number JPMJCR1401 and JPMJCR18K3, and JSPS KAKENHI Grant Number JP19K20350, Japan.

Acknowledgements This work is partially supported by JSPS and MAEDI under the Japan-France Integrated Action Program (SAKURA).

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 80; pp. 80:1–80:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Given an instance of some combinatorial search problem and two of its feasible solutions, a *reconfiguration problem* asks whether one solution can be transformed into the other in a step-by-step fashion, such that each intermediate solution is also feasible. Reconfiguration problems capture dynamic situations, where some solution is in place and we would like to move to a desired alternative solution without becoming infeasible. A systematic study of the complexity of reconfiguration problems was initiated in [21]. Recently the topic has gained a lot of attention in the context of constraint satisfaction problems and graph problems, such as the independent set problem, the matching problem, and the dominating set problem. Reconfiguration problems naturally arise for operational research problems but also are closely related to uniform sampling (using Markov chains) or enumeration of solutions of a problem. For an overview of recent results on reconfiguration problems, the reader is referred to the surveys of van den Heuvel [17] and Nishimura [27].

In order to define valid step-by-step transformations, an adjacency relation on the set of feasible solutions is needed. Depending on the problem, there may be different natural choices of adjacency relations. For instance, we may assume that two matchings of a graph are adjacent if one can be obtained from the other by exchanging precisely one edge, i.e., there exist $e \in M$ and $f \in M'$ such that $M \setminus \{e\} = M' \setminus \{f\}$. The corresponding modification of a matching is usually referred to as *token jumping* (TJ). Here, the *tokens* are placed on the edges of a matching and a token may be "moved" from an edge of the matching to another edge so that we obtain another matching. There is another similar adjacency relation, where two matchings are adjacent if one can be obtained from the other by moving a token to some incident edge. This adjacency relation is called *token sliding* (TS). Ito et al. [21] gave a polynomial-time algorithm that decides if there is a transformation between two given matchings under the TJ and TS operations.

1.1 The perfect matching reconfiguration problem

Recall that a matching of a graph is *perfect* if it covers each vertex. We study the complexity of deciding if there is a step-by-step transformation between two given perfect matchings of a graph. However, according to the adjacency relations given by the TS and TJ operations, there is no transformation between any two distinct perfect matchings of a graph. Since the symmetric difference of any two perfect matchings of a graph consists of even-length disjoint cycles, it is natural to consider a different adjacency relation for perfect matchings. We say that two perfect matchings of a graph differ by a *flip* (or *swap*) if their symmetric difference induces a cycle of length four. We consider two perfect matchings to be *adjacent* if they differ by a flip. Intuitively, for two adjacent perfect matchings M and M', we think of a flip as an operation that exchanges edges in $M \setminus M'$ for edges in $M' \setminus M$.

An example of a transformation between two perfect matchings of a graph is given in Figure 1. We formalize the task of deciding the existence of a transformation between two given perfect matchings as follows.

Perfect Matching Reconfiguration
Input: Graph G, perfect matchings M_s and M_t of G.
Question: Is there a sequence of flips that transforms M_s into M_t?

We take the flip operation on a cycle of length four as the adjacency relation in this paper, because a flip is in some sense a minimal modification of a perfect matching. Note that if we do not restrict the length of a cycle in the definition of a flip, then for any two perfect

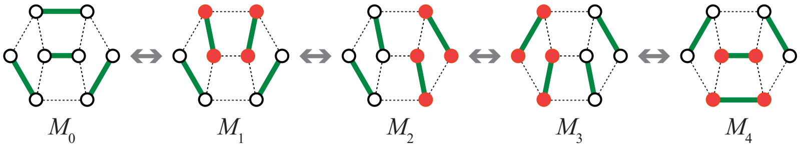


Figure 1 A transformation between perfect matchings M_0 and M_4 under the flip operation. For each i, $1 \leq i \leq 4$, the matching M_i can be obtained from M_{i-1} by applying the flip operation to the cycle induced by the four painted (red) vertices in M_i.

matchings M_s and M_t of a graph, there always exists a sequence of flip operations that transforms M_s into M_t, since we can perform a flip on each (disjoint) cycle in the symmetric difference of M_s and M_t. As a compromise, we may extend our problem definition to flips on cycles of fixed constant length ℓ, where $\ell \geq 4$ and ℓ is even. We refer to the corresponding flip operation as the ℓ-flip operation, and the corresponding reconfiguration problem as PERFECT MATCHING ℓ-FLIP RECONFIGURATION. It should be noted that the ℓ-flip operation must be applied to a cycle of length *exactly* ℓ, and hence there is no guarantee of the existence of a transformation.

1.2 Related work

Transformation of matchings has been considered under several types of flip operations for generating random matchings. Under the TS and TJ operations, numerous algorithms and hardness results are available for finding transformations between matchings, more generally, between independent sets. Furthermore, similar types of flip operations are well known for stable matchings and some geometric matching problems related to finding transformations between triangulations. More directly, a restriction of PERFECT MATCHING (4-FLIP) RECONFIGURATION to grid graphs has been considered before in the setting of domino tilings. In this restricted setting, Saldanha et al. [28] gave a criterion for the existence of a transformation between two tilings (which correspond to perfect matchings of a grid graph) and a formula for their distance of a transformation.

Sampling random perfect matchings

The problem of sampling or enumerating perfect matchings in a graph received considerable attention (see, e.g., [31]). Determining the connectivity and the diameter of the solution space formed by perfect matchings under the flip operation provide some information on the ergodicity or the mixing time of the underlying Markov chain. Indeed, the connectivity of the chain ensures the irreducibility (and usually the ergodicity) of the underlying Markov chain. Additionally, the diameter of the solution space provides a lower bound on the mixing time of the chain.

The use of flips for sampling random perfect matchings was first started in [8] where it is seen as a generalization of transpositions for permutations. Their work was later improved and generalized in [13] and [12]. The focus of these last two articles is to investigate the problem of sampling random perfect matchings using a Markov Chain called the switch chain. Starting from an arbitrary perfect matching, the chain proceeds by applying at each step a random 4-flip operation (called switch in these papers). The aim of these papers is to characterize classes of graphs for which simulating this chain for a polynomial number of steps is enough to generate a perfect matching close to uniformly distributed. Some of their results can be reformulated in the reconfiguration terminology. In [13], it is proved that the largest hereditary class of bipartite graphs for which the solution space formed by perfect matchings under the 4-flip operation is connected is the class of chordal bipartite graphs. This result is generalized in [12] where they characterize the hereditary class of general (non-bipartite) graphs for which the solution space is connected. They call this class Switchable. Note that it is not clear whether graphs in this class can be recognized in polynomial time. The question of the complexity of Perfect Matching (4-Flip) Reconfiguration is also mentioned in [12].

Reconfiguration of matchings and independent sets

Recall that matchings of a graph correspond to independent sets of its line graph. Although reconfiguration of independent sets received a considerable attention in the last decade (e.g., [5, 6, 7, 16, 22, 23, 33]), all the known results for reconfiguration of independent sets are based on the TJ or TS operations as adjacency relations. Thus, none of these results carry over to the Perfect Matching Reconfiguration problem.

A related problem can be found in a more general setting: The problem of determining, enumerating, or randomly generating graphs with a fixed degree sequence has received a considerable attention since the fifties (see, e.g., [30, 15, 32]). Given two graphs with a fixed degree sequence, one might want to know if it is possible to transform the one into the other via a sequence of 4-flip operations and if yes, how many steps are needed for such a transformation; note that the host graph (i.e., the graph G in our problem) is a clique in this setting. Hakimi [15] proved that such a transformation always exists. Will [32] proved that the problem of finding a shortest transformation is NP-complete, and Bereg and Ito [2] provide a $\frac{3}{2}$-approximation algorithm for this problem.

Stable matchings

Suppose we are given a bipartite graph and for each vertex a linear preference order of its neighbors. A matching M is not stable if there is an edge vw not in M such that v prefers w and w prefers v to their respective partners in M. The well-known algorithm by Gale and Shapley yields a stable matching in polynomial time [14]. It is known that any two stable matchings cover the same vertices, so the stable matchings are perfect matchings of some subgraph. Furthermore, they form a distributive lattice under *rotations* on preference-oriented cycles, see for example [14]. Essentially, the symmetric difference of two stable matchings consists of disjoint cycles (of several lengths) and we may exchange edges on these cycles to obtain another stable matching. If we drop the preferences, then the question is simply if we can find a transformation between two perfect matchings by exchanging edges on cycles in the symmetric difference. Clearly the answer is always yes, for example by processing the cycles in the symmetric difference one by one. We consider a similar setting, but restrict the length of the cycles.

Diagonal-flips of triangulations

A diagonal-flip of a triangulation in geometry is similar to our 4-flip operation in the sense that we switch between two states of a quadrilateral. In the context of triangulations, a diagonal-flip operation switches the diagonal of a quadrilateral. Transformations between triangulations of point sets and polygons have been studied mostly in the plane. It is known that the solution space formed by triangulations of point sets and polygons in the plane is connected and has diameter $O(n^2)$, where n is the number of points [20, 24]. Recently, NP-completeness has been proved for deciding the distance in the solution space between triangulations of a point set in the plane [25] and triangulations of a simple polygon [1].

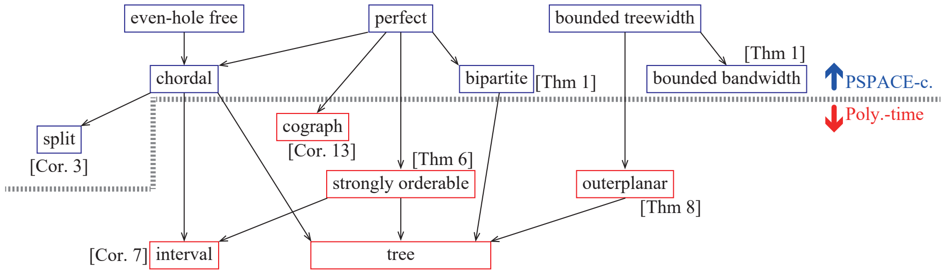


Figure 2 Our results, where each arrow represents the inclusion relationship between graph classes: $A \rightarrow B$ represents that the graph class B is properly included in the graph class A.

Houle et al. [18] have considered triangulations of point sets in the plane that admit a perfect matching. They show that any two such triangulations are connected under the diagonal-flip operation. For this purpose they consider the graph of non-crossing perfect matchings, where two matchings are adjacent if they differ by a single non-crossing cycle (of arbitrary length). They show that the graph of non-crossing perfect matchings is connected and conclude from this that any two triangulations that admit a perfect matching must be connected. In contrast to their setting, we remove all geometric requirements, but restrict the length of the cycles allowed for our flip operation.

1.3 Our results

In this paper, we study the complexity of Perfect Matching Reconfiguration from the viewpoint of graph classes. Figure 2 summarizes our results.

Recall that reconfiguration of matchings under the TS and TJ operations can be solved in polynomial time for any graph [21]. In contrast, we prove that Perfect Matching Reconfiguration is PSPACE-complete, even for split graphs, and for bipartite graphs of bounded bandwidth and of maximum degree five. We note that our hardness result for bipartite graphs gives contrast to chordal bipartite graphs for which there always exists a transformation between any two perfect matchings [13]. In addition, we extend our hardness result to a more general setting, namely the reconfiguration of k-factor subgraphs under the ℓ-flip operation for any fixed $k \geq 1$ and any fixed even integer $\ell \geq 4$.

We then investigate polynomial-time solvable cases. We prove that Perfect Matching Reconfiguration admits a polynomial-time algorithm on strongly orderable graphs (these include interval graphs and strongly chordal graphs), outerplanar graphs, and cographs (also known as P_4-free graphs). More specifically, we give the following results:

- For strongly orderable graphs, a transformation between two perfect matchings always exists; hence the answer is always yes. Furthermore, there is a transformation of linear length (i.e., a linear number of flip operations) between any two perfect matchings and such a transformation can be found in polynomial time.
- Perfect Matching Reconfiguration on outerplanar graphs can be solved in linear time, and we can find a transformation of linear length for a yes-instance in linear time. (Note that there are no-instances, e.g., long cycles).
- Perfect Matching Reconfiguration on cographs can be solved in polynomial time, and we can find a transformation of linear length for a yes-instance in polynomial time. (Again, there are no-instances).

Due to the page limitation, we omit proofs of the claims marked with (∗).

1.4 Notation

For standard definitions and notations on graphs, we refer the reader to [9]. Let $G = (V, E)$ be a simple graph. We sometimes denote by $V(G)$ and $E(G)$ the vertex set and edge set of G, respectively. A *matching* $M \subseteq E$ of G is a set of edges that share no endpoint. A vertex v is *covered* by a matching M if v is incident to an edge in M. For a vertex set $V' \subseteq V$, we denote by $G[V']$ the subgraph of G induced by V'. For a vertex set $W \subseteq V$, let $G - W := G[V \setminus W]$. For a vertex $v \in V$, we denote by $N(v)$ the *neighborhood* of v, that is, $N(v) := \{w \in V \mid vw \in E\}$.

Two perfect matchings M and M' of G are *adjacent* if their symmetric difference $M \triangle M'$ induces a cycle of length four. A sequence $M_0, M_1, \ldots, M_q$ of perfect matchings in G is called a *reconfiguration sequence between M and M'* if $M_0 = M$, $M_q = M'$, and M_{i-1} and M_i are adjacent for each i, $1 \leq i \leq q$. Given two perfect matchings M_{s} and M_{t} of a graph G, Perfect Matching Reconfiguration is to determine whether there exists a reconfiguration sequence between M_{s} and M_{t}. We denote by a triple $(G, M_{\mathrm{s}}, M_{\mathrm{t}})$ an instance of the problem.

2 PSPACE-completeness

In this section, we prove that perfect matching reconfiguration is PSPACE-complete. Interestingly, the problem remains intractable even for bipartite graphs, even though matchings in bipartite graphs satisfy several nice properties.

▶ **Theorem 1.** Perfect matching reconfiguration *is* PSPACE-*complete for bipartite graphs whose maximum degree is five and whose bandwidth is bounded by a fixed constant.*

Proof. Observe that the problem can be solved in (most conveniently, nondeterministic [29]) polynomial space, and hence it is in PSPACE. As a proof of Theorem 1, we thus prove that the problem is PSPACE-hard for such graphs, by giving a polynomial-time reduction from the Nondeterministic Constraint Logic problem (NCL for short) [16].

Definition of nondeterministic constraint logic

An NCL “machine” is an undirected graph together with an assignment of weights from $\{1, 2\}$ to each edge of the graph. An *(NCL) configuration* of this machine is an orientation (direction) of the edges such that the sum of weights of in-coming arcs at each vertex is at least two. Figure 3(a) illustrates a configuration of an NCL machine, where each weight-2 edge is depicted by a (blue) thick line and each weight-1 edge by a (red) thin line. Then, two NCL configurations are *adjacent* if they differ in a single edge direction. Given an NCL machine and its two configurations, it is known to be PSPACE-complete to determine whether there exists a sequence of adjacent NCL configurations which transforms one into the other [16].

An NCL machine is called an and/or *constraint graph* if it consists of only two types of vertices, called “NCL and vertices” and “NCL or vertices” defined as follows: A vertex of degree three is called an *NCL* and *vertex* if its three incident edges have weights 1, 1, and 2. (See Figure 3(b).) An NCL and vertex u behaves as a logical and, in the following sense: the weight-2 edge can be directed outward for u only if both two weight-1 edges are directed inward for u. Note that, however, the weight-2 edge is not necessarily directed outward even when both weight-1 edges are directed inward. A vertex of degree three is called an *NCL* or *vertex* if its three incident edges have weights 2, 2, and 2. (See Figure 3(c).) An NCL

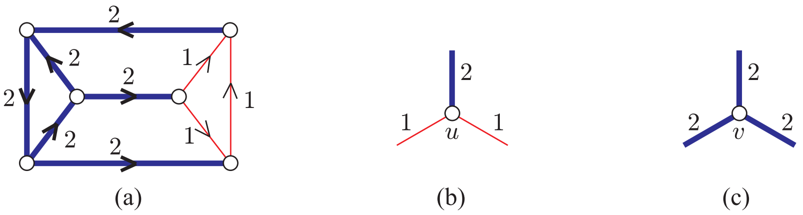


Figure 3 (a) A configuration of an NCL machine, (b) an NCL AND vertex u, and (c) an NCL OR vertex v.

OR vertex v behaves as a logical OR: one of the three edges can be directed outward for v if and only if at least one of the other two edges is directed inward for v. It should be noted that, although it is natural to think of NCL AND/OR vertices as having inputs and outputs, there is nothing enforcing this interpretation; especially for NCL OR vertices, the choice of input and output is entirely arbitrary because an NCL OR vertex is symmetric. For example, the NCL machine in Figure 3(a) is an AND/OR constraint graph. From now on, we call an AND/OR constraint graph simply an *NCL machine*, and call an edge in an NCL machine an *NCL edge*. NCL remains PSPACE-complete even if an input NCL machine is planar, bounded bandwidth, and of maximum degree three [34].

Gadgets and reduction

Suppose that we are given an instance of NCL, that is, an NCL machine and two configurations of the machine. We will replace each of the NCL edges and NCL AND/OR vertices with its corresponding gadget; if an NCL edge e is incident to an NCL vertex v, then we connect the corresponding gadgets for e and v by a pair of vertices, called *connectors* (*between* v *and* e) or (v, e)-*connectors*, as illustrated in Figure 4(a) and (b). Thus, each edge gadget has two pairs of connectors, and each AND/OR gadget has three pairs of connectors. Our gadgets are all edge-disjoint, and share only connectors.

Figure 5 shows our three types of gadgets which correspond to NCL edges and NCL AND/OR vertices. As illustrated in Figure 4, we replace each of the NCL edges and NCL AND/OR vertices with its corresponding gadget; let G be the resulting graph. Notice that each of our three gadgets is of maximum degree three, and connectors in the edge gadget are of degree two; thus, G is of maximum degree five. In addition, each of our three gadgets is a bipartite graph such that two connectors in the same pair belong to different sides of the bipartition; therefore, G is bipartite. Furthermore, since NCL remains PSPACE-complete even if an input NCL machine is bounded bandwidth [34], the resulting graph G is also bounded bandwidth and of maximum degree five; notice that, since each gadget consists of only a constant number of edges, the bandwidth of G is also bounded.

We next construct two perfect matchings of G which correspond to two given NCL configurations C_s and C_t of the NCL machine. In our reduction, we construct the correspondence between orientations of an NCL machine and perfect matchings of the corresponding graph, as follows: We regard that the orientation of an NCL edge $e = vw$ is inward direction for v if the two (v, e)-connectors are both covered by (edges in) the AND/OR gadget for v. On the other hand, we regard that the orientation of $e = vw$ is outward direction for w if the two (w, e)-connectors are both covered by the edge gadget for e. To achieve this correspondence, our gadgets are constructed so that both two (v, e)-connectors are always covered by exactly one of the gadgets for v and e. Note that there are (in general, exponentially) many perfect matchings which correspond to the same NCL configuration. However, by the construction of

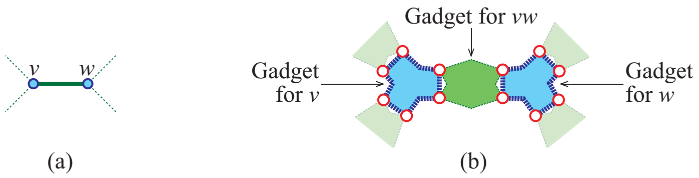


Figure 4 (a) An NCL edge vw, and (b) its corresponding gadgets, where the connectors are depicted by (red) circles.

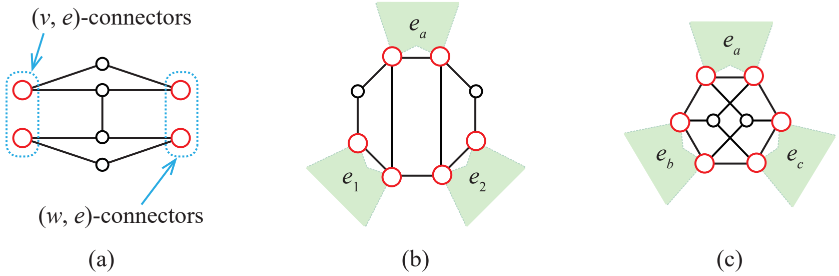


Figure 5 Illustrations of (a) an edge gadget for an NCL edge $e = vw$, (b) an AND gadget, and (c) an OR gadget. In the AND/OR gadget, the three light green parts represent the edge gadgets corresponding to the edges incident to the NCL vertex; e_1 and e_2 in the AND gadget correspond to weight-1 edges.

the three gadgets, no two distinct NCL configurations correspond to the same perfect matching of G. We arbitrarily choose two perfect matchings M_{s} and M_{t} of G which correspond to C_{s} and C_{t}, respectively.

This completes the construction of our corresponding instance of PERFECT MATCHING RECONFIGURATION. The construction can be done in polynomial time. Furthermore, the following lemma gives the correctness of our reduction.

▶ **Lemma 2** (∗). *There exists a desired sequence of NCL configurations between C_{s} and C_{t} if and only if there exists a reconfiguration sequence between M_{s} and M_{t}.*

This completes the proof of Theorem 1. ◀

Remarks

We conclude this section by giving some remarks that can be obtained from Theorem 1. We first prove that the problem remains intractable even for split graphs. A graph is *split* if its vertex set can be partitioned into a clique and an independent set.

▶ **Corollary 3.** PERFECT MATCHING RECONFIGURATION *is* PSPACE*-complete for split graphs.*

Proof. By Theorem 1 the problem remains PSPACE-complete for bipartite graphs. Consider the graph obtained by adding new edges so that one side of the bipartition forms a clique. The resulting graph is a split graph. We claim that these new edges can never be part of any perfect matching of the graph. Indeed, since the original graph was bipartite, there must be the same number of vertices on each side of the bipartition. In a perfect matching of the split graph, all the vertices from the independent set must be matched with vertices from the clique, and no vertex from the clique remains to be matched together. Thus, the claim holds, and hence the corollary follows. ◀

We finally extend Theorem 1 into two directions: regular spanning subgraphs and flip operations on alternating cycles of a fixed length. Let k be a positive integer. A spanning subgraph H of a graph G is a *k-factor* if all the vertices in H have degree exactly k. Thus, a 1-factor of G is a perfect matching of G. Based on the proof of Theorem 1, we can obtain the following theorem.

▶ **Theorem 4** ($*$). *Let $k \geq 1$ be a fixed integer, and let $\ell \geq 4$ be a fixed even integer. Given two k-factors H_{s} and H_{t} of a graph G, it is* PSPACE*-complete to decide if there is a sequence of ℓ-flip operations transforming H_{s} into H_{t}.*

3 Polynomial-time algorithms

In this section, we investigate the polynomial-time solvability of PERFECT MATCHING RECONFIGURATION from the viewpoint of graph classes. We first give the following lemma, which holds for any graph.

▶ **Lemma 5.** *It suffices to solve* PERFECT MATCHING RECONFIGURATION *for* 2*-connected graphs having at least four vertices.*

Proof. Let $(G, M_{\mathrm{s}}, M_{\mathrm{t}})$ be a given instance of PERFECT MATCHING RECONFIGURATION. If G is not connected, then we can simply consider each connected component separately.

Since the input graph $G = (V, E)$ has a perfect matching, it has an even number of vertices. If $|V| = 2$, then it must hold that $M_{\mathrm{s}} = M_{\mathrm{t}} = E$, and hence the instance is trivially a yes-instance. Therefore, it suffices to solve the problem for $|V| \geq 4$.

Suppose that G is not 2-connected and has a cut vertex $v \in V$, that is, $G - \{v\}$ consists of more than one connected component. Since $|V|$ is even, there exists a vertex subset $X \subseteq V \setminus \{v\}$ such that $|X|$ is odd and $G[X]$ forms a connected component of $G - \{v\}$. Then, any perfect matching in G contains an edge connecting v and X. This shows that we can consider two subgraphs $G_1 := G[X \cup \{v\}]$ and $G_2 := G - (X \cup \{v\})$, separately. That is, we output "yes" if $(G_i, M_{\mathrm{s}} \cap E_i, M_{\mathrm{t}} \cap E_i)$ is a yes-instance for every $i \in \{1, 2\}$, where E_i is the edge set of G_i, and output "no" otherwise. Thus, the lemma follows. ◀

3.1 Strongly orderable graphs

Interval graphs form easy instances for many NP-hard problems, and the situation is no different here. In fact, we prove that any instance on an interval graph is a yes-instance. Our argument also yields a linear-time algorithm to compute a reconfiguration sequence of a linear number of flip operations between any two perfect matchings.

For the sake of generality, we consider a wider class of graphs, called strongly orderable graphs. A graph $G = (V, E)$ is *strongly orderable* if there is a *strong ordering* on its vertices, defined as follows: an order $(v_1, v_2, \ldots, v_n)$ of V such that for every i, j, k, ℓ with $i < j$ and $k < \ell$, if all of $v_i v_k, v_i v_\ell$ and $v_j v_k$ are edges, then $v_j v_\ell$ is an edge. Note that the class of strongly orderable graphs is hereditary: every induced subgraph of a strongly orderable graph is strongly orderable.

Our proof strategy for the following theorem is to show that every perfect matching N of a strongly orderable graph G can be transformed into some particular perfect matching M of G, called the canonical perfect matching; then, any two perfect matchings N and N' of G admit a reconfiguration sequence between them via M. The *canonical perfect matching* of a graph G with respect to an order $\mathcal{O} = (v_1, v_2, \ldots, v_n)$ is a perfect matching of G (if any) greedily obtained by selecting, among the available edges, the one with endpoints of smallest indices. Note that any strongly orderable graph that admits a perfect matching, also admits a canonical perfect matching with respect to a corresponding order on the vertices (see, e.g., [10]). We give the following theorem in this subsection.

▶ **Theorem 6** (∗). *Let G be a strongly orderable graph. Then, there is a reconfiguration sequence of linear length between any two perfect matchings of G. Furthermore, such a reconfiguration sequence can be found in linear time if we are given a strong ordering on the vertices of G as a part of the input.*

The natural question regarding Theorem 6 is whether a strong ordering can be computed efficiently. In general, Dragan [11] proved that strongly orderable graphs $G = (V, E)$ can be recognized in $O(|V| \cdot (|V| + |E|))$ time, and if so we can obtain its strong ordering in the same running time. However, when restricted to interval graphs, we can obtain a strong ordering in linear time [19]. We thus have the following corollary.

▶ **Corollary 7.** *Let G be an interval graph. Then, there is a reconfiguration sequence of linear length between any two perfect matchings of G. Furthermore, such a reconfiguration sequence can be found in linear time.*

3.2 Outerplanar graphs

In this subsection, we consider outerplanar graphs. Note that there are no-instances for outerplanar graphs, e.g., induced cycles of even length more than four. Nonetheless, we give the following theorem.

▶ **Theorem 8.** Perfect Matching Reconfiguration *can be solved in linear time for outerplanar graphs. Moreover, for a* yes*-instance, a reconfiguration sequence of linear length can be output in linear time.*

We give such an algorithm as a proof of Theorem 8. Suppose we are given a simple outerplanar graph $G = (V, E)$, and two perfect matchings M_s and M_t in G. By Lemma 5 we assume without loss of generality that G is 2-connected and $|V| \geq 4$. Then, G has a planar embedding such that the outer face boundary is a simple cycle and all the vertices of G are on the outer face boundary. Suppose that the vertices $v_1, v_2, \ldots, v_n$ appear in this order along the cycle. For notational convenience, we denote $v_{n+1} = v_1$, $v_{n+2} = v_2$, and $v_0 = v_n$. We first give the following assumption without loss of generality.

▶ **Lemma 9.** Perfect Matching Reconfiguration *for outerplanar graphs can be reduced to the case when $v_i v_j \notin E$ holds for any pair of indices $i, j \in \{1, 2, \ldots, n\}$ such that $|i - j|$ is even. In particular, $v_i v_{i+2} \notin E$ holds for any $i \in \{1, 2, \ldots, n\}$.*

Proof. Suppose that there exists a pair of indices $i, j \in \{1, 2, \ldots, n\}$ such that $|i - j|$ is even and $v_i v_j \in E$. Then, the edge $v_i v_j$ cannot be contained in any perfect matching of G, because $G[\{v_{i+1}, v_{i+2}, \ldots, v_{j-1}\}]$ forms a connected component in $G - \{v_i, v_j\}$ even though it contains an odd number of vertices. Therefore, we can remove $v_i v_j$ from G. ◀

We now show the following lemma for an outerplanar graph $G = (V, E)$.

▶ **Lemma 10** (∗). *If $v_i v_{i+2} \notin E$ for any $i \in \{1, 2, \ldots, n\}$, then there exists an index $k \in \{1, 2, \ldots, n\}$ such that both v_k and v_{k+1} are of degree two.*

Let $k \in \{1, 2, \ldots, n\}$ be an index such that both v_k and v_{k+1} are of degree two, and let $e_i := v_i v_{i+1}$ for each $i \in \{k - 1, k, k + 1\}$. Note that if a perfect matching of G does not contain $e_k = v_k v_{k+1}$, then it has to contain both $e_{k-1} = v_{k-1} v_k$ and $e_{k+1} = v_{k+1} v_{k+2}$. We consider the following two cases separately.

Case 1: We first consider the case with $v_{k-1}v_{k+2} \notin E$. In this case, we can see that e_k is not contained in any cycles of length four, and hence e_k is never touched by any flip operation. Thus, we consider one of the following three sub-cases.

- If $e_k \in M_{\mathrm{s}} \triangle M_{\mathrm{t}}$, then we can immediately conclude that $(G, M_{\mathrm{s}}, M_{\mathrm{t}})$ is a no-instance.
- If $e_k \in M_{\mathrm{s}} \cap M_{\mathrm{t}}$, then we solve the smaller instance $(G - \{v_k, v_{k+1}\}, M_{\mathrm{s}} \setminus \{e_k\}, M_{\mathrm{t}} \setminus \{e_k\})$.
- If $e_k \notin M_{\mathrm{s}} \cup M_{\mathrm{t}}$, then we solve the smaller instance $(G - \{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}, M_{\mathrm{s}} \setminus \{e_{k-1}, e_{k+1}\}, M_{\mathrm{t}} \setminus \{e_{k-1}, e_{k+1}\})$.

Case 2: We next consider the case with $v_{k-1}v_{k+2} \in E$. Then, $G[\{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}]$ forms a cycle of length four. For each $i \in \{\mathrm{s}, \mathrm{t}\}$, we define

$$M'_i := \begin{cases} M_i \setminus \{e_k\} & \text{if } e_k \in M_i, \\ (M_i \setminus \{e_{k-1}, e_{k+1}\}) \cup \{v_{k-1}v_{k+2}\} & \text{otherwise.} \end{cases}$$

Let $G' = G - \{v_k, v_{k+1}\}$, and we solve the smaller instance $(G', M'_{\mathrm{s}}, M'_{\mathrm{t}})$.

In either case, we can reduce the original instance $(G, M_{\mathrm{s}}, M_{\mathrm{t}})$ to a smaller instance, which implies that our algorithm runs in polynomial time. Indeed, we can implement the above arguments so that the algorithm runs in linear time. (The details are omitted.) The correctness of Case 1 is obvious, while the correctness of Case 2 is guaranteed as follows.

▶ **Lemma 11** (∗). *$(G, M_{\mathrm{s}}, M_{\mathrm{t}})$ is a* yes*-instance if and only if $(G', M'_{\mathrm{s}}, M'_{\mathrm{t}})$ is a* yes*-instance.*

This completes the proof of Theorem 8. ◀

3.3 Cographs

We consider cographs in this subsection. Cographs, also known as P_4-free graphs, are graphs without a path on four vertices as an induced subgraph. As examples concerning reconfiguration on this class of graphs, it is known that the problems INDEPENDENT SET RECONFIGURATION and STEINER TREE RECONFIGURATION can be solved in polynomial time for cographs [3, 4, 26], while they are PSPACE-complete for general graphs [21, 26]. We will show that the situation is similar for PERFECT MATCHING RECONFIGURATION.

To describe our algorithm for cographs, we generalize our problem to non-perfect matchings. In the generalized problem, we regard that two matchings are *adjacent* if their symmetric difference is either a cycle of length four, or a path on three vertices. Note that any two adjacent matchings have the same size. Then, the generalized problem is defined as follows:

GENERAL MATCHING RECONFIGURATION
Input: Graph G, two matchings M_{s} and M_{t} of G.
Question: Is there a sequence of adjacent matchings that transforms M_{s} into M_{t}?

An instance of the generalized problem is also denoted by a triple $(G, M_{\mathrm{s}}, M_{\mathrm{t}})$, and a sequence of adjacent matchings is also called a *reconfiguration sequence*. $(G, M_{\mathrm{s}}, M_{\mathrm{t}})$ is clearly a no-instance if $|M_{\mathrm{s}}| \neq |M_{\mathrm{t}}|$, and hence we assume that $|M_{\mathrm{s}}| = |M_{\mathrm{t}}|$ holds.

Our main result of this subsection is the following theorem.

▶ **Theorem 12** (∗). GENERAL MATCHING RECONFIGURATION *can be solved in polynomial time for cographs. Moreover, for a* yes*-instance, a reconfiguration sequence of linear length can be output in polynomial time.*

We note that when two input matchings are perfect, every connected component in their symmetric difference is a cycle. Therefore, even in the generalized problem, two adjacent perfect matchings differ by a flip operation on a cycle of length four. We thus obtain the following result as a corollary of Theorem 12.

▶ **Corollary 13.** Perfect Matching Reconfiguration *can be solved in polynomial time for cographs. Moreover, for a* yes*-instance, a reconfiguration sequence of linear length can be output in polynomial time.*

As a proof of Theorem 12, we give such an algorithm in this subsection. We will use the recursive characterization of cographs. For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, their *disjoint union* $G_1 \cup G_2$ is the graph such that $V(G_1 \cup G_2) = V_1 \cup V_2$ and $E(G_1 \cup G_2) = E_1 \cup E_2$, while their *complete join* $G_1 \vee G_2$ is the graph such that $V(G_1 \vee G_2) = V_1 \cup V_2$ and $E(G_1 \vee G_2) = E_1 \cup E_2 \cup \{vw \mid v \in V_1, w \in V_2\}$. Then, a *cograph* can be recursively defined, as follows:

- a graph consisting of a single vertex is a cograph;
- if G_1 and G_2 are cographs, then their disjoint union $G_1 \cup G_2$ is a cograph; and
- if G_1 and G_2 are cographs, then their complete join $G_1 \vee G_2$ is a cograph.

Let $(G, M_\mathrm{s}, M_\mathrm{t})$ be a given instance of General Matching Reconfiguration such that $G = (V, E)$ is a cograph. By Lemma 5 we assume without loss of generality that G is connected and $|V| \geq 4$, and hence $G = G_1 \vee G_2$ for two cographs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Assume that $|V_1| \geq |V_2|$, and let $k := |M_\mathrm{s}| = |M_\mathrm{t}|$. We first give a sufficient condition for which there exists a reconfiguration sequence between M_s and M_t.

▶ **Lemma 14** (∗)**.** *There is a reconfiguration sequence between M_s and M_t if G has a matching M of size k such that at least one of the following two conditions holds:*
(C1) *$M \cap E_2 \neq \emptyset$; and*
(C2) *at least one vertex of G_2 is not covered by M.*
Furthermore, a reconfiguration sequence of linear length can be output in polynomial time.

We claim that this sufficient condition can be checked in polynomial time. The existence of a matching satisfying the condition **(C1)** can be checked as follows: For each edge $vw \in E_2$, we check if the graph $G - \{v, w\}$ has a matching of size $k - 1$, or not. This can be done in polynomial time since a maximum matching in a graph can be computed in polynomial time. Similarly, the condition **(C2)** can be checked in polynomial time, as follows: For each vertex $v \in V_2$, we check if the graph $G - \{v\}$ has a matching of size k, or not.

We then consider the case where the sufficient condition of Lemma 14 does not hold. Recall that $G = G_1 \vee G_2$ and $|V_1| \geq |V_2|$.

▶ **Lemma 15** (∗)**.** *Suppose that G does not have a matching of size k satisfying the conditions* **(C1)** *or* **(C2)** *of Lemma 14. Then, the following two claims hold.*

- *$(G, M_\mathrm{s}, M_\mathrm{t})$ is a* yes*-instance if and only if $(G_1, M_\mathrm{s} \cap E_1, M_\mathrm{t} \cap E_1)$ is a* yes*-instance.*
- *For a* yes*-instance $(G, M_\mathrm{s}, M_\mathrm{t})$, a reconfiguration sequence between M_s and M_t of linear length can be output in polynomial time.*

The above arguments can be implemented so that the algorithm runs in polynomial time, since we reduce the original instance $(G, M_\mathrm{s}, M_\mathrm{t})$ to a smaller instance $(G_1, M_\mathrm{s} \cap E_1, M_\mathrm{t} \cap E_1)$.

This completes the proof of Theorem 12. ◀

4 Conclusion

We introduced the Perfect Matching Reconfiguration problem and analyzed its complexity from the viewpoint of graph classes. We showed that this problem is PSPACE-complete on split graphs and bipartite graphs of bounded bandwidth and maximum degree five.

Furthermore, we gave polynomial-time algorithms for strongly orderable graphs, outerplanar graphs, and cographs. Each of the algorithms outputs a reconfiguration sequence of linear length in polynomial time.

A natural open question is on which graph classes a *shortest* reconfiguration sequence can be found in polynomial time. Furthermore, it would be interesting to investigate if the flip operation can be used in order to sample perfect matchings uniformly.

References

1 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.

2 Sergey Bereg and Hiro Ito. Transforming Graphs with the Same Graphic Sequence. *Journal of Information Processing*, 25:627–633, 2017.

3 Marthe Bonamy and Nicolas Bousquet. Reconfiguring independent sets in cographs. *arXiv preprint arXiv:1406.1433*, 2014.

4 Paul Bonsma. Independent set reconfiguration in cographs and their generalizations. *Journal of Graph Theory*, 83(2):164–195, 2016.

5 Paul Bonsma, Marcin Kamiński, and Marcin Wrochna. Reconfiguring Independent Sets in Claw-Free Graphs. In *Proceedings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2014)*, volume 8503 of *Lecture Notes in Computer Science*, pages 86–97, 2014.

6 Nicolas Bousquet, Arnaud Mary, and Aline Parreau. Token Jumping in Minor-Closed Classes. In *Proceedings of the 21st International Symposium on Fundamentals of Computation Theory (FCT 2017)*, volume 10472 of *Lecture Notes in Computer Science*, pages 136–149, 2017.

7 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theoretical Computer Science*, 600:132–142, 2015.

8 Persi Diaconis, Ronald Graham, and Susan P. Holmes. *Statistical problems involving permutations with restricted positions*, volume 36 of *Lecture Notes–Monograph Series*, pages 195–222. Institute of Mathematical Statistics, Beachwood, OH, 2001.

9 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, third edition, 2005.

10 Feodor F. Dragan. On Greedy Matching Ordering and Greedy Matchable Graphs (Extended Abstract). In *Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1997)*, volume 1335 of *Lecture Notes in Computer Science*, pages 184–198, 1997.

11 Feodor F. Dragan. Strongly orderable graphs A common generalization of strongly chordal and chordal bipartite graphs. *Discrete Applied Mathematics*, 99(1–3):427–442, 2000.

12 Martin Dyer and Haiko Müller. Counting perfect matchings and the switch chain. *arXiv preprint arXiv:1705.05790*, 2017.

13 Martin E. Dyer, Mark Jerrum, and Haiko Müller. On the Switch Markov Chain for Perfect Matchings. *Journal of the ACM*, 64(2):12:1–12:33, 2017.

14 Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT press, 1989.

15 S. L. Hakimi. On Realizability of a Set of Integers as Degrees of the Vertices of a Linear Graph II. Uniqueness. *Journal of the Society for Industrial and Applied Mathematics*, 11(1):135–147, 1963.

16 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1–2):72–96, 2005.

17 Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, pages 127–160. Cambridge University Press, 2013.

18 Michael E. Houle, Ferran Hurtado, Marc Noy, and Eduardo Rivera-Campo. Graphs of triangulations and perfect matchings. *Graphs and Combinatorics*, 21(3):325–331, 2005.

19 Wen-Lian Hsu. A Simple Test for Interval Graphs. In *Proceedings of the 18th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1992)*, volume 657 of *Lecture Notes in Computer Science*, pages 11–16, 1992.

20 Ferran Hurtado, Marc Noy, and Jorge Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, 1999.

21 Takehiro Ito, Erik D. Demaine, Nicholas J.A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011.

22 Takehiro Ito, Marcin Kamiński, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. On the Parameterized Complexity for Token Jumping on Graphs. In *Proceedings of the 11th Annual Conference on Theory and Applications of Models of Computation (TAMC 2014)*, volume 8402 of *Lecture Notes in Computer Science*, pages 341–351, 2014.

23 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012.

24 Charles L. Lawson. Software for C_1 surface interpolation. In *Mathematical software*, pages 161–194. Elsevier, 1977.

25 Anna Lubiw and Vinayak Pathak. Flip distance between two triangulations of a point set is NP-complete. *Computational Geometry*, 49:17–23, 2015.

26 Haruka Mizuta, Takehiro Ito, and Xiao Zhou. Reconfiguration of Steiner Trees in an Unweighted Graph. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 100-A(7):1532–1540, 2017.

27 Naomi Nishimura. Introduction to Reconfiguration. *Algorithms*, 11(4):52, 2018.

28 Nicolau C. Saldanha, Carlos Tomei, Mario A. Casarin, and Domingos Romualdo. Spaces of domino tilings. *Discrete & Computational Geometry*, 14(2):207–233, 1995.

29 Walter J. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

30 James K. Senior. Partitions and Their Representative Graphs. *American Journal of Mathematics*, 73(3):663–689, 1951.

31 Daniel Stefankovic, Eric Vigoda, and John Wilmes. On Counting Perfect Matchings in General Graphs. In *Proceedings of the 13th Latin American Theoretical Informatics Symposium (LATIN 2018)*, volume 10807 of *Lecture Notes in Computer Science*, pages 873–885, 2018.

32 Todd G. Will. Switching Distance Between Graphs with the Same Degrees. *SIAM Journal on Discrete Mathematics*, 12(3):298–306, 1999.

33 Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *Journal of Computer and System Sciences*, 93:1–10, 2018.

34 Tom C. van der Zanden. Parameterized Complexity of Graph Constraint Logic. In *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, volume 43 of *Leibniz International Proceedings in Informatics*, pages 282–293, 2015.

Spectral Aspects of Symmetric Matrix Signings

Charles Carlson
University of Colorado Boulder, Boulder, USA
charles.carlson@colorado.edu

Karthekeyan Chandrasekaran
University of Illinois, Urbana-Champaign, USA
karthe@illinois.edu

Hsien-Chih Chang
Duke University, Durham, USA
hsienchih.chang@duke.edu

Naonori Kakimura
Keio University, Yokohama, Japan
kakimura@math.keio.ac.jp

Alexandra Kolla
University of Colorado Boulder, Boulder, USA
alexandra.kolla@colorado.edu

Abstract

The spectra of signed matrices have played a fundamental role in social sciences, graph theory, and control theory. In this work, we investigate the computational problems of *finding* symmetric signings of matrices with natural spectral properties. Our results are the following:

1. We characterize matrices that have an invertible signing: a symmetric matrix has an invertible symmetric signing *if and only if* the support graph of the matrix contains a perfect 2-matching. Further, we present an efficient algorithm to search for an invertible symmetric signing.
2. We use the above-mentioned characterization to give an algorithm to find a minimum increase in the support of a given symmetric matrix so that it has an invertible symmetric signing.
3. We show NP-completeness of the following problems: verifying whether a given matrix has a symmetric signing that is singular or has bounded eigenvalues. However, we also illustrate that the complexity could differ substantially for input matrices that are adjacency matrices of graphs.

We use combinatorial techniques in addition to classic results from matching theory.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases Spectral Graph Theory, Matrix Signing, Matchings

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.81

Related Version An earlier version of this work using alternative non-constructive proof techniques is available at `https://arxiv.org/abs/1611.03624`.

Funding *Karthekeyan Chandrasekaran*: Supported by NSF CCF 18-14613.
Naonori Kakimura: Partly supported by JSPS KAKENHI Grant Numbers JP17K00028 and JP18H05291.
Alexandra Kolla: Supported by NSF CCF 1855919

1 Introduction

The spectra of several graph-related matrices such as the adjacency and the Laplacian matrices have become fundamental objects of study in computer science. In this work, we undertake a systematic and comprehensive investigation of the spectrum and the invertibility of *symmetric signings* of matrices. We study natural spectral properties of symmetric signed

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 81; pp. 81:1–81:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

matrices and address the computational problems of finding and verifying the existence of symmetric signings with these spectral properties.

For a real-valued symmetric $n \times n$ matrix M and a $\{\pm 1\}$-valued $n \times n$ matrix s – which we refer to as a *signing* – we define the *signed matrix* $M(s)$ to be the matrix obtained by taking entry-wise products of M and s. Signed adjacency matrices (respectively, Laplacians) correspond to signed matrices $M(s)$ where M is the adjacency matrix (respectively, Laplacian) of a graph. We say that s is a *symmetric* signing if s is a symmetric matrix and an *off-diagonal* signing if all the diagonal entries of s are $+1$. In this work we consider the following computational problems:

BoundedEvalueSigning: Given a real symmetric matrix M and a real number λ, verify if there exists an off-diagonal symmetric signing s such that the largest eigenvalue $\lambda_{\max}(M(s))$ is at most λ.

IncludeSigning: Given a real symmetric matrix M and a real number λ, verify if there exists an off-diagonal symmetric signing s such that $M(s)$ has λ as an eigenvalue.

AvoidSigning: Given a real symmetric matrix M and a real number λ, verify if there exists a symmetric signing s such that $M(s)$ does not have λ as an eigenvalue.

It suffices to focus on instances where λ is 0. Indeed, solving an instance of one of the above problems on input (M, λ) corresponds exactly to solving the same problem on input $(M - \lambda I, 0)$. Hence, we focus our attention on the corresponding specialized problems:

NsdSigning: Given a real symmetric matrix M, verify if there exists a symmetric signing s such that $M(s)$ is negative semi-definite.

SingularSigning: Given a real symmetric matrix M, verify if there exists an off-diagonal symmetric signing s such that $M(s)$ is singular.

InvertibleSigning: Given a real symmetric matrix M, verify if there exists a symmetric signing s such that $M(s)$ is invertible (that is, non-singular).

1.1 Motivations

Spectra of Signed Matrices and Expanders. Let G be a connected d-regular graph on n vertices and let $d = \lambda_0 > \lambda_1 \geq \cdots \geq \lambda_{n-1}$ be the eigenvalues of its adjacency matrix. Then G is a Ramanujan expander if $\max_{|\lambda_i|<d} |\lambda_i| \leq 2\sqrt{d-1}$. Efficient construction of Ramanujan expanders of arbitrary degrees remains an important open problem.[1] A combinatorial approach to this problem, initiated by Friedman [9], is to obtain larger Ramanujan graphs from smaller ones while preserving the degree. A 2-*lift* H of G is obtained as follows: Introduce two copies of each vertex u of G, say u_1 and u_2, as the vertices of H and for each edge $\{u, v\}$ in G, introduce either $\{u_1, v_2\}, \{u_2, v_1\}$ or $\{u_1, v_1\}, \{u_2, v_2\}$ as edges of H. There is a bijection between 2-lifts and symmetric signed adjacency matrices of G. Furthermore, the eigenvalues of the adjacency matrix of a 2-lift H are given by the union of the eigenvalues of the adjacency matrix of the base graph G (also called the “old” eigenvalues) and the signed adjacency matrix of G that corresponds to the 2-lift. (the “new” eigenvalues).

Marcus, Spielman, and Srivastava [16] showed that every d-regular bipartite graph has a 2-lift whose new eigenvalues are bounded in absolute value by $2\sqrt{d-1}$. However, this result [16] is not constructive and their work raises the question of whether there is an efficient

[1] While efficient construction of bipartite Ramanujan *multi-graphs* of all degrees is known [5], it still remains open to efficiently construct bipartite Ramanujan *simple* graphs of all degrees.

algorithm to find a symmetric signing that minimizes the largest eigenvalue. This motivates investigating BoundedEvalueSigning which is the decision variant of the computational problem. More precisely, it motivates investigating BoundedEvalueSigning when the input matrix is an adjacency matrix.

It is also natural to investigate the complexity of several related problems. As we will see in the next section, BoundedEvalueSigning is NP-hard for arbitrary symmetric matrices. The reduction which shows BoundedEvalueSigning is NP-hard suggests a close relationship with AvoidSigning which is also NP-hard. Hoping to make progress on BoundedEvalueSigning for adjacency matrices, we investigate AvoidSigning for adjacency matrices. IncludeSigning arises naturally as the complement of AvoidSigning.

Solvability Index of a Signed Matrix. The notion of *balance* of a symmetric signed matrix has been studied extensively in social sciences [14, 11, 13, 17]. A signed adjacency matrix is *balanced* if there is a partition of the vertex set such that all edges within each part are positive, and all edges in between two parts are negative (one of the parts could be empty). A number of works [3, 10, 18, 17, 19, 20] have explored the problem of minimally modifying signed graphs (or signed adjacency matrices) to convert it into a balanced graph.

In this work, we introduce a related problem regarding symmetric signed matrices: Given a symmetric matrix M, what is the smallest number of non-diagonal zero entries of M whose replacement by non-zeroes gives a symmetric matrix M' that has an invertible symmetric signing? We define this quantity to be the *solvability index*[2]. Knowing this number might be helpful in studying systems of linear equations in signed matrices that might be ill-defined, and thus do not have a (unique) solution and in minimally modifying such matrices so that the resulting linear system becomes (uniquely) solvable. We use classic graph-theoretic techniques to show that solvability index is indeed computable efficiently.

1.2 Our Results

Intriguingly, the complexity of BoundedEvalueSigning has not been studied in the literature even though it is widely believed to be a difficult problem in the graph sparsification community. We shed light on this problem by showing that it is NP-complete.

▶ **Theorem 1.** NsdSigning *and* SingularSigning *are NP-complete.*

Theorem 1 also implies that BoundedEvalueSigning and AvoidSigning are NP-complete. In contrast to SingularSigning, we show that InvertibleSigning admits an efficient algorithm. In fact, we show a stronger result: there exists an algorithm to efficiently solve the search variant of InvertibleSigning, which we denote by SearchInvertibleSigning (here the goal is to *find* an invertible signing if it exists).

▶ **Theorem 2.** *There exists a polynomial-time algorithm to solve* SearchInvertibleSigning*.*

Theorem 2 also implies that the search variant of IncludeSigning is solvable efficiently. Our proof of Theorem 2 leads to a structural characterization for the existence of invertible signings through the existence of *perfect* 2*-matchings* in the support graph of the matrix.

[2] Our definition of *solvability index* is similar to the notion of *frustration index* [12, 1]. The *frustration index* of a matrix M is the minimum number of non-zero off-diagonal entries of M whose deletion results in a *balanced signed graph*. Computing the frustration index of a signed graph is NP-hard [15].

We believe that this structural characterization could be of independent interest and hence, discuss it in detail in Section 1.2.1.

The hard instances generated by our proof of Theorem 1 are real symmetric matrices with non-zero diagonal entries and hence, it does not resolve the computational complexity of the problem of finding a signing of a given *graph-related* matrix (for example, the adjacency matrix) that minimizes its largest eigenvalue. Our next result provides some evidence that one might be able to design efficient algorithms to solve the NP-complete problems appearing in Theorem 1 for graph-related matrices. In particular, we show that SINGULARSIGNING and its search variant admit efficient algorithms when the input matrix corresponds to the adjacency matrix of a *bipartite* graph.

▶ **Theorem 3.** *There exists a polynomial-time algorithm to verify if the adjacency matrix A_G of a given bipartite graph G has a symmetric signing s such that $A_G(s)$ is singular; and if so, find such a signing.*

Finally, we define the *solvability index* of a real symmetric matrix M to be the smallest number of non-diagonal zero entries that need to be converted to non-zeroes so that the resulting symmetric matrix has an invertible symmetric signing. We emphasize that the support-increase operation that we consider preserves symmetry, that is, if we replace the zero entry $A[i,j]$ by α, then the zero entry $A[j,i]$ is also replaced by α. We give an efficient algorithm to find the solvability index of a given symmetric matrix M.

▶ **Theorem 4.** *There exists a polynomial-time algorithm to find the solvability index of a given real symmetric matrix.*

1.2.1 Structural Characterization for Invertible Signings

Theorem 2, in particular, implies that INVERTIBLESIGNING is solvable efficiently. In fact, our proof-technique gives an efficient characterization for the existence of an invertible signing. This characterization also leads to an alternative algorithm to solve INVERTIBLESIGNING. We believe that this characterization might be of independent interest and hence describe it here.

The *support graph* of a real symmetric $n \times n$ matrix M is an undirected graph G where the vertex set of G is $[n] := \{1, \ldots, n\}$, and the edge set of G is $\big\{\{u,v\} \mid M[u,v] \neq 0\big\}$. We note that G could have self-loops depending on the diagonal entries of M. A *perfect 2-matching* in a graph G with edge set E is an assignment $x : E \to \{0,1,2\}$ of values to the edges such that $\sum_{e \in \delta(v)} x_e = 2$ holds for every vertex v in G (where $\delta(v)$ denotes the set of edges incident to v). Equivalently, a perfect 2-matching in a graph G is a vertex-disjoint union of edges and cycles (cycles could be loop edges) in G such that each vertex is incident to at least one edge. We show the following characterization:

▶ **Theorem 5.** *Let M be a symmetric $n \times n$ matrix and let G be the support graph of M. The following are equivalent:*

(i) *There exists a symmetric signing s such that the signed matrix $M(s)$ is invertible.*

(ii) *The support graph G contains a perfect 2-matching.*

▶ Remark 1. The structural characterization of Theorem 5 leads to a polynomial-time algorithm to solve INVERTIBLESIGNING – it suffices to verify if the support graph of the input matrix contains a perfect 2-matching which can be done in polynomial-time.

▶ Remark 2. We present a constructive proof of Theorem 5 via a generalization (see Theorem 8 in Section 2). Our proof of Theorem 5 is constructive but we are aware of a non-constructive proof using Combinatorial Nullstellensatz. This alternative non-constructive proof is available online in an earlier arXiv version of this [4].

1.3 Related Work

Skew Symmetric Matrix of Indeterminates. A square skew-symmetric matrix of indeterminates with zeroes on the diagonal is known as the *Tutte matrix* of its support graph. A well-known result by Tutte shows that the determinant polynomial of the Tutte matrix is non-zero if and only if the corresponding support graph has a perfect matching. Our result in Theorem 5 can be interpreted as a variant of Tutte's result to square *symmetric* matrices of indeterminates with zeros on the diagonal.

Cunningham and Geelen [6] extended Tutte's work along a different direction by giving a characterization of invertible submatrices of the Tutte matrix using *path-matchings*. Given a graph G with vertex set V and vertex subsets $R, L \subseteq V$, a (R, L)-*path-matching* in G is a collection of vertex-disjoint paths from R to L and edges in $G[V \setminus (R \cup L)]$. A *perfect* (R, L)-path-matching is a (R, L)-path-matching in which every vertex in G is incident to some edges of the vertex-disjoint paths. They showed that the determinant polynomial of a square submatrix of the Tutte matrix of G with column set R and row set L is non-zero if and only if there exists a perfect (R, L)-path-matching in G. The notion of *cycle-covers* that we introduce in Section 2 and our result in Theorem 8 can be interpreted as variants of Cunningham and Geelen's result to square *symmetric* matrices of indeterminates with zeros on the diagonal.

Our results in Theorems 5 and 8 go further than Cunningham and Geelen's result by not only giving similar characterizations for the determinant to be a non-zero polynomial but also by giving polynomial-time algorithms to find a point in $\{\pm 1\}^E$ at which the polynomial is non-zero.

Minimum Rank Problems. A line of work seemingly related to ours is the minimum rank problem (e.g., see [8, 7]): given a graph G, the goal is to compute the minimum rank of the weighted adjacency matrix of a graph obtained by giving non-zero real-valued weights to the edges of G. We emphasize that the allowed weights in the minimum rank problem are arbitrary and are not simply signs of the given adjacency matrix as in the case of our work. A signed variant of the minimum rank problem has also been addressed in the literature: given a sign pattern matrix S, the goal is to compute the minimum rank of a matrix with real-valued entries whose sign pattern is identical to S. Once again, we emphasize the distinction between the signed variant of the minimum rank problem and the problems studied in our work: in the signed variant of the minimum rank problem, the sign pattern is the input and the goal is to find a matrix with real-valued entries matching the input sign pattern and achieving minimum rank. In contrast, the problems studied in our work have real-valued entries as inputs and the goal is to find a symmetric sign pattern of the entries to achieve the specified spectral properties.

A year after posting our work on arXiv [4], Akbari, Ghafari, Kazemian, and Nahvi [2] also posted an article addressing InvertibleSigning[3]. They show the same characterization as Theorem 5 with a proof identical to the non-constructive proof appearing in the early arXiv version of our work [4]. We emphasize that in addition to showing the structural characterization in Theorem 5, this work resolves the search problem in Theorem 2, and

[3] Our arXiv post dated Nov, 2016: `https://arxiv.org/abs/1611.03624`; the post by Akbari, Ghafari, Kazemian, and Nahvi dated Aug, 2017: `https://arxiv.org/abs/1708.07118`.

moreover shows a much more general structural characterization in Theorem 8 with a constructive proof.

1.4 Organization

In Section 1.5, we review definitions and notations. In Section 2, we describe an efficient algorithm to find an invertible signing (Theorem 2). Due to space constraints, we refer the reader to the full version of the paper for all missing proofs [4].

1.5 Preliminaries

Unless otherwise specified, all matrices are symmetric and take values over the reals. Since all of our results are for symmetric signings, we will just use the term *signing* to refer to a symmetric signing in the rest of this work. We denote the entry-wise product of two $n \times n$ matrices M and s as $M(s)$ (even when s is not necessarily a signing).

Let S_n be the set of permutations of n elements, M be a real symmetric $n \times n$ matrix, and s be a symmetric $n \times n$ signing. Then, the *permutation expansion* of the determinant of a signed matrix $M(s)$ is given by

$$\det M(s) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \cdot \prod_{i=1}^{n} M(s)[i, \sigma(i)].$$

A permutation σ in S_n has a unique cycle decomposition and hence corresponds to a vertex-disjoint union of directed cycles on n vertices. Removing the orientation gives an undirected graph which is a vertex disjoint union of cycles, self-loops, and matching edges.

2 Finding Invertible Signings

In this section, we focus on invertible signings and the proof of Theorem 2. We prove a much more general statement in comparison to the one given in Theorem 5, which we believe could be of independent interest. We start by introducing the background needed to state the general version.

Symmetric signings of asymmetric sub-matrices. Let M be a symmetric $n \times n$ matrix. For $X, Y \subseteq [n]$ being a subset of row and column indices of the same cardinality, let $M[X, Y]$ denote the submatrix of M obtained by picking the rows in X and the columns in Y. We note that $M[X, Y]$ is a square matrix, but it may *not* be symmetric even though M is symmetric. We are interested in finding a symmetric $n \times n$ signing s so that the square submatrix $M(s)[X, Y]$ is invertible. We emphasize that for a symmetric signing s, the (possibly asymmetric) matrix $M(s)[X, Y]$ is symmetric on $X \cap Y$, that is, the $[i, j]$'th and the $[j, i]$'th entries of the matrix $M(s)[X, Y]$ are the same for every $i, j \in X \cap Y$.

Perfect 2-matchings in subgraphs. Let G be a simple undirected graph, possibly containing self-loops. Let X, Y be vertex subsets of G. We consider the subgraph $G[X \cup Y]$ induced by $X \cup Y$. An (X, Y)-*cycle-cover* is a collection of edges of the subgraph $G[X \cup Y]$ that induce a vertex-disjoint union of paths and cycles (cycles could be loop edges) in $G[X \cup Y]$ such that (1) every cycle is a subgraph of $G[X \cap Y]$, (2) every vertex of $X \cup Y$ is incident to at least one edge, and (3) every path either has one end-vertex in $X \setminus Y$, the other end-vertex in $Y \setminus X$, and all intermediate vertices in $X \cap Y$, or has both end-vertices in $X \cap Y$ with only one edge (see Figure 1 for an example). We note that (X,X)-cycle-covers correspond to perfect 2-matchings in $G[X]$ and hence, (V,V)-cycle-covers correspond to perfect 2-matchings in G. It follows that in (X,X)-cycle-covers all paths are only a single edge in G. Furthermore, the existence of an (X,Y)-cycle-cover is possible only if $|X|=|Y|$. The following lemma states that the existence of an (X,Y)-cycle-cover in a given graph can be verified efficiently.

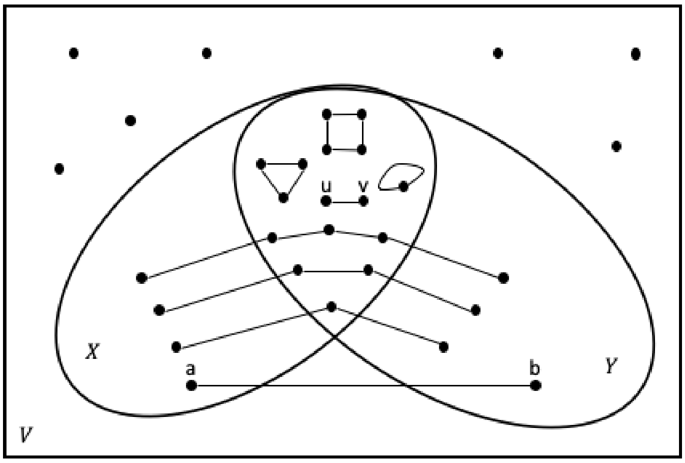


Figure 1 An (X,Y)-cycle-cover F. Furthermore, by our definitions below, the edge $\{a,b\}$ is in Paths(F) while the edge $\{u,v\}$ is in Matchings(F).

▶ **Lemma 6.** *There exists a polynomial-time algorithm that decides if there is an (X,Y)-cycle-cover in a given graph G for given vertex subsets X,Y of G.*

The key observation to prove Lemma 6 is that finding an (X,Y)-cycle-cover can be reduced to finding a perfect matching in an auxiliary bipartite graph.

Let $M \in \mathbb{R}^{n\times n}$ be a symmetric matrix, $X,Y \subseteq [n]$ with $|X|=|Y|$ and s be a symmetric $n\times n$ matrix. Recall that we are interested in finding a symmetric $n\times n$ signing s so that the square submatrix $M(s)[X,Y]$ is invertible. We derive a convenient expression for $\det(M(s)[X,Y])$ that is based on (X,Y)-cycle-covers. For an (X,Y)-cycle-cover F, let Cycles(F), Paths(F), and Matchings(F) denote the set of cycles in F, paths in F with end-vertices in $X\setminus Y$ and $Y\setminus X$, and paths in F that are contained in $G[X\cap Y]$, respectively. Moreover, let Loops(F) and NTCs(F) denote the set of self-loops and non-trivial-cycles in F. We emphasize that Cycles(F) = Loops(F) ∪ NTCs(F). We also note that Cycles(F), Paths(F), and Matchings(F) are all vertex-disjoint from one another and if $X=Y$ then Paths$(F)=\emptyset$. We define

$$\begin{aligned} M(s)_{\mathrm{Cycles}(F)} &:= \prod_{C\in\mathrm{Cycles}(F)}\ \prod_{\{u,v\}\in C} M(s)[u,v], \\ M(s)_{\mathrm{Paths}(F)} &:= \prod_{P\in\mathrm{Paths}(F)}\ \prod_{\{u,v\}\in P} M(s)[u,v], \text{ and} \\ M(s)_{\mathrm{Matchings}(F)} &:= \prod_{\{u,v\}\in\mathrm{Matchings}(F)} M(s)[u,v]^2. \end{aligned}$$

With this notation, we have the following claim that the determinant of $M(s)[X,Y]$ is a $\{\pm1\}$-linear combination of terms corresponding to (X,Y)-cycle-covers in G.

▶ **Lemma 7** ($(\boldsymbol{X},\boldsymbol{Y})$-cycle-cover expansion)**.** *Let $M \in \mathbb{R}^{n\times n}$ be a symmetric $n \times n$ matrix, $X, Y \subseteq [n]$ with $|X| = |Y|$, and s be a symmetric $n \times n$ matrix. Let G be the support graph of M and $\mathcal{F}$ be the set of all (X,Y)-cycle-covers in G. Then, there exists $\lambda_F \in \{\pm 1\}$ for all $F \in \mathcal{F}$ such that*

$$\det(M(s)[X,Y]) = \sum_{F\in\mathcal{F}} \lambda_F \cdot 2^{|NTCs(F)|} \cdot M(s)_{Cycles(F)} \cdot M(s)_{Paths(F)} \cdot M(s)_{Matchings(F)}.$$

Moreover, if there are $F_1, F_2 \in \mathcal{F}$ such that $Cycles(F_1) = Cycles(F_2)$ and $Paths(F_1) = Paths(F_2)$ then $\lambda_{F_1} = \lambda_{F_2}$.

Proof. For simplicity, we denote $M' = M[X,Y]$. Let $k := |X|$ and let S_k denote the set of permutations on k elements. Then, by the permutation expansion of the determinant, we have

$$\det(M'(s)) = \sum_{\sigma\in S_k} \mathrm{sgn}(\sigma) \cdot \prod_{i=1}^{k} s[i,\sigma(i)] \cdot M'[i,\sigma(i)].$$

We recall that $\mathrm{sgn}(\sigma) \in \{\pm 1\}$. Moreover, if $\sigma_1, \sigma_2 \in S_k$ such that σ_1 and σ_2 have the same cycle structure then $\mathrm{sgn}(\sigma_1) = \mathrm{sgn}(\sigma_2)$. Now, we note that there is a one-to-one correspondence between S_k and bijections from X to Y. So, we may view $\sigma \in S_k$ as a bijection $\sigma' : X \to Y$. Now, consider the graph $H_{\sigma'}$ on vertex set $X \cup Y$ and edge set $F_{\sigma'} := \{\{u,v\} \mid \sigma'(u) = v\}$. Since σ' is a bijection, it follows that $F_{\sigma'}$ is an (X,Y)-cycle-cover in the complete graph on vertex set $X \cup Y$. Moreover, since each non-trivial-cycle in an (X,Y)-cycle-cover can take one of two orientations in any corresponding permutation, there are $2^{|\mathrm{NTCs}(F)|}$ distinct permutations which map to each (X,Y)-cycle-cover F. Hence,

$$\begin{aligned}\prod_{i=1}^{n} s[i,\sigma(i)] \cdot M'[i,\sigma(i)] &= \prod_{u\in X} s[u,\sigma(u)] \cdot M[u,\sigma'(u)] \\ &= M(s)_{\mathrm{Cycles}(F_{\sigma'})} \cdot M(s)_{\mathrm{Paths}(F_{\sigma'})} \cdot M(s)_{\mathrm{Matchings}(F_{\sigma'})}.\end{aligned}$$

The above-term is non-zero only if $F_{\sigma'}$ is an (X,Y)-cycle-cover in the support graph of G. Furthermore, if $F_1, F_2 \in \mathcal{F}$ such that $\mathrm{Cycles}(F_1) = \mathrm{Cycles}(F_2)$ and $\mathrm{Paths}(F_1) = \mathrm{Paths}(F_2)$ then $\lambda_{F_1} = \lambda_{F_2}$ since the corresponding permutations would have the same cycle structure. ◀

To prove Theorems 5 and 2, we show the following theorem which gives a generalized structural characterization: it characterizes the existence of invertible symmetric signings for (potentially asymmetric) submatrices of symmetric matrices.

▶ **Theorem 8.** *Let M be a real symmetric $n \times n$ matrix with support graph G and $X, Y \subseteq [n]$ with $|X| = |Y|$. The following are equivalent:*

(i) *There exists an (X,Y)-cycle-cover in G.*

(ii) *There exists a symmetric signing s such that $M(s)[X,Y]$ is invertible.*

Moreover, there exists a polynomial-time algorithm that takes a real symmetric $n \times n$ matrix M and $X, Y \subseteq [n]$ as input and verifies if there exists a symmetric signing s such that $M(s)[X,Y]$ is invertible and if so, find such a signing.

Notation. Let M be a real symmetric $n \times n$ matrix with support graph G. Let A and B be vertex subsets of G. We define $E[A,B]$ to be the set of edges with one end-vertex in A and the other end-vertex in B. We use $E[A]$ to denote $E[A,A]$. Let e be an edge in G corresponding to the non-zero entry $M[u,v]$ $(= M[v,u])$. We define $M^{\overline{e}}$ as the matrix obtained by setting $M[u, v]$ and $M[v, u]$ to 0. For a signing s and row and column indices $u, v \in [n]$, we can obtain another signing s' such that $s'[u, v] := -s[u, v]$, $s'[v, u] := -s[v, u]$ and $s'[i, j] := s[i, j]$ for every entry $(i, j) \in [n] \times [n] \setminus \{(u, v), (v, u)\}$. We call this operation as s' *obtained from* s *by flipping on* $\{u, v\}$.

Proof of Theorem 8. We first present a constructive proof of the characterization. We will then use the proof to design the algorithm.

Lemma 7 immediately shows that (ii) implies (i): If we have a symmetric signing s such that $M(s)[X, Y]$ is invertible, then at least one of the terms in the (X, Y)-cycle-cover expansion of $\det(M(s)[X, Y])$ is non-zero. Hence, there exists an (X, Y)-cycle-cover in G.

We show that (i) implies (ii). Suppose not. Among the counterexamples, consider the ones with $|X|$ minimum and among these, pick a matrix M with minimum number of non-zero entries. Without loss of generality, let M be an $n \times n$ matrix with support graph G and let $X, Y \subseteq [n]$ with $|X| = |Y|$. Since we chose a counterexample, we have that

(A) there exists an (X, Y)-cycle-cover in G, but

(B) there is no symmetric signing s such that $M(s)[X, Y]$ is invertible.

We will arrive at a contradiction by showing that a signing s satisfying (ii) exists. We begin with the following claim about the counterexample.

▷ Claim 9. $E[X \setminus Y, Y] = \emptyset$ and $E[Y \setminus X, X] = \emptyset$.

Proof. Suppose there exists an edge $e \in E[X \setminus Y, Y]$. Let $e := \{u, v\}$ with $u \in X \setminus Y$ and $v \in Y$. Then there exists $\alpha \in \{\pm 1\}$ such that the determinant of $M(s)[X, Y]$ can be expressed as a linear function of $s[u, v]$:

$$\det(M(s)[X, Y]) = \alpha \cdot s[u, v] \cdot M[u, v] \cdot \det(M(s)[X - u, Y - v]) + \det(M^{\overline{e}}(s)[X, Y]). \quad (1)$$

Case 1: Suppose there exists an (X, Y)-cycle-cover F containing e. We observe that $F - e$ is an $(X - u, Y - v)$-cycle-cover in G. Since we have a smallest counterexample, it follows that there exists a symmetric signing s such that $\det(M(s)[X - u, Y - v]) \neq 0$. Since $\det(M(s)[X, Y])$ is a linear function of $s[u, v]$, it follows that $\det(M(s)[X, Y]) \neq 0$ or $\det(M(s')[X, Y]) \neq 0$ where s' is a signing obtained from s by flipping on $\{u, v\}$. Hence, we have a contradiction to assumption B about the counterexample.

Case 2: Suppose that every (X, Y)-cycle-cover in G does not contain e. Then there is no $(X - u, Y - v)$-cycle-cover in G. Since (ii) implies (i), it follows that $\det(M(s)[X - u, Y - v]) = 0$ for every symmetric signing s. Let F be an (X, Y)-cycle-cover in G (as promised to exist by A). Then F is an (X, Y)-cycle-cover in $G - e$. Since we have a smallest counterexample, it follows that there exists a symmetric signing s such that $\det(M^{\overline{e}}(s)[X, Y]) \neq 0$. By (1), we observe that $\det(M(s)[X, Y]) \neq 0$. Thus, the symmetric signing s is a contradiction to assumption B about the counterexample.

Hence, $E[X \setminus Y, Y] = \emptyset$. Similarly $E[Y \setminus X, X] = \emptyset$. ◁

Now, if $X \setminus Y \neq \emptyset$ and there is no edge $e \in E[X \setminus Y, Y]$, then there is no (X, Y)-cycle-cover in G, a contradiction to assumption A about the counterexample. Hence, $X \setminus Y = \emptyset$. Similarly, $Y \setminus X = \emptyset$. Thus, we have $X = Y$ in the counterexample. We next show that the counterexample cannot have any self-loop edges.

▷ Claim 10. There are no self-loop edges in $E[X]$.

Proof. Suppose there exists a self-loop edge in $E[X]$. Let $e = \{u, u\}$ for some $u \in X$. Then, we again have that $\det(M(s)[X, Y])$ is a linear function of $s[u, u]$:

$$\det(M(s)[X, X]) = s[u, u] \cdot M[u, u] \cdot \det(M(s)[X - u, X - u]) + \det(M^{\overline{e}}(s)[X, X]). \quad (2)$$

We arrive at a contradiction by proceeding similar to the proof of the previous claim. We avoid restating the proof in the interests of brevity. ◁

By Claim 10, the counterexample has no self-loop edges in $E[X]$. Our next claim strengthens this further by showing that the counterexample has no (X, Y)-cycle-cover with cycle edges.

▷ Claim 11. Every (X, X)-cycle-cover in G has no cycles.

Proof. Suppose there exists an (X, X)-cycle-cover F in G with a cycle C induced by F. Let $e = \{u, v\}$ be an edge in the cycle. By Claim 10, we know that $u \neq v$. We observe that $\det(M(s)[X, X])$ is a quadratic function of $s[u, v]$, i.e., there exists $\alpha \in \{\pm 1\}$ such that the determinant of $M(s)[X, X]$ can be expressed as

$$\begin{aligned}\det(M(s)[X, X]) = &-s[u, v]^2 \cdot M[u, v]^2 \cdot \det(M(s)[X - \{u, v\}, X - \{u, v\}]) \\ &+ 2\alpha \cdot s[u, v] \cdot M[u, v] \cdot \det(M^{\overline{e}}(s)[X - u, X - v]) \\ &+ \det(M^{\overline{e}}(s)[X, X]).\end{aligned} \quad (3)$$

Furthermore, $F - e$ is an $(X - u, X - v)$-cycle-cover in G. Since we have a smallest counterexample, it follows that there exists a symmetric signing s such that $\det(M^{\overline{e}}(s)[X - u, X - v]) \neq 0$. We now define the quadratic function

$$\begin{aligned}f(x) := &-x^2 \cdot M[u, v]^2 \cdot \det(M(s)[X - \{u, v\}, X - \{u, v\}]) \\ &+ 2\alpha x \cdot M[u, v] \cdot \det(M^{\overline{e}}(s)[X - u, X - v]) + \det(M^{\overline{e}}(s)[X, X]),\end{aligned}$$

and consider the roots of the quadratic equation $f(x) = 0$. Since $\det(M^{\overline{e}}(s)[X-u, X-v]) \neq 0$, the sum of the roots of this quadratic equation is non-zero. Since the real roots of a quadratic function are symmetric about the extreme point of the parabola defined by the function (i.e., symmetric about $\arg\min f(x)$), there exists $x \in \{\pm 1\}$ that is *not* a root of $f(x)$. Hence, either $\det(M(s)[X, Y]) \neq 0$ or $\det(M(s')[X, Y]) \neq 0$ where s' is a signing obtained from s by flipping on $\{u, v\}$. Thus, either s or s' contradict assumption B about the counterexample. ◁

By Claims 9 and 10, the counterexample has $X = Y$ with no loop edges in $E[X]$. Furthermore, by Claim 11, every (X, X)-cycle-cover in G has no cycles. By definition of (X, X)-cycle-covers, it follows that each (X, X)-cycle-cover in G corresponds to a perfect matching in $G[X]$. Let N be an (X, X)-cycle-cover in G.

▷ Claim 12. N is the unique (X, X)-cycle-cover in G.

Proof. Let e be an arbitrary edge in N. Suppose there exists an (X, X)-cycle-cover N' in $G - e$. Then, Claims 10 and 11 imply that N' is also a perfect matching in $G[X]$. We consider $N'' := N \cup N'$. Since N and N' are perfect matchings in $G[X]$, the set of edges N'' induces a vertex-disjoint union of edges and cycles of even length in $G[X]$. Hence, N'' is an (X, X)-cycle-cover in G. Furthermore, since $e \in N \setminus N'$, it follows that N'' contains at least one cycle. This contradicts Claim 11. Thus, every edge $e \in N$ belongs to every (X, X)-cycle-cover in G. Consequently, N is the unique (X, X)-cycle-cover in G. ◁

Since N is the unique (X, X)-cycle-cover in G, by Lemma 7, we have that

$$\det(M(s)[X, X]) = (-1)^{|N|} \prod_{\{u,v\} \in N} M(s)[u, v]^2$$

which is non-zero for every signing s. Thus, there exists a symmetric signing s such that $\det(M(s)[X, X]) \neq 0$, a contradiction to assumption B about the counterexample. This completes the proof of the characterization. We note that the above proof of the characterization is constructive and immediately leads to the algorithm FindSigning(M, X, Y) in Algorithm 1.

Algorithm 1 The algorithm FindSigning(M, X, Y).

```
FindSigning($M, X, Y$):
Input: $M \in \mathbb{R}^{n \times n}$ with support graph $G$, and $X, Y \subseteq [n]$ satisfying $|X| = |Y|$.
Output: A symmetric signing $s \in \{\pm 1\}^{n \times n}$ such that $M(s)[X, Y]$ is invertible if
        such a signing exists.
  1. If $|X| = |Y| \leq 1$, then brute-force search for a symmetric signing $s$ for
     which $\det(M(s))[X, Y] \neq 0$:
       1.1. If such a signing exists, return $s$.
       1.2. Else return “No Invertible Signing”.
  2. If there exists no $(X, Y)$-cycle-cover then return “No Invertible Signing”.
  3. If $E[X \setminus Y, Y] \cup E[Y \setminus X, X] \neq \emptyset$:
       3.1. Pick $e := \{u, v\} \in E[X \setminus Y, Y]$ such that $u \in X \setminus Y$ and $v \in Y$.
       3.2. If there is no $(X - u, Y - v)$-cycle-cover in $G$:
              3.2.1 $s \leftarrow$ FindSigning($M^{\overline{e}}, X, Y$).
       3.3. Else: (when there is an $(X - u, Y - v)$-cycle-cover in $G$)
              3.3.1 $s \leftarrow$ FindSigning($M, X - u, Y - v$).
       3.4. If $M(s)[X, Y]$ is invertible then return $s$.
       3.5. Else return $s'$ obtained from $s$ by flipping on $\{u, v\}$.
  4. Else: (when sets $X$ and $Y$ are identical)
       4.1. If there exists an $(X, Y)$-cycle-cover in $G$ with a cycle edge $\{u, v\}$:
              4.1.1. $s \leftarrow$ FindSigning($M, X - u, Y - v$).
              4.1.2. If $M(s)[X, Y]$ is invertible then return $s$.
              4.1.3. Else return $s'$ obtained from $s$ by flipping on $\{u, v\}$.
       4.2. Else: (when all $(X, Y)$-cycle-covers are perfect matchings in $G[X]$)
              4.2.1 Return $\mathbf{1}$ (the all-positive signing).
```

We now describe an efficient implementation of the non-trivial steps in FindSigning. In Step 1, the algorithm performs a brute-force search. We note that the search needs to be conducted only for the entries $s[u, v]$ where $u, v \in X \cup Y$ since $\det(M(s)[X, Y])$ is independent of the remaining entries of the signing s. Since $|X \cup Y| \leq 2$, the search can be conducted in constant time by picking an arbitrary sign for the remaining entries.

Lemma 6 implies that Steps 2 and 3.2 can be implemented to run in polynomial time. We recall that any cycle edge in an (X, X)-cycle-cover must be a cycle edge in some perfect 2-matching in $G[X]$. Claim 13 shows that Step 4.1 can be implemented to run in polynomial time. Finally, the recursive algorithm terminates in polynomial time since each recursive call reduces either $|X \cup Y|$ or the number of non-zero entries in M. ◀

▷ Claim 13. There is a polynomial-time algorithm that given a graph, finds an edge that belongs to a cycle in some perfect 2-matching of the graph or decides that no such edge exists.

Algorithm 2 The algorithm FindCycleEdge(G).

```
FindCycleEdge($G$):
Input: A graph $G$ with vertex set $V$.
Output: An edge $e$ that is a cycle edge in some perfect 2-matching in $G$ if one exists.
  1. If there exists no perfect 2-matching in $G$ then return "No edge".
  2. Let $F$ be a perfect 2-matching in $G$.
  3. If $F$ contains a cycle $C$ then return any edge in $C$.
  4. For $e \in F$:
       4.1. Let $N_e$ be a perfect 2-matching in $G - e$ if one exists.
       4.2. If $N_e$ exists and has a cycle $C$ then return any edge in $C$.
  5. If Step 4 finds $N_e$ for some $e \in F$, then return $e$.
  6. Else return "No edge".
```

Proof. To prove the claim we consider the algorithm FindCycleEdge(G) in Algorithm 2. If at any point we find a perfect 2-matching with a cycle then we return an edge from it. Hence, it only remains to show the correctness of Steps 5 and 6. Let F be a perfect 2-matching with no cycle edge. Suppose there exists a perfect 2-matching N_e for some edge e with no cycle edge. Then N_e and F are both perfect matchings in G. It follows that $N_e \cup F$ will be a perfect 2-matching where e is in a cycle and hence Step 5 is correct to return e. Now suppose that for all e there is no perfect 2-matching N_e. It follows that G has one unique perfect 2-matching F that is a perfect matching and hence Step 6 correctly returns that no cycle edge exists.

Using the algorithm from Lemma 6 we can perform Steps 1, 2, and 4.1 in polynomial time. Thus, FindCycleEdge(G) runs in polynomial time. ◁

References

1 R. Abelson and M. Rosenberg. Symbolic psycho-logic: A model of attitudinal cognition. *Behavioral Science*, 3:1–13, 1958.

2 S. Akbari, A. Ghafari, K. Kazemian, and M. Nahvi. Some Criteria for a Signed Graph to Have Full Rank. ArXiv. arXiv:1708.07118.

3 J. Akiyama, D. Avis, V. Chvátal, and H. Era. Balancing signed graphs. *Discrete Appl. Math.*, 3:227–233, 1981.

4 C. Carlson, K. Chandrasekaran, H. Chang, and A. Kolla. Invertibility and Largest Eigenvalue of Symmetric Matrix Signings. *arXiv e-prints*, November 2016. arXiv:1611.03624.

5 M. Cohen. Ramanujan Graphs in Polynomial Time. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, pages 276–281, 2016.

6 W. Cunningham and J. Geelen. The Optimal Path-Matching Problem. *Combinatorica*, 17(3):315–337, 1997. doi:10.1007/BF01215915.

7 S. Fallat and L. Hogben. Variants on the minimum rank problem: A survey II. ArXiv. arXiv:1102.5142.

8 S. Fallat and L. Hogben. The minimum rank of symmetric matrices described by a graph: A survey. *Linear Algebra and its Applications*, 426(2):558–582, 2007.

9 J. Friedman. Relative expanders or weakly relatively Ramanujan graphs. *Duke Math. J.*, 118(1):19–35, 2003.

10 P. Hansen. Labelling algorithms for balance in signed graphs. *Problémes Combinatoires et Théorie des Graphes*, pages 215–217, 1978.

11 F. Harary. On the notion of balance of a signed graph. *Michigan Math. J.*, 2(2):143–146, 1953.

12 F. Harary. On the measurement of structural balance. *Behavioral Science*, 4(4):316–323, 1959.

13 F. Harary and J. Kabell. A simple algorithm to detect balance in signed graphs. *Mathematical Social Sciences*, 1(1):131–136, 1980.

14 F. Heider. Attitudes and Cognitive Organization. *J. Psych.*, 21:107–112, 1946.

15 F. Hüffner, N. Betzler, and R. Niedermeier. Optimal edge deletions for signed graph balancing. In *Proceedings of the 6th international conference on Experimental algorithms*, pages 297–310, 2007.

16 A. Marcus, D. Spielman, and N. Srivastava. Interlacing Families I: Ramanujan Graphs of All Degrees. *Annals of Mathematics*, 182(1):307–325, 2015.

17 K. Osamu and S. Iwai. Studies on the balancing, the minimal balancing, and the minimum balancing processes for social groups with planar and nonplanar graph structures. *J. Math. Psychology*, 18(2):140–176, 1978.

18 V. Sivaraman. *Some topics concerning graphs, signed graphs and matroids.* Ph.D. dissertation, Ohio State University, 2012.

19 T. Zaslavsky. The geometry of root systems and signed graphs. *Amer. Math. Monthly*, 2:88–105, 1981.

20 T. Zaslavsky. Signed graphs. *Discrete Appl. Math.*, 1:47–74, 1982.

Efficient Analysis of Unambiguous Automata Using Matrix Semigroup Techniques

Stefan Kiefer
University of Oxford, UK

Cas Widdershoven
University of Oxford, UK

Abstract

We introduce a novel technique to analyse unambiguous Büchi automata quantitatively, and apply this to the model checking problem. It is based on linear-algebra arguments that originate from the analysis of matrix semigroups with constant spectral radius. This method can replace a combinatorial procedure that dominates the computational complexity of the existing procedure by Baier et al. We analyse the complexity in detail, showing that, in terms of the set Q of states of the automaton, the new algorithm runs in time $O(|Q|^4)$, improving on an efficient implementation of the combinatorial algorithm by a factor of $|Q|$.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Design and analysis of algorithms

Keywords and phrases Algorithms, Automata, Markov Chains, Matrix Semigroups

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.82

Related Version A full version of the paper is available at `https://arxiv.org/abs/1906.10093`.

Funding *Stefan Kiefer*: Work supported by a Royal Society University Research Fellowship.

1 Introduction

Given a finite automaton $\mathcal{A}$, what is the proportion of words accepted by it? This question is natural but imprecise: there are infinitely many words and the proportion of accepted words may depend on the word length. One may consider the sequence $d_0, d_1, \ldots$ where d_i is the proportion of length-i words accepted by $\mathcal{A}$, i.e., $d_i = \frac{|L(\mathcal{A}) \cap \Sigma^i|}{|\Sigma|^i}$. The sequence does not necessarily converge, but one may study, e.g., possible limits and accumulation points [5].

Alternatively, one can specify a probability distribution on words, e.g., with a Markov chain, and ask for the probability that a word is accepted by $\mathcal{A}$. For instance, if $\Sigma = \{a, b\}$, one may generate a random word, letter by letter, by outputting a, b with probability $1/3$ each, and ending the word with probability $1/3$. For an NFA $\mathcal{A}$, determining whether the probability of generating an accepted word is 1 is equivalent to universality (is $L(\mathcal{A}) = \Sigma^*$?), a PSPACE-complete problem. However, if $\mathcal{A}$ is unambiguous, i.e., every accepted word has exactly one accepting path, then one can compute the probability of generating an accepted word in polynomial time by solving a linear system of equations. Unambiguousness allows us to express the probability of a union as the sum of probabilities:

▶ **Example 1.** Consider the unambiguous automaton $\mathcal{A}$ in Figure 1 (left). If we generate a random word over $\{a, b\}$ according to the process described above, we have the following linear system for the vector $\vec{z}$ where $\vec{z}_q$ is, for each $q \in \{q_0, q_1, q_2, q_3\}$, the probability that the word is accepted when q is taken as initial state:

$$\begin{aligned} \vec{z}_{q_0} &= \tfrac{1}{3}\vec{z}_{q_1} + \tfrac{1}{3} & \qquad \vec{z}_{q_1} &= \tfrac{1}{3}\vec{z}_{q_0} + \tfrac{1}{3}(\vec{z}_{q_1} + \vec{z}_{q_3}) \\ \vec{z}_{q_2} &= \tfrac{1}{3}\vec{z}_{q_3} + \tfrac{1}{3}(\vec{z}_{q_0} + \vec{z}_{q_2}) & \qquad \vec{z}_{q_3} &= \tfrac{1}{3}\vec{z}_{q_2} \end{aligned}$$

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 82; pp. 82:1–82:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

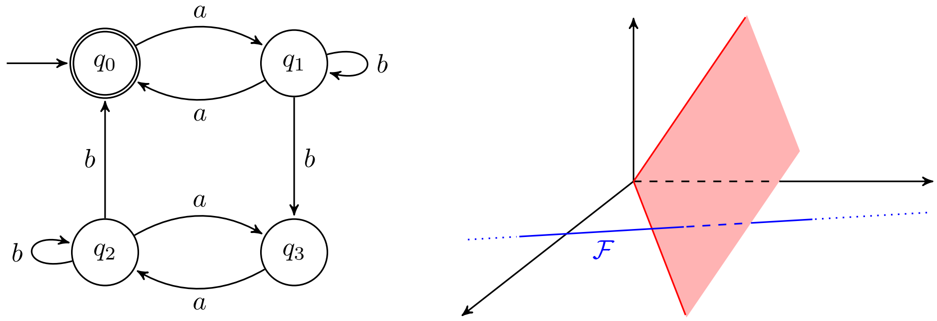


Figure 1 Left: unambiguous automaton $\mathcal{A}$. Right: visualisation of the affine space $\mathcal{F}$ (blue) and the vector space spanned by (pseudo-)cuts (red); these spaces are orthogonal.

The constant term in the equation for $\vec{z}_{q_0}$ reflects the fact that q_0 is accepting. The (other) coefficients $\frac{1}{3}$ correspond to the production of either a or b. The linear system has a unique solution.

One may view an NFA $\mathcal{A}$ as a Büchi automaton, so that its language $L(\mathcal{A}) \subseteq \Sigma^\omega$ is the set of those infinite words that have an accepting run in $\mathcal{A}$, i.e., a run that visits accepting states infinitely often. There is a natural notion of an infinite random word over Σ: in each step sample a letter from Σ uniformly at random, e.g., if $\Sigma = \{a, b\}$ then choose a and b with probability $1/2$ each. Perhaps more significantly, model checking Markov chains against Büchi automata, i.e., computing the probability that the random word generated by the Markov chain is accepted by the automaton, is a key problem in the verification of probabilistic systems. Unfortunately, like the aforementioned problem on finite words, it is also PSPACE-complete [8]. However, if the Büchi automaton is unambiguous, i.e., every accepted (infinite) word has exactly one accepting path, then one can compute the probability of generating an accepted word in polynomial time [2], both in the given Büchi automaton and in a given (discrete-time, finite-state) Markov chain. Since LTL specifications can be converted to unambiguous Büchi automata with a single-exponential blow-up, this leads to an LTL model-checking algorithm with single-exponential runtime, which is optimal. The polynomial-time algorithm from [2] for unambiguous Büchi automata is more involved than in the finite-word case.

▶ **Example 2.** In the following we view the automaton $\mathcal{A}$ from Figure 1 as an (unambiguous) Büchi automaton. If we generate a random word over $\{a, b\}$ according to the process described above, then the vector $\vec{z}$ where for each $q \in \{q_0, q_1, q_2, q_3\}$, $\vec{z}_q$ is the probability that the word is accepted when q is taken as initial state, is a solution to the following linear system:

$$\begin{aligned} \vec{z}_{q_0} &= \tfrac{1}{2}\vec{z}_{q_1} & \qquad \vec{z}_{q_1} &= \tfrac{1}{2}\vec{z}_{q_0} + \tfrac{1}{2}(\vec{z}_{q_1} + \vec{z}_{q_3}) \\ \vec{z}_{q_2} &= \tfrac{1}{2}\vec{z}_{q_3} + \tfrac{1}{2}(\vec{z}_{q_0} + \vec{z}_{q_2}) & \qquad \vec{z}_{q_3} &= \tfrac{1}{2}\vec{z}_{q_2} \end{aligned}$$

However, this linear system has multiple solutions: indeed, any scalar multiple $(1, 2, 2, 1)^\top$ is a solution.

In order to make such a linear system uniquely solvable, one needs to add further equations, and finding these further equations is where the real challenge lies. Assuming that the state space Q of $\mathcal{A}$ is strongly connected and the Markov chain generates letters uniformly at random as described above, a single additional equation $\vec{\mu}^\top \vec{z} = 1$ suffices (this can be shown with Perron-Frobenius theory: the eigenspace for the dominant eigenvalue of a nonnegative irreducible matrix is one-dimensional). We call such a vector $\vec{\mu} \in \mathbb{R}^Q$ a *normaliser*. The aim of this paper is to use a novel, linear-algebra based technique to compute normalisers more efficiently.

The suggestion in [2] was to take as normaliser the characteristic vector $[c] \in \{0,1\}^Q$ of a so-called *cut* $c \subseteq Q$. To define this, let us write $\delta(q, w)$ for the set of states reachable from a state $q \in Q$ via the word $w \in \Sigma^*$. A *cut* is a set of states of the form $c = \delta(q, w)$ such that $\delta(q, wx) \neq \emptyset$ holds for all $x \in \Sigma^*$. If a cut does not exist or if $\mathcal{A}$ does not have accepting states, then we have $\vec{z} = \vec{0}$.

▶ **Example 3.** In the automaton $\mathcal{A}$ from Figure 1, we have a cut $c = \delta(q_0, aba) = \{q_0, q_2\}$. Hence its characteristic vector $\vec{\mu} = (1, 0, 1, 0)^\top$ is a normaliser, allowing us to add the equation $\vec{\mu}^\top \vec{z} = \vec{z}_{q_0} + \vec{z}_{q_2} = 1$. Now the system is uniquely solvable: $\vec{z} = \frac{1}{3}(1, 2, 2, 1)^\top$. The equation $\vec{z}_{q_0} + \vec{z}_{q_2} = 1$ is valid by an ergodicity argument: intuitively, given a finite word that leads to q_0 *and* q_2, a random infinite continuation will almost surely enable an accepting run. For instance, $\vec{z}_{q_0} = \frac{1}{3}$ is the probability that a random infinite word over $\{a, b\}$ has an odd number of as before the first b. (This holds despite the fact that the word $abbb\ldots$ is not accepted from q_0.)

In Proposition 14 we show that an efficient implementation of the algorithm from [2] for computing a cut runs in time $O(|Q|^5)$. Our goal is to find a normaliser $\vec{\mu}$ more efficiently.

The general idea is to move from a combinatorial problem, namely computing a set $c \subseteq Q$, to a continuous problem, namely computing a vector $\vec{\mu} \in \mathbb{R}^Q$. To illustrate this, note that since we can choose as $\vec{\mu}$ the characteristic vector of an arbitrary cut, we may also choose a convex combination of such vectors, leading to a normaliser $\vec{\mu}$ with entries other than 0 or 1.

The technical key ideas of this paper draw on the observation that for unambiguous automata with cuts, the transition matrices generate a semigroup of matrices whose spectral radii are all 1. (The spectral radius of a matrix is the largest absolute value of its eigenvalues.) This observation enables us to adopt techniques that have recently been devised by Protasov and Voynov [16] for the analysis of matrix semigroups with constant spectral radius. To the best of the authors' knowledge, such semigroups have not previously been connected to unambiguous automata. This transfer is the main contribution of this paper.

To sketch the gist of this technique, for any $a \in \Sigma$ write $M(a) \in \{0,1\}^{Q \times Q}$ for the transition matrix of the unambiguous automaton $\mathcal{A}$, define the average matrix $\overline{M} = \frac{1}{|\Sigma|}\sum_{a \in \Sigma} M(a)$, and let $\vec{y} = \overline{M}\vec{y} \in \mathbb{R}^Q$ be an eigenvector with eigenvalue 1 (the matrix $\overline{M}$ has such an eigenvector if $\mathcal{A}$ has a cut). Since the matrix semigroup, $\mathcal{S} \subseteq \{0,1\}^{Q \times Q}$, generated by the transition matrices $M(a)$ has constant spectral radius, it follows from [16] that one can efficiently compute an affine space $\mathcal{F} \subseteq \mathbb{R}^Q$ with $\vec{y} \in \mathcal{F}$ and $\vec{0} \notin \mathcal{F}$ such that for any $\vec{v} \in \mathcal{F}$ and any $M \in \mathcal{S}$ we have $M\vec{v} \in \mathcal{F}$. Using the fact that $\delta(q, wx)$ is a cut (for all $x \in \Sigma^*$) whenever $\delta(q, w)$ is a cut, one can show that all characteristic vectors of cuts have the same scalar product with all $\vec{v} \in \mathcal{F}$, i.e., all characteristic vectors of cuts are in the vector space orthogonal to $\mathcal{F}$. Indeed, we choose as normaliser $\vec{\mu}$ a vector that is orthogonal to $\mathcal{F}$. This linear-algebra computation can be carried out in time $O(|Q|^3)$. In the visualisation on the right of Figure 1, the characteristic vectors of cuts lie in the plane shaded in red, which is orthogonal to straight line $\mathcal{F}$ (blue).

▶ **Example 4.** In the automaton $\mathcal{A}$ from Figure 1, the vector $\vec{y} = (1, 2, 2, 1)^\top$ satisfies $\overline{M}\vec{y} = \vec{y}$ where $\overline{M} = \frac{1}{2}(M(a) + M(b))$. The affine space $\mathcal{F} := \{\vec{y} + s(1, -1, -1, 1)^\top \mid s \in \mathbb{R}\}$ has the mentioned closure properties, i.e., $M(a)\mathcal{F} \subseteq \mathcal{F}$ and $M(b)\mathcal{F} \subseteq \mathcal{F}$. Note that the vector $\vec{\mu}$ from Example 3 is indeed orthogonal to $\mathcal{F}$, i.e., $\vec{\mu}^\top(1, -1, -1, 1)^\top = 0$.

However, to ensure that $\vec{\mu}$ is a valid normaliser, we need to restrict it further. To this end, we compute, for some state $q \in Q$, the set $Co(q) \subseteq Q$ of *co-reachable* states, i.e., states $r \in Q$ such that $\delta(q, w) \supseteq \{q, r\}$ holds for some $w \in \Sigma^*$. This requires a combinatorial algorithm, which is similar to a straightforward algorithm that would verify the unambiguousness of $\mathcal{A}$. Its runtime is quadratic in the number of transitions of $\mathcal{A}$, i.e., $O(|Q|^4)$ in the worst case. Then we restrict $\vec{\mu}$ such that $\vec{\mu}_q = 1$ and μ is non-zero only in entries that correspond to $Co(q)$. In the visualisation on the right of Figure 1, restricting some components of $\vec{\mu}$ to be 0 corresponds to the vectors in the shaded (red) plane that lie on the plane described by $q' = 0$ for all $q' \in Q \setminus Co(q)$.

▶ **Example 5.** We have $Co(q_0) = \{q_0, q_2\}$. So we restrict $\vec{\mu}$ to be of the form $(1, 0, x, 0)^\top$. Together with the equation $\vec{\mu}^\top (1, -1, -1, 1)^\top = 0$ this implies $\vec{\mu} = (1, 0, 1, 0)^\top$. The point is that, although this is the same vector computed via a cut in Example 3, the linear-algebra based computation of $\vec{\mu}$ is more efficient.

In the rest of the paper we analyse the general case of model checking a given Markov chain against a given unambiguous Büchi automaton. The efficiency gain we aim for with our technique can only be with respect to the automaton, not the Markov chain; nevertheless, we analyse in detail the runtime in terms of the numbers of states and transitions in both the automaton and the Markov chain. The main results are developed in Section 3. In Section 3.1 we describe the general approach from [2, 3]. In Section 3.2 we analyse the runtime of an efficient implementation of the algorithm from [2, 3] for computing a cut. Our main contribution lies in Section 3.3, where we develop a new approach for computing a normaliser, based on the mentioned spectral properties of the transition matrices in unambiguous automata. We close in Section 4 with a discussion. The full version of this paper [12] contains an appendix with proofs.

2 Preliminaries

We assume the reader to be familiar with basic notions of finite automata over infinite words and Markov chains, see, e.g., [9, 13]. In the following we provide a brief summary of our notation and a few facts related to linear algebra.

Finite automata. A *Büchi automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ where Q is the finite set of states, $Q_0 \subseteq Q$ is the set of initial states, Σ is the finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states. We extend the transition function to $\delta : Q \times \Sigma^* \to 2^Q$ and to $\delta : 2^Q \times \Sigma^* \to 2^Q$ in the standard way. For $q \in Q$ we write $\mathcal{A}_q$ for the automaton obtained from $\mathcal{A}$ by making q the only initial state.

Given states $q, r \in Q$ and a finite word $w = a_0 a_1 \cdots a_{n-1} \in \Sigma^*$, a *run* for w from q to r is a sequence $q_0 q_1 \cdots q_n \in Q^{n+1}$ with $q_0 = q$, $q_n = r$ and $q_{i+1} \in \delta(q_i, a_i)$ for $i \in \{0, \ldots, n-1\}$. A *run* in $\mathcal{A}$ for an infinite word $w = a_0 a_1 a_2 \cdots \in \Sigma^\omega$ is an infinite sequence $\rho = q_0 q_1 \cdots \in Q^\omega$ such that $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i, a_i)$ for all $i \in \mathbb{N}$. Run ρ is called *accepting* if $\inf(\rho) \cap F \neq \emptyset$ where $\inf(\rho) \subseteq Q$ is the set of states that occur infinitely often in ρ. The *language* $\mathcal{L}(\mathcal{A})$ of accepted words consists of all infinite words $w \in \Sigma^\omega$ that have at least one accepting run. $\mathcal{A}$ is called *unambiguous* if each word $w \in \Sigma^\omega$ has at most one accepting run. We use the acronym UBA for unambiguous Büchi automaton.

We define $|\delta| := |\{(q, r) \mid \exists\, a \in \Sigma : r \in \delta(q, a)\}|$, i.e., $|\delta| \leq |Q|^2$ is the number of transitions in $\mathcal{A}$ when allowing for multiple labels per transition. In [12, Appendix A] we give an example that shows that the number of transitions can be quadratic in $|Q|$, even for UBAs with a strongly connected state space. We assume $|Q| \le |\delta|$, as states without outgoing transitions can be removed. In this paper, Σ may be a large set (of states in a Markov chain), so it is imperative to allow for multiple labels per transition. We use a lookup table to check in constant time whether $r \in \delta(q, a)$ holds for given r, q, a.

A *diamond* is given by two states $q, r \in Q$ and a finite word w such that there exist at least two distinct runs for w from q to r. One can remove diamonds (see [12, Appendix B.1]):

▶ **Lemma 6.** *Given a UBA, one can compute in time $O(|\delta|^2|\Sigma|)$ a UBA of at most the same size, with the same language and without diamonds.*

For the rest of the paper, we assume that UBAs do not have diamonds.

Vectors and matrices. We consider vectors and square matrices indexed by a finite set S. We write (column) vectors $\vec{v} \in \mathbb{R}^S$ with arrows on top, and $\vec{v}^\top$ for the transpose (a row vector) of $\vec{v}$. The zero vector and the all-ones vector are denoted by $\vec{0}$ and $\vec{1}$, respectively. For a set $T \subseteq S$ we write $[T] \in \{0,1\}^S$ for the characteristic vector of T, i.e., $[T]_s = 1$ if $s \in T$ and $[T]_s = 0$ otherwise. A matrix $M \in [0,1]^{S \times S}$ is called *stochastic* if $M\vec{1} = \vec{1}$, i.e., if every row of M sums to one. For a set $U \subseteq S$ we write $\vec{v}_U \in \mathbb{R}^U$ for the restriction of $\vec{v}$ to U. Similarly, for $T, U \subseteq S$ we write $M_{T,U}$ for the submatrix of M obtained by deleting the rows not indexed by T and the columns not indexed by U. The (directed) *graph* of a nonnegative matrix $M \in \mathbb{R}^{S \times S}$ has vertices $s \in S$ and edges (s,t) if $M_{s,t} > 0$. We may implicitly associate M with its graph and speak about graph-theoretic concepts such as reachability and strongly connected components (SCCs) in M.

Markov chains. A (finite-state discrete-time) *Markov chain* is a pair $\mathcal{M} = (S, M)$ where S is the finite set of states, and $M \in [0,1]^{S \times S}$ is a stochastic matrix that specifies transition probabilities. An *initial distribution* is a function $\iota : S \to [0,1]$ satisfying $\sum_{s \in S} \iota(s) = 1$. Such a distribution induces a probability measure $\Pr_\iota^{\mathcal{M}}$ on the measurable subsets of S^ω in the standard way, see for instance [1, chapter 10.1, page 758]. If ι is concentrated on a single state s, we may write $\Pr_s^{\mathcal{M}}$ for $\Pr_\iota^{\mathcal{M}}$. We write E for the set of edges in the graph of M. Note that $|S| \le |E| \le |S|^2$, as M is stochastic.

Solving linear systems. Let $\kappa \in [2,3]$ be such that one can multiply two $n \times n$-matrices in time $O(n^\kappa)$ (in other literature, κ is often denoted by ω). We assume that arithmetic operations cost constant time. One can choose $\kappa = 2.4$, see [14] for a recent result. One can check whether an $n \times n$ matrix is invertible in time $O(n^\kappa)$ [6]. Finally, one can solve a linear system with n equations using the Moore-Penrose pseudo-inverse [11] in time $O(n^\kappa)$ [15].

Spectral theory. The *spectral radius* of a matrix $M \in \mathbb{R}^{S \times S}$, denoted $\rho(M)$, is the largest absolute value of the eigenvalues of M. By the Perron-Frobenius theorem [4, Theorems 2.1.1, 2.1.4], if M is nonnegative then the spectral radius $\rho(M)$ is an eigenvalue of M and there is a nonnegative eigenvector $\vec{x}$ with $M\vec{x} = \rho(M)\vec{x}$. Such a vector $\vec{x}$ is called *dominant*. Further, if M is nonnegative and strongly connected then $\vec{x}$ is strictly positive in all components and the eigenspace associated with $\rho(M)$ is one-dimensional.

3 Algorithms

Given a Markov chain $\mathcal{M}$, an initial distribution ι, and a Büchi automaton $\mathcal{A}$ whose alphabet is the state space of $\mathcal{M}$, the *probabilistic model-checking problem* is to compute $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$.

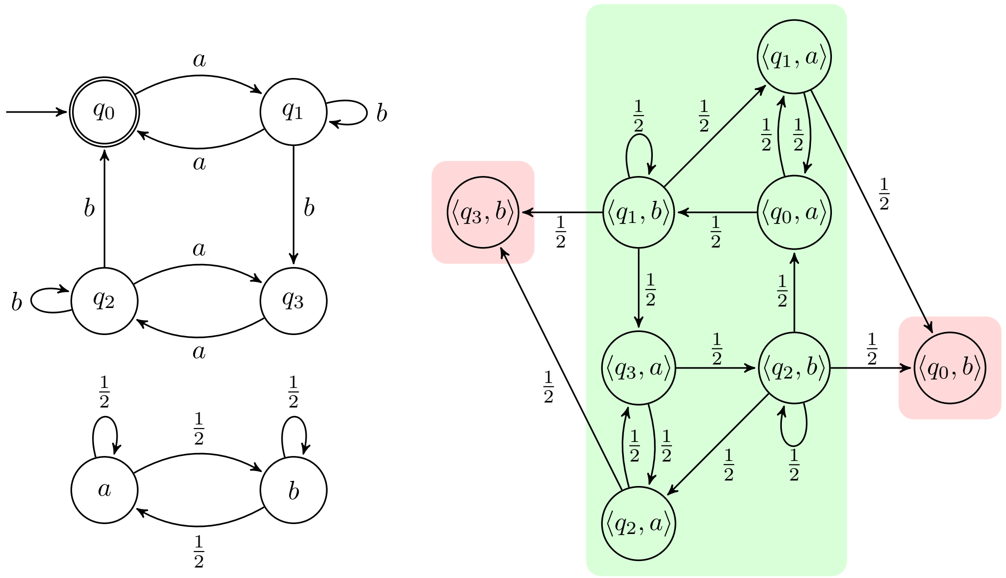


Figure 2 The UBA from Figure 1 and the Markov chain $\mathcal{M}$ on the left, and their product, B, on the right. The (single) accepting recurrent SCC is shaded green, and the two other SCCs are shaded red.

This problem is PSPACE-complete [8, 7], but solvable in polynomial time if $\mathcal{A}$ is deterministic. For UBAs a polynomial-time algorithm was described in [2, 3]. In this paper we obtain a faster algorithm (recall that E is the set of transitions in the Markov chain):

▶ **Theorem 7.** *Given a Markov chain $\mathcal{M} = (S, M)$, an initial distribution ι, and a UBA $\mathcal{A} = (Q, S, \delta, Q_0, F)$, one can compute $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ in time $O(|Q|^\kappa |S|^\kappa + |Q|^3|E| + |\delta|^2|E|)$.*

Before we prove this theorem in Section 3.3, we describe the algorithm from [2, 3] and analyse the runtime of an efficient implementation.

3.1 The Basic Linear System

Let $\mathcal{M} = (S, M)$ be a Markov chain, ι an initial distribution. Let $B \in \mathbb{R}^{(Q\times S)\times(Q\times S)}$ be the following matrix:

$$B_{\langle q,s\rangle,\langle q',s'\rangle} = \begin{cases} M_{s,s'} & \text{if } q' \in \delta(q,s) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Define $\vec{z} \in \mathbb{R}^{Q\times S}$ by $\vec{z}_{\langle q,s\rangle} = \Pr_s^{\mathcal{M}}(\mathcal{L}(\mathcal{A}_q))$. Then $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A})) = \sum_{q\in Q_0}\sum_{s\in S}\iota(s)\vec{z}_{\langle q,s\rangle}$. Lemma 4 in [3] implies that $\vec{z} = B\vec{z}$.

▶ **Example 8.** Consider the UBA $\mathcal{A}$ from Figure 1 and the two-state Markov chain $\mathcal{M}$ shown on the left of Figure 2. The weighted graph on the right of Figure 2 represents the matrix B, obtained from $\mathcal{A}$ and $\mathcal{M}$ according to Equation (1). It is natural to think of B as a product of $\mathcal{A}$ and $\mathcal{M}$. Notice that B is not stochastic: the sum of the entries in each row (equivalently, the total outgoing transition weight of a graph node) is not always one.

Although $\vec{z}$ is a solution the system of equations $\vec{\zeta} = B\vec{\zeta}$, this system does not uniquely identify $\vec{z}$. Indeed, any scalar multiple of $\vec{z}$ is a solution for these equations. To uniquely identify $\vec{z}$ by a system of linear equations, we need to analyse the SCCs of B.

All SCCs D satisfy $\rho(D) \leq 1$, see [3, Proposition 7]. An SCC D of B is called *recurrent* if $\rho(B_{D,D}) = 1$. It is called *accepting* if there is $\langle q, s\rangle \in D$ with $q \in F$.

▶ **Example 9.** The matrix B from Figure 2 has three SCCs, namely the two singleton sets $\{\langle q_0, b\rangle\}$ and $\{\langle q_3, b\rangle\}$, and $D = \{\langle q_0, a\rangle, \langle q_1, a\rangle, \langle q_1, b\rangle, \langle q_2, a\rangle, \langle q_2, b\rangle, \langle q_3, a\rangle\}$. Only D is recurrent; indeed, $\vec{y} = (\vec{y}_{\langle q_0,a\rangle}, \vec{y}_{\langle q_1,a\rangle}, \vec{y}_{\langle q_1,b\rangle}, \vec{y}_{\langle q_2,a\rangle}, \vec{y}_{\langle q_2,b\rangle}, \vec{y}_{\langle q_3,a\rangle})^\top = (2, 1, 3, 1, 3, 2)^\top$ is a dominant eigenvector with $B_{D,D}\vec{y} = \vec{y}$. Since q_0 is accepting, D is accepting recurrent.

Denote the set of accepting recurrent SCCs by $\mathcal{D}_+$ and the set of non-accepting recurrent SCCs by $\mathcal{D}_0$. By [3, Lemma 8], for $D \in \mathcal{D}_+$ we have $\vec{z}_d > 0$ for all $d \in D$, and for $D \in \mathcal{D}_0$ we have $\vec{z}_D = \vec{0}$. Hence, for $D \in \mathcal{D}_+$, there exists a *D-normaliser*, i.e., a vector $\vec{\mu} \in \mathbb{R}^D$ such that $\vec{\mu}^\top \vec{z}_D = 1$. This gives us a system of linear equations that identifies $\vec{z}$ uniquely [3]:

▶ **Lemma 10** (Lemma 12 in [3]). *Let $\mathcal{D}_+$ be the set of accepting recurrent SCCs, and $\mathcal{D}_0$ the set of non-accepting recurrent SCCs. For each $D \in \mathcal{D}_+$ let $\vec{\mu}(D)$ be a D-normaliser. Then $\vec{z}$ is the unique solution of the following linear system:*

$$\begin{aligned} & & \vec{\zeta} &= B\vec{\zeta} \\ &\textit{for all } D \in \mathcal{D}_+: & \vec{\mu}(D)^\top \vec{\zeta}_D &= 1 \\ &\textit{for all } D \in \mathcal{D}_0: & \vec{\zeta}_D &= \vec{0} \end{aligned} \tag{2}$$

Uniqueness follows from the fact that the system $\vec{\zeta} = B\vec{\zeta}$ describes the eigenspace of the dominant eigenvalue (here, 1) of a nonnegative strongly connected matrix (here, B), and such eigenspaces are one-dimensional. This leads to the following result:

▶ **Proposition 11.** *Suppose N is the runtime of an algorithm to calculate a normaliser for each accepting recurrent SCC. Then one can compute $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ in time $O(|Q|^\kappa |S|^\kappa) + N$.*

Proof. Lemma 10 implies correctness of the following procedure to calculate $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$:

1. Set up the matrix B from Equation (1).
2. Compute the SCCs of B.
3. For each SCC C, check whether C is recurrent.
4. For each accepting recurrent SCC D, compute its D-normaliser $\vec{\mu}(D)$.
5. Compute $\vec{z}$ by solving the linear system (2) in Lemma 10.
6. Compute $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A})) = \sum_{s\in S} \sum_{q\in Q_0} \iota(s)\vec{z}_{q,s}$.

One can set up B in time $O(|Q|^2|S|^2)$. Using Tarjan's algorithm one can compute the SCCs of B in time linear in the vertices and edges of B, hence in $O(|Q|^2|S|^2)$ [17]. One can find those SCCs D which are recurrent in time $O(|Q|^\kappa|S|^\kappa)$ by checking if $I - B_{D,D}$ is invertible. The linear system (2) has $O(|Q||S|)$ equations, and thus can be solved in time $O(|Q|^\kappa|S|^\kappa)$. Hence the total runtime is $O(|Q|^\kappa|S|^\kappa) + N$. ◀

In Section 3.2 we describe the combinatorial, *cut* based, approach from [2, 3] to calculating D-normalisers and analyse its complexity. In Section 3.3 we describe a novel linear-algebra based approach, which is faster in terms of the automaton.

3.2 Calculating D-Normalisers Using Cuts

For the remainder of the paper, let D be an accepting recurrent SCC. A *fibre over $s \in S$* is a subset of D of the form $\alpha \times \{s\}$ for some $\alpha \subseteq Q$. Given a fibre $f = \alpha \times \{s\}$ and a state $s' \in S$, if $M_{s,s'} > 0$ we define the fibre $f \triangleright s'$ as follows:

$$f \triangleright s' := \{\langle q, s'\rangle \mid q \in \delta(\alpha, s)\} \cap D.$$

If $M_{s,s'} = 0$, then $f \triangleright s'$ is undefined, and for $w \in S^*$ we define $f \triangleright w = f$ if $w = \varepsilon$ and $f \triangleright ws' = (f \triangleright w) \triangleright s'$. If $f = \{d\}$ for some $d \in D$ we may write $d \triangleright s'$ for $f \triangleright s'$.

We call a fibre c a *cut* if $c = d \triangleright v$ for some $v \in S^*$ and $d \in D$, and $c \triangleright w \neq \emptyset$ for all $w \in S^*$ whenever $c \triangleright w$ is defined. Note that if c is a cut then so is $c \triangleright w$ whenever it is defined. Given a cut $c \subseteq D$ we call its characteristic vector $[c] \in \{0,1\}^D$ a *cut vector.* In the example in Figure 2, it is easy to see that $\{\langle q_1, b\rangle\} = \langle q_0, a\rangle \triangleright b$ is a cut.

▶ **Lemma 12** (Lemma 10 in [3]). *There exists a cut. Any cut vector $\vec{\mu}$ is a normaliser, i.e., $\vec{\mu}^\top \vec{z}_D = 1$.*

Loosely speaking, $\vec{\mu}^\top \vec{z}_D \leq 1$ follows from unambiguousness, and $\vec{\mu}^\top \vec{z}_D \not< 1$ follows from an ergodicity argument (intuitively, all states in the cut are almost surely visited infinitely often). The following lemma is the basis for the cut computation algorithm in [2, 3]:

▶ **Lemma 13** (Lemma 17 in [3]). *Let $D \subseteq Q \times S$ be a recurrent SCC. Let $d \in D$. Suppose $w \in S^*$ is such that $d \triangleright w \ni d$ is not a cut. Then there are $v \in S^*$ and $e \neq d$ with $d \triangleright v \supseteq \{d, e\}$ and $e \triangleright w \neq \emptyset$. For any such e, $d \triangleright w \cap e \triangleright w = \emptyset$. Hence $d \triangleright vw \supseteq \{d, e\} \triangleright w \supsetneq d \triangleright w$.*

This suggests a way of generating an increasing sequence of fibres, culminating in a cut. We prove the following proposition:

▶ **Proposition 14.** *Let $D \subseteq Q \times S$ be a recurrent SCC. Denote by T the set of edges in $B_{D,D}$. One can compute a cut in time $O(|Q|^2|\delta||D| + |\delta||T|)$.*

Define, for some $d = \langle q, s\rangle \in D$, its *co-reachability* set $Co(d) \subseteq D$: it consists of those $e \in D$ such that there exists a word w with $\{d, e\} \subseteq d \triangleright w$. Note that $Co(d)$ is a fibre over s. In the example of Figure 2 we have that $Co(\langle q_0, a\rangle) = \{\langle q_0, a\rangle, \langle q_2, a\rangle\}$, with $\{\langle q_0, a\rangle, \langle q_2, a\rangle\} \in \langle q_0, a\rangle \triangleright ba$. The following lemma (proof in [12, Appendix B.2]) gives a bound on the time to compute $Co(d)$:

▶ **Lemma 15.** *One can compute $Co(d)$ in time $O(|Q||D| + |\delta||T|)$. Moreover, one can compute in time $O(|Q|^2|D| + |\delta||T|)$ a list $(CoPath(d)(e))_{e \in Co(d)}$ such that $CoPath(d)(e) \in S^*$ and $\{d, e\} \subseteq d \triangleright CoPath(d)(e)$ and $|CoPath(d)(e)| \leq |Q||D|$.*

The lemma is used in the proof of Proposition 14:

Proof sketch of Proposition 14. Starting from a singleton fibre $\{d\}$, where $d = \langle q, s\rangle \in D$ is chosen arbitrarily, we keep looking for words $v \in S^*$ that have the properties described in Lemma 13 to generate larger fibres $d \triangleright w$:

```
1. $w := \varepsilon$ (the empty word)
2. while $\exists\, v \in S^*$ and $\exists\, e \neq d$ such that $d \triangleright v \supseteq \{d, e\}$ and $e \triangleright w \neq \emptyset$ :
       $w := vw$
3. return $d \triangleright w$.
```

By [2, Lemma 18] the algorithm returns a cut. In every loop iteration the fibre $d \triangleright w$ increases, so the loop terminates after at most $|Q|$ iterations. For efficiency we calculate $Co(d)$ and $CoPath(d)$ using Lemma 15, and we use dynamic programming to maintain the set, *Survives*, of those $e \in D$ for which $e \triangleright w \neq \emptyset$ holds. Whenever a prefix v is added to w, we update *Survives* by processing v backwards. This leads to the following algorithm:

```
1. Calculate $Co(d)$ and $CoPath(d)$ using Lemma 15
2. $w := \varepsilon$; $Survives := (Q \times \{s\}) \cap D$
```

```
3. while $\exists\, e \in \mathit{Co}(d) \setminus \{d\}$ such that $e \in \mathit{Survives}$:
       $v_0 = s$; $v_1 \dots v_n := \mathit{CoPath}(d)(e)$
       for $i = n, n-1, \dots, 1$:
           $\mathit{Survives} := \{\langle p, v_{i-1}\rangle \in D \mid (\delta(p, v_{i-1}) \times \{v_i\}) \cap \mathit{Survives} \neq \emptyset\}$
       $w := v_1 \dots v_n w$
4. return $d \triangleright w$
```

The runtime analysis is in [12, Appendix B.2]. ◀

▶ **Example 16.** Letting $d = \langle q_0, a\rangle$ and $e = \langle q_2, a\rangle$ we have $\mathit{Co}(d) = \{d, e\}$ with $\mathit{CoPath}(d)(d) = \varepsilon$ and $\mathit{CoPath}(d)(e) = baa$. Initially we have $\mathit{Survives} = Q \times \{a\}$. In the first iteration the algorithm can only pick e. The inner loop updates *Survives* first to $\{q_0, q_1, q_2, q_3\} \times \{a\}$ (i.e., to itself), then to $\{q_1, q_2\} \times \{b\}$, and finally to $\{q_0, q_3\} \times \{a\}$. Now $(\mathit{Co}(d) \setminus d) \cap \mathit{Survives}$ is empty and the loop terminates. The algorithm returns the cut $d \triangleright baa = \{d, e\}$.

Applying Proposition 14 to the general procedure (Proposition 11) leads to the following result on the combinatorial approach:

▶ **Theorem 17.** *Given a Markov chain $\mathcal{M} = (S, M)$, an initial distribution ι, and a UBA $\mathcal{A} = (Q, S, \delta, Q_0, F)$, one can compute $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ in time $O(|Q|^\kappa |S|^\kappa + |Q|^3|\delta||S| + |\delta|^2|E|)$.*

3.3 Calculating D-Normalisers Using Linear Algebra

Recall that D is an accepting recurrent SCC. For $t \in S$ define the matrix $\Delta(t) \in \{0,1\}^{D \times D}$ as follows:

$$\Delta(t)_{\langle q,s\rangle,\langle q',s'\rangle} := \begin{cases} 1 & \text{if } s' = t,\ M_{s,t} > 0, \text{ and } q' \in \delta(q,s) \\ 0 & \text{otherwise} \end{cases}$$

Note that the graph of $\Delta(t)$ contains exactly the edges of the graph of $B_{D,D}$ that end in vertices in $Q \times \{t\}$. If $M_{s,t} > 0$ holds for all pairs (s,t), then the matrices $(\Delta(t))_{t \in S}$ generate a semigroup of matrices, all of which have spectral radius 1. Such semigroups were recently studied by Protasov and Voynov [16]. Specifically, Theorem 5 in [16] shows that there exists an affine subspace $\mathcal{F}$ of $\mathbb{R}^D$ which excludes $\vec{0}$ and is invariant under multiplication by matrices from the semigroup. Moreover, they provide a way to compute this affine subspace efficiently. One can show that cut vectors are orthogonal to $\mathcal{F}$. The key idea of our contribution is to generalise cut vectors to *pseudo-cuts*, which are vectors $\vec{\mu} \in \mathbb{R}^D$ that are orthogonal to $\mathcal{F}$. We will show (in Lemma 20 below) how to derive a D-normaliser based on a pseudo-cut that is non-zero only in components that are in a co-reachability set $\mathit{Co}(d)$ (from Lemma 15).

If $M_{s,t} = 0$ holds for some s, t (which will often be the case in model checking), then $\Delta(s)\Delta(t)$ is the zero matrix, which has spectral radius 0, not 1. Therefore, the results of [16] are not directly applicable and we have to move away from matrix semigroups. In the following we re-develop and generalise parts of the theory of [16] so that the paper is self-contained and products of $\Delta(s)\Delta(t)$ with $M_{s,t} = 0$ are not considered.

Let $w = s_1 s_2 \dots s_n \in S^*$. Define $\Delta(w) = \Delta(s_1)\Delta(s_2)\cdots\Delta(s_n)$. We say w is *enabled* if $M_{s_i, s_{i+1}} > 0$ holds for all $i \in \{1, \dots, n-1\}$. If $f \subseteq D$ is a fibre over s such that sw is enabled, we have $[f \triangleright w]^\top = [f]^\top \Delta(w)$. We overload the term *fibre over s* to describe any vector $\vec{\mu} \in \mathbb{R}^D$ such that $\vec{\mu}_{\langle q,s'\rangle} = 0$ whenever $s' \neq s$. We define *pseudo-cuts over s* to be fibres $\vec{\mu}$ over s such that $\vec{\mu}^\top \Delta(w) \vec{z} = \vec{\mu}^\top \vec{z}$ holds for all $w \in S^*$ such that sw is enabled. Let $c \subseteq Q \times \{s\}$ be a cut with sw enabled. Then $c \triangleright w$ is a cut, and $[c]^\top \Delta(w) \vec{z} = 1 = [c]^\top \vec{z}$ holds by Lemma 12. It follows that cut vectors are pseudo-cuts.

▶ **Example 18.** Since $c = \{\langle q_0, a\rangle, \langle q_2, a\rangle\}$ from Example 16 is a cut, $[c]$ is a pseudo-cut over a. Pseudo-cuts do not need to be combinations of cut vectors: although the fibre $f = \{\langle q_0, a\rangle, \langle q_1, a\rangle\}$ is not a cut, $[f]$ is a pseudo-cut over a.

Fix some $d = \langle q, s\rangle \in D$. Recall that $Co(d)$ consists of those $e \in D$ such that there exists a word w with $\{d, e\} \subseteq d \triangleright w$. We define *$Co(d)$-pseudo-cuts* to be pseudo-cuts $\vec{\mu}$ over s such that $\vec{\mu}_d \neq 0$ and $\vec{\mu}_e = 0$ holds for all $e \notin Co(d)$.

▶ **Example 19.** Any cut vector is a $Co(d)$-pseudo-cut for some $d \in D$, by definition, and so are scalar multiples of cut vectors. The vector $[f]$ in Example 18, however, is not a $Co(d)$-pseudo-cut, since $\langle q_1, a\rangle \notin Co(\langle q_0, a\rangle)$ and $\langle q_0, a\rangle \notin Co(\langle q_1, a\rangle)$.

From a $Co(d)$-pseudo-cut we can easily derive a D-normaliser:

▶ **Lemma 20.** *Let $\vec{\mu} \in \mathbb{R}^D$ be a $Co(d)$-pseudo-cut. Then $\frac{1}{\vec{\mu}_d}\vec{\mu}$ is a D-normaliser.*

Proof. Let w be an enabled word in M such that $d \triangleright w$ is a cut containing d. Such a word exists (see the proof sketch of Proposition 14). Since $([d]^\top \Delta(w))^\top = [d \triangleright w]$ is a D-normaliser (by Lemma 12), it suffices to prove that $\frac{1}{\vec{\mu}_d}\vec{\mu}^\top \vec{z} = [d]^\top \Delta(w)\vec{z}$.

We can write $\vec{\mu}$ as $\sum_{d' \in Co(d)} \vec{\mu}_{d'}[d']$, so $\vec{\mu}^\top \Delta(w) = \sum_{d' \in Co(d)} \vec{\mu}_{d'}[d']^\top \Delta(w)$. For any $d' \in Co(d) \setminus \{d\}$, let w' be such that $\{d, d'\} \subseteq d \triangleright w'$. Now we see that $d' \in d \triangleright ww'$, and since $d \triangleright w$ is a cut so are $d \triangleright ww'$ and $d \triangleright ww'w$. Thus,

$$[d]^\top \Delta(w)\vec{z} \;=\; [d]^\top \Delta(ww'w)\vec{z} \;\geq\; [d]^\top \Delta(w)\vec{z} + [d']^\top \Delta(w)\vec{z},$$

which implies $[d']^\top \Delta(w)\vec{z} = 0$ for every $d' \in Co(d) \setminus \{d\}$. This means that

$$\vec{\mu}^\top \Delta(w)\vec{z} = \sum_{d' \in Co(d)} \vec{\mu}_{d'}[d']^\top \Delta(w)\vec{z} = \vec{\mu}_d[d]^\top \Delta(w)\vec{z}.$$

Since $\vec{\mu}$ is a pseudo-cut, this implies that $\frac{1}{\vec{\mu}_d}\vec{\mu}^\top \vec{z} = \frac{1}{\vec{\mu}_d}\vec{\mu}^\top \Delta(w)\vec{z} = [d]^\top \Delta(w)\vec{z}$. ◀

By Lemma 20, to find a D-normaliser it suffices to find a $Co(d)$-pseudo-cut. Fix a dominant eigenvector $\vec{y}$ of $B_{D,D}$ so that $\vec{y}$ is strictly positive in all components. One can compute such $\vec{y}$ in time $O(|D|^\kappa)$. By [2, Lemma 8] the vector $\vec{z}_D$ is also a dominant eigenvector of $B_{D,D}$, hence $\vec{y}$ and $\vec{z}_D$ (the latter of which is yet unknown) are scalar multiples. In order to compute a $Co(d)$-pseudo-cut, we compute a basis for the space spanned by $\Delta(w)\vec{y}$ for all enabled words w. We use a technique similar to the one employed by Tzeng in [18] for checking equivalence of probabilistic automata. To make this more efficient, we compute separate basis vectors for each $s \in S$. Define $\Delta'(t) \in \{0,1\}^{D \times D}$ as $\Delta'(t)_{\langle q_1, s_1\rangle, \langle q_2, s_2\rangle} = 1$ if $q_1 = q_2$ and $s_1 = s_2 = t$ and 0 otherwise. Note that $\Delta(s)\Delta'(s) = \Delta(s)$ holds for all $s \in S$.

▶ **Lemma 21.** *Suppose $\vec{y} = B_{D,D}\vec{y}$ is given. Denote by $V(s) \subseteq \mathbb{R}^D$ the vector space spanned by the vectors $\Delta'(s)\Delta(w)\vec{y}$ for $w \in S^*$ and $s \in S$. Let $Q_{D,t} = (Q \times \{t\}) \cap D$ and let $E(t) = \{(s, t) \mid M_{s,t} > 0\}$ be the set of edges in M that end in t. One can compute a basis $R(s)$ of $V(s)$ for all $s \in S$ in time $O(|Q|^2 \sum_{t \in S} |Q_{D,t}||E(t)|)$, where for each $\vec{r} \in R(s)$ we have $\vec{r} = \Delta'(s)\Delta(w)\vec{y}$ for some enabled word sw.*

Proof sketch. Fix an arbitrary total order $<_S$ on S. We define a total order $\ll_S$ on S^* as the "shortlex" order but with words read from right to left. That is, the empty word ε is the smallest element, and for $v, w \in S^*$ and $s, t \in S$, we have $vs \ll_S wt$ if (1) $|vs| < |wt|$ or (2) $|vs| = |wt|$ and $s <_S t$ or (3) $|vs| = |wt|$ and $s = t$ and $v \ll_S w$.

We use a technique similar to the one by Tzeng in [18]. At every step in the algorithm, *worklist* is a set of pairs $(sw, \Delta'(s)\Delta(w)\vec{y})$. We write $\min_{\ll_S}(\textit{worklist})$ to denote the pair in *worklist* where sw is minimal with respect to $\ll_S$.

```
1. for each $s \in S$, let $R(s) := \{\Delta'(s)\vec{y}\}$ and $R(s)_\perp := \{\Delta'(s)\vec{y}\}$
2. $worklist := \{(st, \Delta'(s)\Delta(t)\vec{y}) \mid M_{s,t} > 0\}$
3. while $worklist \neq \emptyset$:
       $(tw, \vec{u}) := \min_{\ll_S}(worklist)$; $worklist := worklist \setminus \{(tw, \vec{u})\}$
       Using the Gram-Schmidt process[1], let $\vec{u}_\perp$ be the orthogonalisation of $\vec{u}$ against $R_\perp(t)$
       if $\vec{u}_\perp \neq \vec{0}$, i.e., if $\vec{u}$ is linearly independent of $R_\perp(t)$:
           $R(t) := R(t) \cup \{\vec{u}\}$ and $R_\perp(t) := R_\perp(t) \cup \{\vec{u}_\perp\}$
           $worklist := worklist \cup \{(stw, \Delta'(s)\Delta(t)\vec{u}) \mid M_{s,t} > 0\}$
4. return $R(s)$ for all $s \in S$
```

At any point and for all $s \in S$, the sets $R(s)$ and $R(s)_\perp$ span the same vector space, and this space is a subspace of $V(s)$. The sets $R(s)$ and $R(s)_\perp$ consist of linearly independent fibres over s, and these fibres are possibly nonzero only in the $Q_{D,s}$-components. Hence $|\bigcup_{s \in S} R(s)| \leq |D|$ and thus there are at most $|D|$ iterations of the while loop that increase *worklist*. At every iteration where $\vec{u}$ is dependent on $R(t)_\perp$ the set *worklist* decreases by one, and therefore the algorithm terminates. In [12, Appendix B.3] we prove that in the end we have that $R(s)$ spans $V(s)$, and we analyse the runtime. ◀

▶ **Example 22.** Let us return to our running example. We see that the vector $\vec{y} = (\vec{y}_{\langle q_0,a\rangle}, \vec{y}_{\langle q_1,a\rangle}, \vec{y}_{\langle q_1,b\rangle}, \vec{y}_{\langle q_2,a\rangle}, \vec{y}_{\langle q_2,b\rangle}, \vec{y}_{\langle q_3,a\rangle})^\top = (2,1,3,1,3,2)^\top$ is a dominant eigenvector of $B_{D,D}$. Fix the order $a <_S b$. Step 1 initialises $R(a)$ to $\{\Delta'(a)\vec{y}\}$ and $R(b)$ to $\{\Delta'(b)\vec{y}\}$, where $\Delta'(a)\vec{y} = (2,1,0,1,0,2)^\top$ and $\Delta'(b)\vec{y} = (0,0,3,0,3,0)^\top$. Step 2 computes $\Delta'(a)\Delta(a)\vec{y} = (1,2,0,2,0,1)^\top$, which is linearly independent of $\Delta'(a)\vec{y}$. However, $\Delta'(b)\Delta(a)\vec{y} = (0,0,3,0,3,0)^\top = \Delta'(b)\vec{y}$. Also, $\Delta'(a)\Delta(b)\vec{y} = (3,0,0,0,0,3)^\top = 2\Delta'(a)\vec{y} - \Delta'(a)\Delta(a)\vec{y}$ and $\Delta'(b)\Delta(b)\vec{y} = (0,0,3,0,3,0)^\top = \Delta'(b)\vec{y}$. One can check that $\Delta'(a)\Delta(aa)\vec{y} = \Delta'(a)\vec{y}$ and $\Delta'(b)\Delta(aa)\vec{y} = \Delta'(b)\vec{y}$. Hence the algorithm returns $R(a) = \{(2,1,0,1,0,2)^\top, (1,2,0,2,0,1)^\top\}$ and $R(b) = \{(0,0,3,0,3,0)^\top\}$.

Fix $d = \langle q, s\rangle \in D$ for the rest of the paper. The following lemma characterises $Co(d)$-pseudo-cuts in a way that is efficiently computable:

▶ **Lemma 23.** *A vector $\vec{\mu} \in \mathbb{R}^D$ with $\vec{\mu}_d = 1$ and $\vec{\mu}_e = 0$ for all $e \notin Co(d)$ is a $Co(d)$-pseudo-cut if and only if $\vec{\mu}^\top \vec{r} = \vec{\mu}^\top \vec{y}$ holds for all $\vec{r} \in R(s)$.*

For an intuition of the proof, consider the affine space, $\mathcal{F} \subseteq \mathbb{R}^D$, affinely spanned by those $\Delta'(s)\Delta(w)\vec{y}$ for which sw is enabled. This affine space was alluded to in the beginning of this subsection and is visualised as a blue straight line on the right of Figure 1. The shaded plane in this figure is the vector space of pseudo-cuts over s. This space is orthogonal to $\mathcal{F}$. The following lemma says that $\mathcal{F}$ is affinely spanned by the points in $R(s)$. This strengthens the property of $R(s)$ in Lemma 21 where $R(s)$ was defined to span a *vector* space.

▶ **Lemma 24.** *Let $w \in S^*$ be such that sw is enabled. By the definition of $R(s)$ there are $\gamma_{\vec{r}} \in \mathbb{R}^D$ for each $\vec{r} \in R(s)$ such that $\Delta'(s)\Delta(w)\vec{y} = \sum_{\vec{r} \in R(s)} \gamma_{\vec{r}}\vec{r}$. We have $\sum_{\vec{r} \in R(s)} \gamma_{\vec{r}} = 1$.*

Proof. Let c be a cut containing d. Since $R(s)$ is a basis, for any $\vec{r} = \Delta'(s)\Delta(w_{\vec{r}})\vec{y} \in R(s)$ the word $sw_{\vec{r}}$ is enabled. Therefore, $c \triangleright w_{\vec{r}}$ is a cut and by Lemma 12 we have $[c \triangleright w_{\vec{r}}]^\top \vec{y} = [c]^\top \vec{y}$.

[1] For good numerical stability, one should use the so-called Modified Gram-Schmidt process [10, Chapter 19].

Hence $[c]^\top \vec{r} = [c]^\top \Delta'(s)\Delta(w_{\vec{r}})\vec{y} = [c]^\top \Delta(w_{\vec{r}})\vec{y} = [c \triangleright w_{\vec{r}}]^\top \vec{y} = [c]^\top \vec{y}$. Moreover, we have:

$$\begin{aligned}
[c]^\top \vec{y} &= [c]^\top \Delta(w)\vec{y} && \text{since } sw \text{ is enabled and by Lemma 12}\\
&= [c]^\top \Delta'(s)\Delta(w)\vec{y} && \text{since } [c] \text{ is a fibre over } s\\
&= [c]^\top \sum_{\vec{r}\in R(s)} \gamma_{\vec{r}}\vec{r} && \text{by the definition of } \gamma_{\vec{r}}\\
&= [c]^\top \vec{y} \sum_{\vec{r}\in R(s)} \gamma_{\vec{r}} && \text{as argued above.}
\end{aligned}$$

Therefore, $\sum_{\vec{r}\in R(s)} \gamma_{\vec{r}} = 1$. ◀

Now we can prove Lemma 23:

Proof of Lemma 23. For the “if” direction, let w be such that sw is enabled, and it suffices to show that $\vec{\mu}^\top \Delta(w)\vec{y} = \vec{\mu}^\top \vec{y}$. By Lemma 24 there are $\gamma_{\vec{r}}$ such that $\Delta'(s)\Delta(w)\vec{y} = \sum_{\vec{r}\in R(s)} \gamma_{\vec{r}}\vec{r}$ and $\sum_{\vec{r}\in R(s)} \gamma_{\vec{r}} = 1$. We have:

$$\vec{\mu}^\top \Delta(w)\vec{y} \;=\; \vec{\mu}^\top \Delta'(s)\Delta(w)\vec{y} \;=\; \sum_{\vec{r}\in R(s)} \gamma_{\vec{r}}\vec{\mu}^\top \vec{r} \;=\; \sum_{\vec{r}\in R(s)} \gamma_{\vec{r}}\vec{\mu}^\top \vec{y} \;=\; \vec{\mu}^\top \vec{y}\,,$$

where the last equality is from Lemma 24.

For the “only if” direction, suppose $\vec{\mu}$ is a $Co(d)$-pseudo-cut. Let $\vec{r} = \Delta'(s)\Delta(w_{\vec{r}})\vec{y} \in R(s)$. Then $sw_{\vec{r}}$ is enabled and $\vec{\mu}^\top \vec{r} = \vec{\mu}^\top \Delta'(s)\Delta(w_{\vec{r}})\vec{y} = \vec{\mu}^\top \Delta(w_{\vec{r}})\vec{y} = \vec{\mu}^\top \vec{y}$. ◀

▶ **Example 25.** In Example 16 we derived that $\vec{y} = (2,1,3,1,3,2)^\top$ and $R(a) = \{(2,1,0,1,0,2)^\top, (1,2,0,2,0,1)^\top\}$. The cut vector $\vec{\mu} = (1,0,0,1,0,0)^\top$ from Example 18 satisfies $\vec{\mu}^\top \vec{r} = 3 = \vec{\mu}^\top \vec{y}$ for both $\vec{r} \in R(a)$.

Using Lemmas 15, 21 and 23 we obtain:

▶ **Proposition 26.** *Let $D \subseteq Q \times S$ be a recurrent SCC. Denote by T_D the set of edges of $B_{D,D}$. For $t \in S$, let $E(t)$ denote the set of edges of M that end in t, and let $Q_{D,t} = (Q \times \{t\}) \cap D$. Let $d = \langle q, s\rangle \in D$. One can compute a $Co(d)$-pseudo-cut in time $O(|D|^\kappa + |Q||D| + |\delta||T_D| + |Q|^2 \sum_{t\in S} |Q_{D,t}||E(t)|)$.*

Now our main result follows, which we restate here:

▶ **Theorem 7.** *Given a Markov chain $\mathcal{M} = (S, M)$, an initial distribution ι, and a UBA $\mathcal{A} = (Q, S, \delta, Q_0, F)$, one can compute $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ in time $O(|Q|^\kappa |S|^\kappa + |Q|^3|E| + |\delta|^2|E|)$.*

4 Discussion

We have analysed two algorithms for computing normalisers: the cut-based one by Baier et al. [2, 3], and a new one, which draws from techniques by Protasov and Voynov [16] for the analysis of matrix semigroups. The first approach is purely combinatorial, and in terms of the automaton, an efficient implementation runs in time $O(|Q|^3|\delta| + |\delta|^2) = O(|Q|^3|\delta|)$ (Proposition 14).

The second approach combines a linear-algebra component to compute $R(s)$ with a combinatorial algorithm to compute the co-reachability set $Co(d)$. In terms of the automaton, the linear-algebra component runs in time $O(|Q|^3)$ (Lemma 21), while the combinatorial part runs in time $O(|\delta|^2)$, leading to an overall runtime of $O(|Q|^3 + |\delta|^2)$. Note that for all $r \in [1, 2]$, if $|\delta| = \Theta(|Q|^r)$ then the second approach is faster by at least a factor of $|Q|$.

Although it is not the main focus of this paper, we have analysed also the model-checking problem, where a non-trivial Markov chain is part of the input. The purely combinatorial algorithm runs in time $O(|Q|^\kappa|S|^\kappa + |Q|^3|\delta||S| + |\delta|^2|E|)$, and the linear-algebra based algorithm in time $O(|Q|^\kappa|S|^\kappa + |Q|^3|E| + |\delta|^2|E|)$. There are cases in which the latter is asymptotically worse, but not if $\kappa = 3$ (i.e., solving linear systems in a normal way such as Gaussian elimination) or if $|E|$ is $O(|S|)$.

It is perhaps unsurprising that a factor of $|\delta|^2$ from the computation of $Co(d)$ occurs in the runtime, as it also occurs when one merely verifies the unambiguousness of the automaton, by searching the product of the automaton with itself. Can the factor $|\delta|^2$ (which may be quartic in $|Q|$) be avoided?

References

1 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT press, 2008.

2 Christel Baier, Stefan Kiefer, Joachim Klein, Sascha Klüppelholz, David Müller, and James Worrell. Markov chains and unambiguous Büchi automata. In *Proceedings of Computer Aided Verification (CAV)*, volume 9779 of *LNCS*, pages 23–42, 2016.

3 Christel Baier, Stefan Kiefer, Joachim Klein, Sascha Klüppelholz, David Müller, and James Worrell. Markov chains and unambiguous automata. Draft journal submission. Available at https://arxiv.org/abs/1605.00950, 2019.

4 Abraham Berman and Robert J. Plemmons. *Nonnegative matrices in the mathematical sciences.* SIAM, 1994.

5 Manuel Bodirsky, Tobias Gärtner, Timo von Oertzen, and Jan Schwinghammer. Efficiently Computing the Density of Regular Languages. In *LATIN 2004: Theoretical Informatics*, pages 262–270. Springer, 2004.

6 James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28:231–236, 1974.

7 Doron Bustan, Sasha Rubin, and Moshe Y. Vardi. Verifying ω-Regular Properties of Markov Chains. In *16th International Conference on Computer Aided Verification (CAV)*, volume 3114 of *Lecture Notes in Computer Science*, pages 189–201. Springer, 2004.

8 Costas Courcoubetis and Mihalis Yannakakis. The Complexity of Probabilistic Verification. *Journal of the ACM*, 42(4):857–907, 1995.

9 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.

10 Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, second edition, 2002.

11 M. James. The generalised inverse. *The Mathematical Gazette*, 62:109–114, 1978.

12 Stefan Kiefer and Cas Widdershoven. Efficient Analysis of Unambiguous Automata Using Matrix Semigroup Techniques (full version). arXiv:1906.10093, 2016. URL: `http://arxiv.org/abs/1906.10093`.

13 Vidyadhar G. Kulkarni. *Modeling and Analysis of Stochastic Systems.* Chapman & Hall, 1995.

14 François Le Gall. Powers of Tensors and Fast Matrix Multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC'14, pages 296–303. ACM, 2014.

15 Marko D. Petković and Predrag S. Stanimirović. Generalised matrix inversion is not harder than matrix multiplication. *Journal of Computational and Applied Mathematics*, 230:270–282, 2009.

16 V.Yu. Protasov and A.S. Voynov. Matrix semigroups with constant spectral radius. *Linear Algebra and its Applications*, 513:376–408, 2017.

17 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

18 Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing*, 21(2):216–227, 1992.

On the Mortality Problem: From Multiplicative Matrix Equations to Linear Recurrence Sequences and Beyond

Paul C. Bell
Department of Computer Science, Byrom Street, Liverpool John Moores University,
Liverpool, L3-3AF, UK
p.c.bell@ljmu.ac.uk

Igor Potapov
Department of Computer Science, Ashton Building, Ashton Street,
University of Liverpool, Liverpool, L69-3BX, UK
potapov@liverpool.ac.uk

Pavel Semukhin
Department of Computer Science, University of Oxford, Wolfson Building,
Parks Road, Oxford, OX1 3QD, UK
pavel.semukhin@cs.ox.ac.uk

Abstract

We consider the following variant of the Mortality Problem: given $k \times k$ matrices $A_1, A_2, \dots, A_t$, does there exist nonnegative integers $m_1, m_2, \dots, m_t$ such that the product $A_1^{m_1} A_2^{m_2} \cdots A_t^{m_t}$ is equal to the zero matrix? It is known that this problem is decidable when $t \leq 2$ for matrices over algebraic numbers but becomes undecidable for sufficiently large t and k even for integral matrices.

In this paper, we prove the first decidability results for $t > 2$. We show as one of our central results that for $t = 3$ this problem in any dimension is Turing equivalent to the well-known Skolem problem for linear recurrence sequences. Our proof relies on the Primary Decomposition Theorem for matrices that was not used to show decidability results in matrix semigroups before. As a corollary we obtain that the above problem is decidable for $t = 3$ and $k \leq 3$ for matrices over algebraic numbers and for $t = 3$ and $k = 4$ for matrices over real algebraic numbers. Another consequence is that the set of triples (m_1, m_2, m_3) for which the equation $A_1^{m_1} A_2^{m_2} A_3^{m_3}$ equals the zero matrix is equal to a finite union of direct products of semilinear sets.

For $t = 4$ we show that the solution set can be non-semilinear, and thus it seems unlikely that there is a direct connection to the Skolem problem. However we prove that the problem is still decidable for upper-triangular 2×2 rational matrices by employing powerful tools from transcendence theory such as Baker's theorem and S-unit equations.

2012 ACM Subject Classification Theory of computation → Computability; Mathematics of computing → Computations on matrices

Keywords and phrases Linear recurrence sequences, Skolem's problem, mortality problem, matrix equations, primary decomposition theorem, Baker's theorem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.83

Related Version A full version of the paper is available at `https://arxiv.org/abs/1902.10188`.

Funding *Igor Potapov*: Partially supported by EPSRC grants EP/R018472/1 and EP/M00077X/1.
Pavel Semukhin: Supported by ERC grant AVS-ISS (648701).

Acknowledgements We thank Prof. James Worrell for useful discussions, particularly related to S-unit equations.

44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).
Editors: Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen; Article No. 83; pp. 83:1–83:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A large number of naturally defined matrix problems are still unanswered, despite the long history of matrix theory. Some of these questions have recently drawn renewed interest in the context of the analysis of digital processes, verification problems, and links with several fundamental questions in mathematics [11, 7, 37, 39, 38, 35, 17, 14, 15, 36, 5, 41, 27].

One of these challenging problems is the *Mortality Problem* of whether the zero matrix belongs to a finitely generated matrix semigroup. It plays a central role in many questions from control theory and software verification [44, 10, 8, 36, 2]. The mortality problem has been known to be undecidable for matrices in $\mathbb{Z}^{3\times 3}$ since 1970 [40] and the current undecidability bounds for the $M(d, k \times k)$ problem (i.e. the mortality problem for semigroups generated by d matrices of size $k \times k$) are $M(6, 3 \times 3)$, $M(4, 5 \times 5)$, $M(3, 9 \times 9)$ and $M(2, 15 \times 15)$, see [13]. It is also known that the problem is NP-hard for 2×2 integer matrices [4] and is decidable for 2×2 integer matrices with determinant $0, \pm 1$ [34]. In the case of finite matrix semigroups of any dimension the mortality problem is known to be PSPACE-complete [26].

In this paper, we study a very natural variant of the mortality problem when matrices must appear in a fixed order (i.e. under bounded language constraint): *Given $k \times k$ matrices $A_1, A_2, \ldots, A_t$ over a ring $\mathcal{F}$, do there exist $m_1, m_2, \ldots, m_t \in \mathbb{N}$ such that $A_1^{m_1} A_2^{m_2} \ldots A_t^{m_t} = \mathbf{O}_{k,k}$, where $\mathbf{O}_{k,k}$ is $k \times k$ zero matrix?*

In general (i.e. replacing $\mathbf{O}_{k,k}$ by other matrices) this problem is known as the solvability of multiplicative matrix equations and has been studied for many decades. In its simplest form, when $k = 1$, the problem was studied by Harrison in 1969 [21] as a reformulation of the "accessibility problem" for linear sequential machines. The case $t = 1$ was solved in polynomial time in a celebrated paper by Kannan and Lipton in 1980 [25]. The case $t = 2$, i.e. $A^x B^y = C$ where A, B and C are commuting matrices was solved by Cai, Lipton and Zalcstein in 1994 [12]. Later, in 1996, the solvability of matrix equations over commuting matrices was solved in polynomial time in [1] and in 2010 it was shown in [3] that $A^x B^y = C$ is decidable for non-commuting matrices of any dimension with algebraic coefficients by a reduction to the commutative case from [1]. However, it was also shown in [3] that the solvability of multiplicative matrix equations for sufficiently large natural numbers t and k is in general undecidable by an encoding of *Hilbert's tenth problem* and in particular for the mortality problem with bounded language constraint. In 2015 it was also shown that the undecidability result holds for such equations with unitriangluar matrices [31] and also in the case of specific equations with nonnegative matrices [23].

The decidability of matrix equations for non-commuting matrices is only known as corollaries of either recent decidability results for solving membership problem in 2×2 matrix semigroups [41, 42] or in the case of quite restricted classes of matrices, e.g. matrices from the Heisenberg group [27, 28] or row-monomial matrices over commutative semigroups [30]. In the other direction, progress has been made for matrix-exponential equations, but again in the case of commuting matrices [36].

In this paper, we prove the first decidability results for the above problem when $t = 3$ and $t = 4$. We will call these problems the ABC-Z and ABCD-Z problems, respectively. More precisely, we will show that the ABC-Z problem in any dimension is Turing equivalent to the Skolem problem (also known as Skolem-Pisot problem) which asks whether a given linear recurrence sequence ever reaches zero. Our proof relies on the Primary Decomposition Theorem for matrices (Theorem 2) that was not used to show decidability results in matrix semigroups before. As a corollary, we obtain that the ABC-Z problem is decidable for 2×2 and 3×3 matrices over algebraic numbers and also for 4×4 matrices over real algebraic numbers. Another consequence of the above equivalence is that the set of triples (m, n, ℓ) that satisfy the equation $A^m B^n C^\ell = \mathbf{O}_{k\times k}$ can be expressed as a finite union of direct products of semilinear sets.

In contrast to the ABC-Z problem, we show that the solution set of the ABCD-Z problem can be non-semilinear. This indicates that the ABCD-Z problem is unlikely to be related to the Skolem problem. However we will show that the ABCD-Z problem is decidable for upper-triangular 2×2 rational matrices. The proof of this result relies on powerful tools from transcendence theory such as Baker's theorem for linear forms in logarithms, S-unit equations from algebraic number theory and the Frobenius rank inequality from matrix analysis. More precisely, we will reduce the ABCD-Z equation for upper-triangular 2×2 rational matrices to an equation of the form $ax + by + cz = 0$, where x, y, z are S-units, and then use an upper bound on the solutions of this equation (as in Theorem 5). On the other hand, if we try to generalize this result to arbitrary 2×2 rational matrices or to upper-triangular matrices of higher dimension, then we end up with an equation that contain a sum of four or more S-units, and for such equations no effective upper bounds on their solutions are known. So, these generalizations seems to lie beyond the reach of current mathematical knowledge.

2 Preliminaries

We denote by $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{C}$ the sets of natural, integer, rational and complex numbers, respectively. Further, we denote by $\mathbf{A}$ the set of algebraic numbers and by $\mathbf{A}_\mathbb{R}$ the set of real algebraic numbers.

For a prime number p we define a valuation $v_p(x)$ for nonzero $x \in \mathbb{Q}$ as follows: if $x = p^k \frac{m}{n}$, where $m, n \in \mathbb{Z}$ and p does not divide m or n, then $v_p(x) = v_p(p^k \frac{m}{n}) = k$.

Throughout this paper $\mathcal{F}$ will denote either the ring of integers $\mathbb{Z}$ or one of the fields $\mathbb{Q}$, $\mathbf{A}$, $\mathbf{A}_\mathbb{R}$ or $\mathbb{C}$. We will use the notation $\mathcal{F}^{n\times m}$ for the set of $n \times m$ matrices over $\mathcal{F}$.

We denote by $\mathbf{e}_i$ the i'th standard basis vector of some dimension (which will be clear from the context). Let $\mathbf{O}_{n,m}$ be the zero matrix of size $n \times m$, $\mathbf{I}_n$ be the identity matrix of size $n \times n$, and $\mathbf{0}_n$ be the zero column vector of length n. Given a finite set of matrices $\mathcal{G} \subseteq \mathcal{F}^{n\times n}$, we denote by $\langle \mathcal{G} \rangle$ the multiplicative semigroup generated by $\mathcal{G}$.

If $A \in \mathcal{F}^{m\times m}$ and $B \in \mathcal{F}^{n\times n}$, then we define their direct sum as $A \oplus B = \left[\begin{array}{c|c} A & \mathbf{O}_{m,n} \\ \hline \mathbf{O}_{n,m} & B \end{array}\right]$. Let $C \in \mathcal{F}^{k\times k}$ be a square matrix. We write $\det(C)$ for the determinant of C. We call C singular if $\det(C) = 0$, otherwise it is said to be invertible (or non-singular). Matrices A and B from $\mathcal{F}^{k\times k}$ are called *similar* if there exists an invertible $k \times k$ matrix S (perhaps over a larger field containing $\mathcal{F}$) such that $A = SBS^{-1}$. In this case, S is said to be a similarity matrix transforming A to B.

We will also require the following inequality regarding ranks of matrices, known as the *Frobenius rank inequality*, see [24] for further details.

▶ **Theorem 1** (Frobenius Rank Inequality). *Let $A, B, C \in \mathcal{F}^{k\times k}$. Then*

$$\mathit{Rk}(AB) + \mathit{Rk}(BC) \leq \mathit{Rk}(ABC) + \mathit{Rk}(B)$$

In the proof of our first main result about the ABC-Z problem we will make use of the primary decomposition theorem for matrices.

▶ **Theorem 2** (Primary Decomposition Theorem [22]). *Let A be a matrix from $\mathcal{F}^{n\times n}$, where $\mathcal{F}$ is a field. Let $m_A(x)$ be the minimal polynomial for A such that*

$$m_A(x) = p_1(x)^{r_1} \cdots p_k(x)^{r_k},$$

where the $p_i(x)$ are distinct irreducible monic polynomials over $\mathcal{F}$ and the r_i are positive integers. Let W_i be the null space of $p_i(A)^{r_i}$ and let S_i be a basis for W_i. Then

(1) *$S_1 \cup \cdots \cup S_k$ is a basis for $\mathcal{F}^n$ and $\mathcal{F}^n = W_1 \oplus \cdots \oplus W_k$,*

(2) *each W_i is invariant under A, that is, $A\mathbf{x} \in W_i$ for any $\mathbf{x} \in W_i$,*

(3) *let S be a matrix whose columns are equal to the basis vectors from $S_1 \cup \cdots \cup S_k$; then*

$$S^{-1}AS = A_1 \oplus \cdots \oplus A_k,$$

where each A_i is a matrix over $\mathcal{F}$ of the size $|S_i| \times |S_i|$, and the minimal polynomial of A_i is equal to $p_i(x)^{r_i}$.

The next two propositions are well-known facts.

▶ **Proposition 3.** *If $p(x)$ is a polynomial over a field $\mathcal{F}$, where $\mathcal{F}$ is either $\mathbb{Q}$, $\mathbf{A}$ or $\mathbf{A}_{\mathbb{R}}$, then the primary decomposition of $p(x)$ can be algorithmically computed.*

▶ **Proposition 4.** *Let $A \in \mathcal{F}^{n\times n}$ and $m_A(x)$ be the minimal polynomial of A. Then A is invertible if and only if $m_A(x)$ has nonzero free coefficient, i.e., $m_A(x)$ is not divisible by x.*

Our proof of the decidability of ABCD-Z problem for 2×2 upper-triangular rational matrices relies on the following result which is proved using Baker's theorem on linear forms in logarithms (see Corollary 4 in [16] and also [18]).

▶ **Theorem 5.** *Let $S = \{p_1, \dots, p_s\}$ be a finite collection of prime numbers and let a, b, c be relatively prime nonzero integers, that is, $\gcd(a, b, c) = 1$.*

If x, y, z are relatively prime nonzero integers composed of primes from S that satisfy the equation $ax + by + cz = 0$, then

$$\max\{|x|, |y|, |z|\} < \exp(s^{Cs} P^{4/3} \log A)$$

for some constant C, where $P = \max\{p_1, \dots, p_s\}$ and $A = \max\{|a|, |b|, |c|, 3\}$.

▶ Remark. Rational numbers whose numerator and denominator are divisible only by the primes from S are called *S-units*.

3 Linear recurrence sequences and semilinear sets

There is a long history in computer science and mathematics of studying sequences of numbers defined by some recurrence relation, where the next value in the sequence depends upon some "finite memory" of previous values in the sequence. Possibly the simplest, and certainly the most well known of these, is the *Fibonacci sequence*, which may be defined by the recurrence $F(n) = F(n-1) + F(n-2)$ with $F(0) = F(1) = 1$ being given as the *initial conditions* of the sequence. We may generalise the Fibonacci sequence to define a *linear recurrence sequence*, which find application in many areas of mathematics and other sciences and for which many questions remain open. Let $\mathcal{F}$ be a ring; a sequence $(u_n)_{n=0}^{\infty}$ is called a *linear recurrence sequence* (1-LRS) if it satisfies a relation of the form:

$$u_n = a_{k-1}u_{n-1} + \cdots + a_1 u_{n-k+1} + a_0 u_{n-k},$$

for any $n \geq k$, where each $a_0, a_1, \dots, a_{k-1} \in \mathcal{F}$ are fixed coefficients[1]. Such a sequence $(u_n)_{n=0}^{\infty}$ is said to be of depth k if it satisfies no shorter linear recurrence relation (for any

[1] In the literature, such a sequence is ordinarily called an LRS; we use the nomenclature 1-LRS since we will study a multidimensional variant of this concept. Also, 1-LRS are usually considered over integers, but in the present paper we will consider such sequences over algebraic numbers.

$k' < k$). We call the initial k values of the sequence $u_0, u_1, \dots, u_{k-1}$ the initial conditions of the 1-LRS. Given the initial conditions and coefficients of a 1-LRS, every element is uniquely determined.

The *zero set* of a 1-LRS is defined as follows: $\mathcal{Z}(u_n) = \{j \in \mathbb{N} \mid u_j = 0\}$.

There are various questions that one may ask regarding $\mathcal{Z}(u_n)$. One notable example relates to the famous "Skolem's problem" which is stated in the following way:

▶ **Problem 6** (Skolem's Problem)**.** *Given the coefficients and initial conditions of a depth k 1-LRS $(u_n)_{n=0}^{\infty}$, determine if $\mathcal{Z}(u_n)$ is the empty set.*

Skolem's problem has a long and rich history, see [19] for a good survey. We note here that the problem remains open despite properties of zero sets having been studied even since 1934 [43]. It is known that the Skolem problem is at least NP-hard [9] and that it is decidable for depth 3 over $\mathbf{A}$ and for depth 4 over $\mathbf{A}_{\mathbb{R}}$, see [44] and [33][2]. Other interesting questions are related to the structure of $\mathcal{Z}(u_n)$. We remind the reader the definition of semilinear sets.

▶ **Definition 7** (Semilinear set)**.** *A set $S \subseteq \mathbb{N}$ is called* semilinear *if it is the union of a finite set and finitely many arithmetic progressions.*

A seminal result regarding 1-LRSs is that there zero sets are semilinear.

▶ **Theorem 8** (Skolem, Mahler, Lech [32, 43, 29] and [19, 20])**.** *The zero set of a 1-LRS over $\mathbb{C}$ (or more generally over any field of characteristic 0) is semilinear.*

In particular, if $(u_n)_{n=0}^{\infty}$ is a 1-LRS whose coefficients and initial conditions are algebraic numbers, then one can algorithmically find a number $L \in \mathbb{N}$ such that for every $i = 0, \dots, L-1$, if we let $u_m^i = u_{i+mL}$, then

(1) *the sequence $(u_m^i)_{m=0}^{\infty}$ is a 1-LRS of the same depth as $(u_n)_{n=0}^{\infty}$, and*

(2) *either $\mathcal{Z}(u_m^i) = \mathbb{N}$ or $\mathcal{Z}(u_m^i)$ is finite.*

Note that in the above theorem we can decide whether $\mathcal{Z}(u_m^i)$ is finite or $\mathcal{Z}(u_m^i) = \mathbb{N}$ because $\mathcal{Z}(u_m^i) = \mathbb{N}$ if and only if $u_0^i = \cdots = u_{k-1}^i = 0$, where k is the depth of $(u_m^i)_{m=0}^{\infty}$.

We will also consider a stronger version of the Skolem problem.

▶ **Problem 9** (Strong Skolem's Problem)**.** *Given the coefficients and initial conditions of a 1-LRS $(u_n)_{n=0}^{\infty}$ over $\mathbf{A}$, find a description of the set $\mathcal{Z}(u_n)$. That is, find a finite set F such that $\mathcal{Z}(u_n) = F$ if $\mathcal{Z}(u_n)$ is finite or, if $\mathcal{Z}(u_n)$ is infinite, find a finite set F, a constant $L \in \mathbb{N}$ and numbers $i_1, \dots, i_t \in \{0, \dots, L-1\}$ such that*

$$\mathcal{Z}(u_n) = F \cup \{i_1 + mL : m \in \mathbb{N}\} \cup \dots \cup \{i_t + mL : m \in \mathbb{N}\}.$$

Using the Skolem-Mahler-Lech theorem we can prove an equivalence between the strong version of the Skolem problem and the standard version[3].

▶ **Theorem 10.** *Skolem's problem of depth k over $\mathbf{A}$ is Turing equivalent to the strong Skolem's problem of the same depth.*

Proof. Obviously, Skolem's problem is reducible to the strong Skolem's problem. We now show a reduction in the other direction.

Let $(u_n)_{n=0}^{\infty}$ be a depth-k 1-LRS over $\mathbf{A}$. By Theorem 8, we can algorithmically find a number L such that, for every $i = 0, \dots, L-1$, the sequence $u_m^i = u_{i+mL}$ is a 1-LRS of

[2] A proof of decidability for depth 5 was claimed in [19], although there is possibly a gap in the proof [37].

[3] This result was announced in [44] without a proof, probably with a similar construction in mind.

depth k which is either everywhere zero, that is, $\mathcal{Z}(u^i_m) = \mathbb{N}$ or $\mathcal{Z}(u^i_m)$ is finite. Recall that we can decide whether $\mathcal{Z}(u^i_m)$ is equal to $\mathbb{N}$ by considering the first k terms of $(u^i_m)_{m=0}^\infty$.

By definition, we have $\mathcal{Z}(u_n) = \bigcup_{i=0}^{L-1} \{i+L\mathcal{Z}(u^i_m)\}$. So, if $\mathcal{Z}(u^i_m) = \mathbb{N}$, then $\{i+L\mathcal{Z}(u^i_m)\} = \{i + mL : m \in \mathbb{N}\}$, and if $\mathcal{Z}(u^i_m)$ is finite, then so is $\{i + L\cdot\mathcal{Z}(u^i_m)\}$.

To finish the proof we need to show how to compute $\mathcal{Z}(u^i_m)$, and hence $\{i + L\cdot\mathcal{Z}(u^i_m)\}$, when it is finite. For this we will use an oracle for the Skolem problem. Let m' be the smallest number such that $\mathcal{Z}(u^i_{m+m'})$ is empty. Such m' exists because $\mathcal{Z}(u^i_m)$ is finite. Furthermore, $(u_{m+m'})_{m=0}^\infty$ is a 1-LRS of depth k for any m'. So, we ask the oracle for the Skolem problem to decide whether $\mathcal{Z}(u^i_{m+m'}) = \emptyset$ for each $m' \in \mathbb{N}$ starting from 0 until we find one for which $\mathcal{Z}(u^i_{m+m'})$ is empty. Note that we do not have any bound on m' because we do not even know the size of $\mathcal{Z}(u^i_m)$. All we know is that $\mathcal{Z}(u^i_m)$ is finite, and hence the above algorithm will eventually terminate. Since $\mathcal{Z}(u^i_m)$ is a subset of $\{0, \dots, m'\}$, then we can compute it by checking whether $u^i_m = 0$ for $m = 0, \dots, m'$. ◀

Linear recurrence sequences can also be represented using matrices [19]:

▶ **Lemma 11.** *Let $\mathcal{F}$ be a ring; for a sequence $(u_n)_{n=0}^\infty$ over $\mathcal{F}$ the following are equivalent:*

(1) *$(u_n)_{n=0}^\infty$ is a 1-LRS of depth k.*

(2) *There are vectors $\mathbf{u}, \mathbf{v} \in \mathcal{F}^k$ and a matrix $M \in \mathcal{F}^{k\times k}$ such that $u_n = \mathbf{u}^T M^n \mathbf{v}$ for $n \in \mathbb{N}$.*

Moreover, for any matrix $M \in \mathcal{F}^{k\times k}$, the sequence $u_n = (M^n)_{[1,k]}$ is a 1-LRS of depth at most k. On the other hand, if $(u_n)_{n=0}^\infty$ is a 1-LRS of depth k, then there is a matrix $M \in \mathcal{F}^{(k+1)\times(k+1)}$ such that $u_n = (M^n)_{[1,k+1]}$ for all $n \in \mathbb{N}$.

Lemma 11 motivates the following definition of n-dimensional Linear Recurrence Sequences (n-LRSs) which as we show later are related to the mortality problem for bounded languages.

▶ **Definition 12** (n-LRS)**.** *A multidimensional sequence $u_{m_1,m_2,\dots,m_n}$ is called an n-LRS of depth k over $\mathcal{F}$ if there exist two vectors $\mathbf{u}, \mathbf{v} \in \mathcal{F}^k$ and matrices $M_1, M_2, \dots, M_n \in \mathcal{F}^{k\times k}$ such that*

$$u_{m_1,m_2,\dots,m_n} = \mathbf{u}^T M_1^{m_1} M_2^{m_2} \cdots M_n^{m_n} \mathbf{v}.$$

4 The mortality problem for bounded languages

We remind the reader the definition of the mortality problem for bounded languages.

▶ **Problem 13** (Mortality for bounded languages)**.** *Given $k \times k$ matrices $A_1, \dots, A_t$ over a ring $\mathcal{F}$, do there exist $m_1, m_2, \dots, m_t \in \mathbb{N}$ such that*

$$A_1^{m_1} A_2^{m_2} \dots A_t^{m_t} = \mathbf{O}_{k,k}.$$

Recall that for $t = 3$ and $t = 4$ this problem is called the ABC-Z and ABCD-Z problem, respectively. Our first main result is that the ABC-Z problem is computationally equivalent to the Skolem problem for 1-LRS. Our reduction holds in any dimension and over the same number field which means that any new decidability results for the Skolem problem will automatically extend the decidability of ABC-Z equations and can immediately lead to new decidability results for equations in dimensions 2, 3 and 4. For the proof we will need the following technical lemma.

▶ **Lemma 14.** *Let $\mathcal{F}$ be a field, and suppose $A, B, C \in \mathcal{F}^{k\times k}$ are matrices of the form*

$$A = \left[\begin{array}{c|c} A_{s,s} & \mathbf{O}_{s,k-s} \\ \hline \mathbf{O}_{k-s,s} & \mathbf{O}_{k-s,k-s} \end{array}\right], \; B = \left[\begin{array}{c|c} B_{s,t} & X_{s,k-t} \\ \hline Y_{k-s,t} & Z_{k-s,k-t} \end{array}\right], \; C = \left[\begin{array}{c|c} C_{t,t} & \mathbf{O}_{t,k-t} \\ \hline \mathbf{O}_{k-t,t} & \mathbf{O}_{k-t,k-t} \end{array}\right]$$

for some $s, t \le k$, where $A_{s,s}$, $B_{s,t}$, $X_{s,k-t}$, $Y_{k-s,t}$, $Z_{k-s,k-t}$ and $C_{t,t}$ are matrices over $\mathcal{F}$ whose dimensions are indicated by their subscripts (in particular, $A = A_{s,s} \oplus \mathbf{O}_{k-s,k-s}$ and $C = C_{t,t} \oplus \mathbf{O}_{k-t,k-t}$). If $A_{s,s}$ and $C_{t,t}$ are invertible matrices, then the equation $ABC = \mathbf{O}_{k,k}$ is equivalent to $B_{s,t} = \mathbf{O}_{s,t}$.

Proof. It is not hard to check that

$$AB = \left[\begin{array}{c|c} A_{s,s} & \mathbf{O}_{s,k-s} \\ \hline \mathbf{O}_{k-s,s} & \mathbf{O}_{k-s,k-s} \end{array}\right] \cdot \left[\begin{array}{c|c} B_{s,t} & X_{s,k-t} \\ \hline Y_{k-s,t} & Z_{k-s,k-t} \end{array}\right] = \left[\begin{array}{c|c} A_{s,s}B_{s,t} & A_{s,s}X_{s,k-t} \\ \hline \mathbf{O}_{k-s,t} & \mathbf{O}_{k-s,k-t} \end{array}\right],$$

and hence

$$(AB)C = \left[\begin{array}{c|c} A_{s,s}B_{s,t} & A_{s,s}X_{s,k-t} \\ \hline \mathbf{O}_{k-s,t} & \mathbf{O}_{k-s,k-t} \end{array}\right] \cdot \left[\begin{array}{c|c} C_{t,t} & \mathbf{O}_{t,k-t} \\ \hline \mathbf{O}_{k-t,t} & \mathbf{O}_{k-t,k-t} \end{array}\right] = \left[\begin{array}{c|c} A_{s,s}B_{s,t}C_{t,t} & \mathbf{O}_{s,k-t} \\ \hline \mathbf{O}_{k-s,t} & \mathbf{O}_{k-s,k-t} \end{array}\right].$$

So, if $B_{s,t} = \mathbf{O}_{s,t}$, then $ABC = \mathbf{O}_{k,k}$. Conversely, if $ABC = \mathbf{O}_{k,k}$, then $A_{s,s}B_{s,t}C_{t,t} = \mathbf{O}_{s,t}$. Using the fact that $A_{s,s}$ and $C_{t,t}$ are invertible matrices, we can multiply the equation $A_{s,s}B_{s,t}C_{t,t} = \mathbf{O}_{s,t}$ by $A_{s,s}^{-1}$ on the left and by $C_{t,t}^{-1}$ on the right to obtain that $B_{s,t} = \mathbf{O}_{s,t}$. ◀

▶ **Theorem 15.** *Let $\mathcal{F}$ be the ring of integers $\mathbb{Z}$ or one of the fields $\mathbb{Q}$, $\mathbf{A}$ or $\mathbf{A}_{\mathbb{R}}$. Then the ABC-Z problem for matrices from $\mathcal{F}^{k\times k}$ is Turing equivalent to the Skolem problem of depth k over $\mathcal{F}$.*

Proof. First, we show reduction from the ABC-Z problem to the Skolem problem.

Clearly, the ABC-Z problem over $\mathbb{Z}$ is equivalent to the ABC-Z problem over $\mathbb{Q}$ (by multiplying the matrices A, B, C by a suitable integer number). It is also not hard to see that the Skolem problem for 1-LRS over $\mathbb{Q}$ is equivalent to the Skolem problem over $\mathbb{Z}$ for 1-LRS of the same depth. Indeed, by Lemma 11 we can express any 1-LRS $(u_n)_{n=0}^{\infty}$ over $\mathbb{Q}$ as $u_n = \mathbf{u}^T M^n \mathbf{v}$ for some rational vectors $\mathbf{u}$ and $\mathbf{v}$ and a rational matrix M. If we multiply $\mathbf{u}$, $\mathbf{v}$ and M by a suitable natural number t, then $(t^{n+2}u_n)_{n=0}^{\infty}$ will be an integer 1-LRS, which has the same zero set as $(u_n)_{n=0}^{\infty}$. Hence, without loss of generality, we will assume that $\mathcal{F}$ is one of the fields $\mathbb{Q}$, $\mathbf{A}$ or $\mathbf{A}_{\mathbb{R}}$.

Consider an instance of the ABC-Z problem: $A^m B^n C^\ell = \mathbf{O}_{k,k}$, where $A, B, C \in \mathcal{F}^{k,k}$. Let $\chi_A(x)$ be the characteristic polynomial of A. By Proposition 3, we can find a primary decomposition $\chi_A(x) = p_1(x)^{m_1} \cdots p_t(x)^{m_t}$, where $p_1(x), \dots, p_t(x)$ are distinct irreducible monic polynomials. From this decomposition we can find the minimal polynomial $m_A(x)$ of A because $m_A(x)$ is a factor of $\chi_A(x)$, and we can check all divisors of $\chi_A(x)$ to find $m_A(x)$.

Let $m_A(x) = p_1(x)^{r_1} \cdots p_u(x)^{r_u}$, where $p_1(x), \dots, p_u(x)$ are distinct irreducible monic polynomials. Now we apply the Primary Decomposition Theorem (Theorem 2) to A. Let S_i be a basis for the null space of $p_i(A)^{r_i}$, which can be found, e.g., using Gaussian elimination. Let S be a matrix whose columns are the vectors of the basis $S_1 \cup \cdots \cup S_u$. Then

$$S^{-1}AS = A_1 \oplus \cdots \oplus A_u,$$

where the minimal polynomial of A_i is $p_i(A)^{r_i}$ for $i = 1, \dots, u$. Similarly, we can compute a primary decomposition $m_C(x) = q_1(x)^{s_1} \cdots q_v(x)^{s_v}$ of the minimal polynomial for C, where $q_1(x), \dots, q_v(x)$ are distinct irreducible monic polynomials, and a matrix T such that

$$T^{-1}CT = C_1 \oplus \cdots \oplus C_v,$$

where the minimal polynomial of C_i is $q_i(C)^{s_i}$ for $i = 1, \dots, v$.

Note that if $p(x)$ is an irreducible monic polynomial, then either $p(x) = x$ or x does not divide $p(x)$. So, among the polynomials $p_1(x), \dots, p_u(x)$ in the primary decomposition of $m_A(x)$ at most one is equal to x, and the same holds for the polynomials $q_1(x), \dots, q_v(x)$ in the primary decomposition of $m_C(x)$.

Suppose, for example, that $p_u(x) = x$. In this case $m_A(x) = p_1(x)^{r_1} \cdots p_{u-1}(x)^{r_{u-1}} x^{r_u}$, and $S^{-1}AS = A_1 \oplus \cdots \oplus A_{u-1} \oplus A_u$, where the minimal polynomial of A_u is x^{r_u}, and hence A_u is a nilpotent matrix of index r_u. Recall that, for $i = 1, \dots, u-1$, the polynomial $p_i(x)$ is not divisible by x, and so is $p_i(x)^{r_i}$, which is the minimal polynomial for A_i. Hence, by Proposition 4, A_i is invertible. Let $A_{\text{inv}} = A_1 \oplus \cdots \oplus A_{u-1}$ and $A_{\text{nil}} = A_u$. Then we obtain

$$S^{-1}AS = A_{\text{inv}} \oplus A_{\text{nil}}, \tag{1}$$

where A_{inv} is invertible, and A_{nil} is nilpotent. If $p_i(x) = x$ for some $i < u$, then we need in addition to permute some rows and columns of matrix S to obtain one that gives us Equation (1) above. If none of the $p_i(x)$ is equal to x, then we assume that A_{nil} is the empty matrix of size 0×0.

The same reasoning can be applied to matrix C, that is, we can compute an invertible matrix C_{inv}, a nilpotent (or empty) matrix C_{nil}, and an invertible matrix T such that

$$T^{-1}CT = C_{\text{inv}} \oplus C_{\text{nil}}.$$

Note that the indices of the nilpotent matrices A_{nil} and C_{nil} are at most k, and hence A^k_{nil} and C^k_{nil} are zero (or empty) matrices.

Our goal is to find all triples $(m, n, \ell) \in \mathbb{N}^3$ for which $A^m B^n C^\ell = \mathbf{O}_{k,k}$. In order to do this we will consider four cases: (1) $m \geq k$ and $\ell \geq k$, (2) $m < k$ and $\ell < k$, (3) $m \geq k$ and $\ell < k$, and (4) $m < k$ and $\ell \geq k$.

Before dealing with each of these cases, we note that the equation $A^m B^n C^\ell = \mathbf{O}_{k,k}$ is equivalent to

$$\begin{aligned} &S(A^m_{\text{inv}} \oplus A^m_{\text{nil}})S^{-1}B^nT(C^\ell_{\text{inv}} \oplus C^\ell_{\text{nil}})T^{-1} = \mathbf{O}_{k,k} \quad \text{or to} \\ &\ (A^m_{\text{inv}} \oplus A^m_{\text{nil}})S^{-1}B^nT(C^\ell_{\text{inv}} \oplus C^\ell_{\text{nil}}) = \mathbf{O}_{k,k} \end{aligned}$$

because S and T are invertible matrices.

Now suppose A_{inv} has size $s \times s$, and C_{inv} has size $t \times t$ for some $s, t \leq k$.

Case 1: $m \geq k$ and $\ell \geq k$. Since $m, \ell \geq k$, we have $A^m_{\text{nil}} = \mathbf{O}_{k-s,k-s}$ and $C^\ell_{\text{nil}} = \mathbf{O}_{k-t,k-t}$, and hence the equation $A^m B^n C^\ell = \mathbf{O}_{k,k}$ is equivalent to

$$(A^m_{\text{inv}} \oplus \mathbf{O}_{k-s,k-s})S^{-1}B^nT(C^\ell_{\text{inv}} \oplus \mathbf{O}_{k-t,k-t}) = \mathbf{O}_{k,k}. \tag{2}$$

Suppose the matrix $S^{-1}B^nT$ has a form $S^{-1}B^nT = \left[\begin{array}{c|c} B_{s,t} & X_{s,k-t} \\ \hline Y_{k-s,t} & Z_{k-s,k-t} \end{array}\right]$. Since A^m_{inv} and C^ℓ_{inv} are invertible matrices, Lemma 14 implies that Equation (2) is equivalent to $B_{s,t} = \mathbf{O}_{s,t}$. Therefore, we obtain the following equivalence: $A^m B^n C^\ell = \mathbf{O}_{k,k}$ if and only if

$$s^{i,j}_n = (\mathbf{e}_i^\top S^{-1})B^n(T\mathbf{e}_j) = 0 \quad \text{for all } i = 1, \dots, s \text{ and } j = 1, \dots, t. \tag{3}$$

By Lemma 11, the sequence $(s^{i,j}_n)^\infty_{n=0}$ is a 1-LRS of order k over $\mathcal{F}$. As in the proof of Theorem 10, we can use an oracle for the Skolem problem for 1-LRS of depth k over $\mathcal{F}$ to compute the descriptions of the semilinear sets $\mathcal{Z}(s^{i,j}_n)$. Hence we can compute a description of the intersection $Z_1 = \bigcap\limits_{\substack{i=1,\dots,s \\ j=1,\dots,t}} \mathcal{Z}(s^{i,j}_n)$, which is also a semilinear set. An important observation is that the set Z_1 does not depend on m and ℓ.

Below is a brief description of the remaining cases. The detailed proof of Cases 2, 3 and 4 can be found in the full version [6].

Case 2: $m < k$ and $\ell < k$. Fix some $m < k$ and $\ell < k$. For this particular choice of m and ℓ, we can compute a description of the semilinear set $Z_2(m, \ell)$ which is equal to all values of n for which $A^m B^n C^\ell = \mathbf{O}_{k,k}$ holds for fixed $m, \ell < k$.

Case 3: $m \geq k$ and $\ell < k$ and **Case 4: $m < k$ and $\ell \geq k$.** To solve these cases we will combine ideas from Cases 1 and 2. Namely, we can compute the descriptions of the semilinear sets $Z_3(\ell)$ and $Z_4(m)$ such that

- $Z_3(\ell) \;= \{\, n \;:\; A^m B^n C^\ell = \mathbf{O}_{k,k} \;\text{ for all } m \geq k\,\}$.
- $Z_4(m) = \{\, n \;:\; A^m B^n C^\ell = \mathbf{O}_{k,k} \;\text{ for all } \ell \geq k\,\}$.

Combining all the above cases together, we conclude that the set of all triples $(m, n, \ell) \in \mathbb{N}^3$ that satisfy the equation $A^m B^n C^\ell = \mathbf{O}_{k,k}$ is equal to the following union

$$\begin{aligned}\{(m,n,\ell) : n \in Z_1 \text{ and } m, \ell \geq k\} \bigcup \bigcup_{m,\ell<k} \{(m,n,\ell) : n \in Z_2(m,\ell)\} \bigcup \\ \bigcup_{\ell<k} \{(m,n,\ell) : n \in Z_3(\ell) \text{ and } m \geq k\} \bigcup \bigcup_{m<k} \{(m,n,\ell) : n \in Z_4(m) \text{ and } \ell \geq k\}.\end{aligned} \tag{4}$$

Having a description for the above set, we can decide whether is it empty or not, that is, whether there exist $m, n, \ell \in \mathbb{N}$ such that $A^m B^n C^\ell = \mathbf{O}_{k,k}$.

We now show the reduction in the other direction. Let $(u_n)_{n=0}^\infty$ be a 1-LRS that satisfies a relation

$$u_n = a_{k-1}u_{n-1} + \cdots + a_1 u_{n-k+1} + a_0 u_{n-k},$$

where $a_0 \neq 0$. Let A, B and C be the following matrices of size $k \times k$:

$$A = \begin{pmatrix} u_{k-1} & \cdots & u_1 & u_0 \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \end{pmatrix}, \; B = \begin{pmatrix} a_{k-1} & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_2 & 0 & \cdots & 1 & 0 \\ a_1 & 0 & \cdots & 0 & 1 \\ a_0 & 0 & \cdots & 0 & 0 \end{pmatrix}, \; C = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}.$$

A straightforward computation shows that the product $A^m B^n C^\ell$ is equal to a matrix where all entries equal zero except for the entry in the upper-right corner which is equal to $u_{k-1}^{m-1} u_n$. So, if we assume that $u_{k-1} \neq 0$, then we have the following implications: (1) if $A^m B^n C^\ell = \mathbf{O}_{k,k}$ for some $m, n, \ell \in \mathbb{N}$ with $m, \ell \geq 1$, then $u_n = 0$; and (2) if $u_n = 0$, then the equation $A^m B^n C^\ell = \mathbf{O}_{k,k}$ holds for any $m, \ell \geq 1$.

The assumption that $u_{k-1} \neq 0$ is not a serious restriction because we can shift the original sequence by at most k positions to ensure that $u_{k-1} \neq 0$. In other words, instead of $(u_n)_{n=0}^\infty$ we can consider a sequence $(u_{n+t})_{n=0}^\infty$ for some $t > 0$. It is easy to check that a 1-LRS of depth k is identically zero if and only if it contains k consecutive zeros. Hence if $(u_n)_{n=0}^\infty$ is not identically zero, then we can find $t < k$ such that the sequence $(u_{n+t})_{n=0}^\infty$ satisfies the condition that $u_{k-1+t} \neq 0$. ◀

▶ **Corollary 16.** *The set of triples (m, n, ℓ) that satisfy an equation $A^m B^n C^\ell = \mathbf{O}_{k,k}$ is equal to a finite union of direct products of semilinear sets.*

Proof. The corollary follows from Equation (4) above that describes all triples (m, n, ℓ) that satisfy the equation $A^m B^n C^\ell = \mathbf{O}_{k,k}$. By construction and the Skolem-Mahler-Lech theorem, the sets Z_1, $Z_2(m, \ell)$, $Z_3(\ell)$ and $Z_4(m)$ are semilinear. In Equation (4) we take direct product of these sets either with singleton sets or with sets of the form $\mathbb{N}_k = \{n \in \mathbb{N} : n \geq k\}$, which are also semilinear sets, and then take a finite union of such products. In other words, Equation (4) can be rewritten as follows

$$\mathbb{N}_k \times Z_1 \times \mathbb{N}_k \bigcup \bigcup_{m,\ell<k} \{m\} \times Z_2(m,\ell) \times \{\ell\} \bigcup \bigcup_{\ell<k} \mathbb{N}_k \times Z_3(\ell) \times \{\ell\} \bigcup \bigcup_{m<k} \{m\} \times Z_4(m) \times \mathbb{N}_k.$$

The main corollary of Theorem 15 is the following result. ◀

▶ **Corollary 17.** *The ABC-Z problem is decidable for* 3×3 *matrices over algebraic numbers and for* 4×4 *matrices over real algebraic numbers.*

Proof. By Theorem 15, the ABC-Z problem for 3×3 matrices over $\mathbf{A}$ is equivalent to the Skolem problem of depth 3 over $\mathbf{A}$, and the ABC-Z problem 4×4 matrices over $\mathbf{A}_{\mathbb{R}}$ is equivalent to the Skolem problem of depth 4 over $\mathbf{A}_{\mathbb{R}}$. Now the corollary follows from the fact that the Skolem problem is decidable for linear recurrence sequences of depth 3 over $\mathbf{A}$ and of depth 4 over $\mathbf{A}_{\mathbb{R}}$ [44, 33]. ◀

5 The ABCD-Z problem in dimension two

Recall that the ABCD-Z problem in dimension two asks whether there exist natural numbers $k, m, n, \ell \in \mathbb{N}$ such that

$$A^k B^m C^n D^\ell = \mathbf{O}_{2,2}. \tag{5}$$

In this section we will show that this problem is decidable for 2×2 upper-triangular matrices with rational coefficients. In the proof we will use the following simple lemmas which show how to diagonalise and compute powers of an upper-triangular 2×2 matrix.

▶ **Lemma 18.** *Given an upper triangular matrix* $\begin{pmatrix} a & b \\ 0 & c \end{pmatrix}$ *such that* $a \neq c$ *then:*

$$\begin{pmatrix} a & b \\ 0 & c \end{pmatrix} = \begin{pmatrix} 1 & -\frac{b}{a-c} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a & 0 \\ 0 & c \end{pmatrix} \begin{pmatrix} 1 & \frac{b}{a-c} \\ 0 & 1 \end{pmatrix}$$

▶ **Lemma 19.** *For any matrices of the form* $\begin{pmatrix} a & b \\ 0 & c \end{pmatrix}$ *and* $\begin{pmatrix} a & b \\ 0 & a \end{pmatrix}$, *such that* $a \neq c$, *and any* $k \in \mathbb{N}$ *we have* $\begin{pmatrix} a & b \\ 0 & c \end{pmatrix}^k = \begin{pmatrix} a^k & b\dfrac{a^k - c^k}{a - c} \\ 0 & c^k \end{pmatrix}$ *and* $\begin{pmatrix} a & b \\ 0 & a \end{pmatrix}^k = \begin{pmatrix} a^k & kba^{k-1} \\ 0 & a^k \end{pmatrix}$.

▶ **Theorem 20.** *The ABCD-Z problem is decidable for* 2×2 *upper-triangular matrices over rational numbers.*

Proof. First, note that if one of the matrices A, B, C or D is nilpotent, then Equation (5) obviously has a solution. So from now on we assume that none of A, B, C or D is nilpotent. Furthermore, if A or D is invertible, then the ABCD-Z problem reduces to the ABC-Z problem for rational matrices of dimension two that is decidable by Theorem 15. So, without loss of generality, we will assume that both A and D are singular matrices.

Now suppose we are given an instance of the ABCD-Z problem which satisfies the above-mentioned requirements. We will show that if Equation (5) has a solution, then it has a solution with $k = \ell = 1$. Indeed, assume Equation (5) has a solution, and let (k, m, n, ℓ) be a solution of minimal length, where the length of a solution is the sum $k + m + n + \ell$. Using

Theorem 15 we can exclude the case when $k = 0$ or $\ell = 0$ since in this case our problem is just an instance of the ABC-Z for 2×2 matrices. So we will assume that in the above solution $k, \ell \geq 1$.

By the Frobenius rank inequality (Theorem 1), we have that:

$$\mathsf{Rk}(A^k B^m C^n D^{\ell-1}) + \mathsf{Rk}(A^{k-1} B^m C^n D^{\ell}) \leq \mathsf{Rk}(A^{k-1} B^m C^n D^{\ell-1}).$$

This follows since $\mathsf{Rk}(A^k B^m C^n D^{\ell}) = \mathsf{Rk}(\mathbf{O}_{2,2}) = 0$. In the above inequality, notice that neither $A^k B^m C^n D^{\ell-1}$ nor $A^{k-1} B^m C^n D^{\ell}$ is the zero matrix by the assumption that the solution has minimal length. Hence the ranks of the matrices on the left hand side are at least 1. Therefore, $\mathsf{Rk}(A^{k-1} B^m C^n D^{\ell-1}) = 2$. Since we assumed that A and D are singular, it is necessary that $k = \ell = 1$. Also notice that if $\mathsf{Rk}(B) = 1$ or $\mathsf{Rk}(C) = 1$, then we must have $m = 0$ or $n = 0$, respectively. Again these cases can be excluded by Theorem 15.

Thus, using the Frobenius rank inequality and the assumption that the solution is of minimal length, we reduced the ABCD-Z problem to an equation of the form:

$$AB^m C^n D = \mathbf{O}_{2,2},$$

where $\mathsf{Rk}(A) = \mathsf{Rk}(D) = 1$ and $\mathsf{Rk}(B) = \mathsf{Rk}(C) = 2$.

We assumed that A and D have rank one but are not nilpotent. This means that they have one zero and one nonzero element on the diagonal, in particular, they satisfy the condition of Lemma 18. Hence we can find invertible rational matrices S_A and S_D such that

$$A = S_A^{-1} \begin{pmatrix} a & 0 \\ 0 & 0 \end{pmatrix} S_A \quad \text{and} \quad D = S_D^{-1} \begin{pmatrix} d & 0 \\ 0 & 0 \end{pmatrix} S_D,$$

where a, d are nonzero rational numbers. Applying Lemma 14 with matrix $S_A B^m C^n S_D^{-1}$ in place of B, we conclude that $AB^m C^n D = \mathbf{O}_{2,2}$ holds if and only if the $(1,1)$-entry of $S_A B^m C^n S_D^{-1}$ equals zero. In other words, the equation $AB^m C^n D = \mathbf{O}_{2,2}$ is equivalent to $s_{m,n} = \mathbf{u}^\top B^m C^n \mathbf{v} = 0$, where $\mathbf{u}^\top = \mathbf{e}_1^\top S_A$ and $\mathbf{v} = S_D^{-1} \mathbf{e}_1$ are vectors with rational coefficients.

We will consider three cases: (1) both B and C have distinct eigenvalues, (2) both B and C have a single eigenvalue of multiplicity 2 and (3) one matrix has distinct eigenvalues and the other has a single eigenvalue of multiplicity 2.

Case (1): B and C have distinct eigenvalues, that is, $B = \begin{pmatrix} b_1 & b_3 \\ 0 & b_2 \end{pmatrix}$ and $C = \begin{pmatrix} c_1 & c_3 \\ 0 & c_2 \end{pmatrix}$, where $b_1 \neq b_2$ and $c_1 \neq c_2$. By Lemma 19 we have

$$B^m = \begin{pmatrix} b_1^m & b_3 \dfrac{b_1^m - b_2^m}{b_1 - b_2} \\ 0 & b_2^m \end{pmatrix} \quad \text{and} \quad C^n = \begin{pmatrix} c_1^n & c_3 \dfrac{c_1^n - c_2^n}{c_1 - c_2} \\ 0 & c_2^n \end{pmatrix}$$

Multiplying these matrices we obtain: $B^m C^n =$

$$= \begin{pmatrix} b_1^m & b_3 \dfrac{b_1^m - b_2^m}{b_1 - b_2} \\ 0 & b_2^m \end{pmatrix} \begin{pmatrix} c_1^n & c_3 \dfrac{c_1^n - c_2^n}{c_1 - c_2} \\ 0 & c_2^n \end{pmatrix} = \begin{pmatrix} b_1^m c_1^n & b_3 \dfrac{b_1^m c_2^n - b_2^m c_2^n}{b_1 - b_2} + c_3 \dfrac{b_1^m c_1^n - b_1^m c_2^n}{c_1 - c_2} \\ 0 & b_2^m c_2^n \end{pmatrix}$$

From this formula one can see that the entries of $B^m C^n$ are linear combinations of $b_1^m c_1^n$, $b_1^m c_2^n$ and $b_2^m c_2^n$ with rational coefficients. Notice that the term $b_2^m c_1^n$ does not appear in the entries of $B^m C^n$. Since $\mathbf{u}$ and $\mathbf{v}$ are rational vectors we conclude that

$$s_{m,n} = \mathbf{u}^\top B^m C^n \mathbf{v} = \alpha b_1^m c_1^n + \beta b_1^m c_2^n + \gamma b_2^m c_2^n,$$

where $\alpha, \beta, \gamma \in \mathbb{Q}$. Multiplying the equation $s_{m,n} = \alpha b_1^m c_1^n + \beta b_1^m c_2^n + \gamma b_2^m c_2^n = 0$ by the product of denominators of α, β, γ and b_1, b_2, c_1, c_2 we can rewrite it in the following form

$$as^m r^n + bs^m t^n + cq^m t^n = 0$$

where a, b, c, q, r, s, t are integers and a, b, c are relatively prime. Recall that we want to find out if there exist $m, n \in \mathbb{N}$ that satisfy the above exponential Diophantine equation. If one of the coefficients a, b, c is zero, then this problem is easy to solve. For instance, if $b = 0$ then the above equation is equivalent to $as^m r^n = -cq^m t^n$. This equality holds if and only if $as^m r^n$ and $-cq^m t^n$ have the same sign and $v_p(as^m r^n) = v_p(-cq^m t^n)$ for every prime divisor p of a, c, s, r, q or t. Each of these conditions can be expressed as a linear Diophantine equation. For instance, the sign of $as^m r^n$ or $-cq^m t^n$ depends on the parity of m and n. So, the requirement that $as^m r^n$ and $-cq^m t^n$ have the same sign can be written as a linear congruence equation in m and n modulo 2, which in turn can be expressed as a linear Diophantine equation. Since a system of linear Diophantine equations can be effectively solved, we can find out whether there exist m and n that satisfy $as^m r^n = -cq^m t^n$.

Now suppose that a, b, c are relatively prime nonzero integers. Let T be all primes that appear in s, r, q or t. Theorem 5 gives an upper bound on nonzero relatively prime integers x, y, z that are composed of the primes from T and satisfy the equation $ax + by + cz = 0$. Therefore, we can algorithmically compute the following set

$$\mathcal{U} = \{(x, y, z) \ : \ ax + by + cz = 0, \ x, y, z \neq 0 \text{ and } \gcd(x, y, z) = 1\}.$$

Next for each triple $(x, y, z) \in \mathcal{U}$ we want to find out if there exist $m, n \in \mathbb{N}$ such that

$$(s^m r^n, s^m t^n, q^m t^n) = (xg, yg, zg) \tag{6}$$

for some $g \in \mathbb{N}$ that is composed of the primes from T. It is not hard to see that Equation (6) holds if and only if for every $p \in T$

$$v_p(s^m r^n) - v_p(x) = v_p(s^m t^n) - v_p(y) = v_p(q^m t^n) - v_p(z)$$

and $s^m r^n, s^m t^n, q^m t^n$ have the same signs as x, y, z, respectively. Since these conditions can be expressed as a system of linear Diophantine equations, we can algorithmically find if there are $m, n \in \mathbb{N}$ that satisfy Equation (6). If such m and n exist for at least one triple $(x, y, z) \in \mathcal{U}$, then the original equation $s_{m,n} = 0$ has a solution. Otherwise, the equation $s_{m,n} = 0$ does not have a solution.

To finish the proof we also need to consider the following cases:

Case (2): both B and C have a single eigenvalue of multiplicity 2.

Case (3): one matrix has distinct eigenvalues and the other has a single eigenvalue of multiplicity 2.

Case (2) is the easiest one in the sense that in this case we can reduce the ABCD-Z problem to a single linear Diophantine equation without using Baker's theorem. In Case (3) we will reduce the ABCD-Z problem to a linear-exponential Diophantine equation of the form $c\frac{s^m}{t^m} = a + bn$, where $a, b, c, s, t \in \mathbb{Z}$, $t > 0$ and $\gcd(s, t) = 1$, which is not hard solve. Again, this case does not require the use of Baker's theorem. All the necessary details for the proof of Cases (2) and (3) are given in the full version [6]. ◀

▶ Remark. It is interesting to note that in Cases (1) and (2) the solutions (m, n) of the equation $s_{m,n} = 0$ are described by linear Diophantine equations, and only in Case (3) we have a linear-exponential equation. This agrees with an example from Proposition 23, in which matrix A has a single eigenvalue 1 of multiplicity 2 and matrix B has distinct eigenvalues 1 and 2.

▶ Remark. In the above argument we used Theorem 5 to obtain a bound on the solutions of the equation $as^m r^n + bs^m t^n + cq^m t^n = 0$, which is a special type of an S-unit equation. This leaves open an interesting question of whether any S-unit equation can be encoded into the ABCD-Z problem.

The obvious question is how hard would it be to solve n-LRSs, or in general multiplicative matrix equations, in low dimensions. In fact we can show that the Skolem problem for n-LRSs of depth 2 is NP-hard. It is not direct but an easy corollary following the hardness proof of the mortality problem for 2 × 2 matrices [4].

▶ **Theorem 21** ([4]). *The mortality problem for integer matrices of dimension two is NP-hard.*

▶ **Corollary 22.** *Determining if the zero set of an n-LRS of depth 2 is empty is NP-hard.*

Proof. The main idea of the proof is to reuse the NP-hardness result for the mortality problem in 2 × 2 integral matrices from [4] by replacing the simulation of alternatives in subset sum problem defined by a state structure (which is guaranteed by specific order of unique cancellations) by the equation structure in which avoiding alternatives corresponds to the elements in the equation with zero power. See the formal construction in the full version [6]. ◀

Another interesting observation is that the zero set of a 2-LRS is not necessarily semilinear, in contrast to the situation for 1-LRSs, which indicates that the Skolem problem for 2-LRSs is likely to be significantly harder than the Skolem problem for 1-LRSs even for sequences of small depth.

▶ **Proposition 23.** *There exists a 2-LRS of depth 2 for which the zero set is not semilinear.*

Proof. Let $u = (0,1)^T, v = (1,-1)^T$, $A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $B = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$. Define

$$s_{n,m} = u^T A^n B^m v = (0,1) \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}^m \begin{pmatrix} 1 \\ -1 \end{pmatrix} = n - 2^m.$$

Then $s_{n,m} = 0$ if and only if $n = 2^m$. Clearly, the zero set is not semilinear. ◀

References

1 L. Babai, R. Beals, J-Y. Cai, G. Ivanyos, and E. M. Luks. Multiplicative equations over commuting matrices. In *Proc. of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 96, 1996.

2 C. Baier, S. Kiefer, J. Klein, S. Klüppelholz, D. Müller, and J. Worrell. Markov Chains and Unambiguous Büchi Automata. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, pages 23–42, 2016.

3 P. C. Bell, V. Halava, T. Harju, J. Karhumäki, and I. Potapov. Matrix equations and Hilbert's tenth problem. *International Journal of Algebra and Computation*, 18:1231–1241, 2008.

4 P. C. Bell, M. Hirvensalo, and I. Potapov. Mortality for 2 × 2 matrices is NP-hard. In *Mathematical Foundations of Computer Science (MFCS 2012)*, volume LNCS 7464, pages 148–159, 2012.

5 P. C. Bell, M. Hirvensalo, and I. Potapov. The identity problem for matrix semigroups in $SL_2(\mathbb{Z})$ is NP-complete. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*, pages 187–206, 2017.

6 Paul C. Bell, Igor Potapov, and Pavel Semukhin. On the Mortality Problem: from multiplicative matrix equations to linear recurrence sequences and beyond. *CoRR*, abs/1902.10188, 2019. `arXiv:1902.10188`.

7 V. Blondel, E. Jeandel, P. Koiran, and N. Portier. Decidable and undecidable problems about quantum automata. *SIAM Journal on Computing*, 34:6:1464–1473, 2005.

8 V. D. Blondel, O. Bournez, P. Koiran, C. Papadimitriou, and J. N. Tsitsiklis. Deciding stability and mortality of piecewise affine dynamical systems. *Theoretical Computer Science*, 255(1-2):687–696, 2001.

9 V. D. Blondel and N. Portier. The presence of a zero in an integer linear recurrent sequence is NP-hard to decide. *Linear Algebra and its Applications*, pages 91–98, 2002.

10 V. D. Blondel and J. N. Tsitsiklis. Complexity of stability and controllability of elementary hybrid systems. *Automatica*, 35:479–489, 1999.

11 Jin-yi Cai, Wolfgang H. J. Fuchs, Dexter Kozen, and Zicheng Liu. Efficient Average-Case Algorithms for the Modular Group. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 143–152, 1994.

12 Jin-yi Cai, Richard J. Lipton, and Yechezkel Zalcstein. The Complexity of the Membership Problem for 2-generated Commutative Semigroups of Rational Matrices. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 135–142, 1994.

13 J. Cassaigne, V. Halava, T. Harju, and F. Nicolas. Tighter Undecidability Bounds for Matrix Mortality, Zero-in-the-Corner Problems, and More. *CoRR*, abs/1404.0644, 2014.

14 V. Chonev, J. Ouaknine, and J. Worrell. On the Complexity of the Orbit Problem. *Journal of the ACM*, 63(3):1–18, 2016.

15 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the Skolem Problem for Continuous Linear Dynamical Systems. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 100:1–100:13, 2016.

16 J.-H. Evertse, K. Győry, C. L. Stewart, and R. Tijdeman. *S*-unit equations and their applications. In *New advances in transcendence theory (Durham, 1986)*, pages 110–174. Cambridge Univ. Press, Cambridge, 1988.

17 E. Galby, J. Ouaknine, and J. Worrell. On matrix powering in low dimensions. In *32nd International Symposium on Theoretical Aspects of Computer Science (STACS'15)*, pages 329–340, 2015.

18 K. Győry. On the *abc* conjecture in algebraic number fields. *Acta Arith.*, 133(3):281–295, 2008.

19 V. Halava, T. Harju, M. Hirvensalo, and J. Karhumäki. Skolem's problem - on the border between decidability and undecidability. In *TUCS Technical Report Number 683*, 2005.

20 G. Hansel. Une démonstration simple du théorème de Skolem-Mahler-Lech. *Theoret. Comput. Sci.*, 43(1):91–98, 1986.

21 Michael A. Harrison. *Lectures on Linear Sequential Machines.* Academic Press, Inc., Orlando, FL, USA, 1969.

22 K. Hoffman and R. Kunze. *Linear algebra.* Second edition. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1971.

23 Juha Honkala. Products of matrices and recursively enumerable sets. *Journal of Computer and System Sciences*, 81(2):468–472, 2015.

24 R. Horn and C. Johnson. *Matrix Analysis.* Cambridge University Press, 1990.

25 Ravindran Kannan and Richard J. Lipton. The Orbit Problem is Decidable. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 252–261, New York, NY, USA, 1980. ACM.

26 J.-Y. Kao, N. Rampersad, and J. Shallit. On NFAs where all states are final, initial, or both. *Theor. Comput. Sci.*, 410(47-49):5010–5021, 2009.

27 S.-K. Ko, R. Niskanen, and I. Potapov. On the Identity Problem for the Special Linear Group and the Heisenberg Group. In *45th International Colloquium on Automata, Languages, and*

Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic, pages 132:1–132:15, 2018.

28 Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack and subset sum problems in nilpotent, polycyclic, and co-context-free groups. *CoRR*, abs/1507.05145, 2015. `arXiv:1507.05145`.

29 C. Lech. A note on recurring series. *Ark. Mat. 2*, 1953.

30 Alexei Lisitsa and Igor Potapov. Membership and Reachability Problems for Row-Monomial Transformations. In Jiří Fiala, Václav Koubek, and Jan Kratochvíl, editors, *Mathematical Foundations of Computer Science 2004*, pages 623–634, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

31 Markus Lohrey. Rational subsets of unitriangular groups. *IJAC*, 25(1-2):113–122, 2015.

32 K. Mahler. Eine arithmetische Eigenschaft der Taylor-Koeffizienten rationaler Funktionen. In *Akad. Wet. Amsterdam 38*, pages 50–60, 1935.

33 M. Mignotte, T. N. Shorey, and R. Tijdeman. The distance between terms of an algebraic recurrence sequence. *J. Reine Angew. Math.*, 349:63–76, 1984.

34 C. Nuccio and E. 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`.

35 J. Ouaknine, J. Sousa Pinto, and J. Worrell. On termination of integer linear loops. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 957–969, 2015.

36 J. Ouaknine, A. Pouly, J. Sousa-Pinto, and J. Worrell. Solvability of Matrix-Exponential Equations. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'16)*, pages 798–806, 2016.

37 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.

38 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.

39 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.

40 M. S. Paterson. Unsolvability in 3×3 matrices. *Studies in Applied Mathematics*, 49(1):105–107, 1970.

41 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.

42 Igor Potapov and Pavel Semukhin. Membership Problem in GL(2, Z) Extended by Singular Matrices. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 44:1–44:13, 2017.

43 T. Skolem. Ein Verfahren zur Behandlung gewisser exponentialer Gleichungen und diophantischer Gleichungen. *Skand. Mat. Kongr.*, 8:163–188, 1934.

44 N. K. Vereshchagin. The problem of the appearance of a zero in a linear recursive sequence. *Mat. Zametki 38*, 347(2):609–615, 1985.